Računalništvo (fiziki): 3. izpit

28. 8. 2015

Naloge rešujte na strežniku Tomo (`https://tomo.fmf.uni-lj.si/`). V sistem se prijavite s svojim uporabniškim računom na omrežju FMF.
Čas reševanja je 150 minut. Doseženih 100 točk šteje za maksimalno oceno. Veliko uspeha!

1. naloga (Zima prihaja, 30 točk)

Vsi dobro vemo, da v Sedmih kraljestvih zime in poletja trajajo po več let skupaj. Veliki moister Pycelle bi rad preveril domnevo, da dolgemu in vročemu poletju sledi dolga in mrzla zima. Zbral je podatke o povprečnih letnih temperaturah za zelo dolgo obdobje. Ker mu programiranje ne gre, potrebuje vašo pomoč. Pripraviti morate nekaj funkcij, ki mu bodo v pomoč pri analizi podatkov.

a) (15 točk) Sestavite funkcijo `povprecje_petih`(l, i), ki kot argumenta dobi seznam l in indeks i. V seznamu l so shranjene letne temperature za neko obdobje. Funkcija naj vrne povprečno temperaturo za 5 zaporednih let od i-tega do $(i+4)$-tega. Če ni dovolj podatkov, naj funkcija vrne `None`. Zgled:

```
>>> t = [20, 22, 21, 23, 26, 30, 33, 32, 10, -8, -10, -5, -20, -10]
>>> povprecje_petih(t, 4)
26.2
>>> povprecje_petih(t, 10)
None
```

b) (15 točk) Sestavite funkcijo `vroce_poletje`(l), ki kot argument dobi seznam l z letnimi temperaturami in vrne par dveh števil:

- prvo število je indeks tistega leta, ko se je začelo najtoplejše obdobje petih zaporednih let,
- drugo število je povprečna temperatura tega obdobja.

Zagotovljeno je, da bo seznam l dolžine 5 ali več. Če obstaja več obdobij, kjer je dosežen maksimum povprečja, poiščite tistega, ki se je prej začelo (tj. kjer je indeks začetka najmanjši). Zgled:

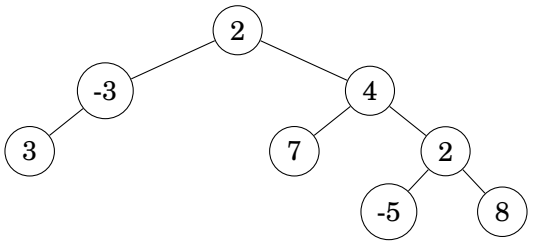
```
>>> t = [20, 22, 21, 23, 26, 30, 33, 32, 10, -8, -10, -5, -20, -10]
>>> vroce_poletje(t)
(3, 28.8)
```

2. naloga (Dvojiško drevo, 30 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(2, levo=Drevo(-3, levo=Drevo(3)), desno=Drevo(4, levo=Drevo(7),
        desno=Drevo(2, levo=Drevo(-5), desno=Drevo(8))))
```

Pripadajoča slika:



Delno implementiran razred `Drevo` najdete na strežniku. 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 `plitev_max(self, l)`, ki vrne vrednost največjega elementa v poddrevesu, pri čemer upošteva samo vozlišča do globine `l`. Koren drevesa ima globino 0. Za prazno drevo naj metoda vrne `None`. Zgledi:

```
>>> s.plitev_max(1)
4
>>> s.plitev_max(2)
7
>>> s.plitev_max(3)
8
```

b) (10 točk) Razredu `Drevo` dodajte metodo `kje_je_max(self)`, ki vrne seznam sestavljen iz znakov `'l'` in `'d'` (npr. `['d', 'd', 'l', 'l', 'd']`), ki pove, kako v drevesu iz korena pridemo do največjega elementa. Prazen seznam pomeni, da je največji element v korenu drevesa. Znak `'l'` pomeni, da se moramo spustiti v levo poddrevo, znak `'d'` pa, da moramo v desno poddrevo. Primer:

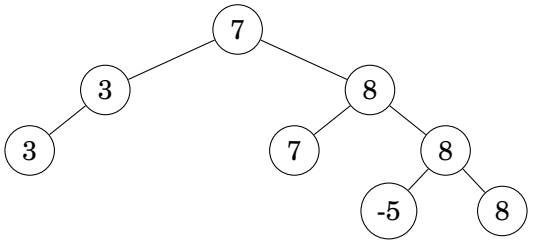
```
>>> s.kje_je_max()
['d', 'd', 'd']
```

Če metodo pokličemo na praznem drevesu, naj vrne `None`. V primeru, ko je v drevesu več maksimalnih elementov, ima prednost najprej tisti v korenu, nato tisti v levem poddrevesu in nato tisti v desnem poddrevesu.

c) (10 točk) Razredu `Drevo` dodajte metodo `plitvinsko_drevo(self, l)`, ki sestavi in vrne novo drevo, ki je enake oblike kot `self`. V vsakem vozlišču naj ima novo drevo shranjen največji element v pripadajočem poddrevesu drevesa `self`, pri čemer upoštevamo samo vozlišča do globine `l`. Primer:

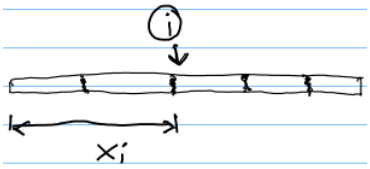
```
>>> s.plitvinsko_drevo(2)
Drevo(7, levo=Drevo(3, levo=Drevo(3), desno=Drevo()), desno=Drevo(8,
levo=Drevo(7), desno=Drevo(8, levo=Drevo(-5), desno=Drevo(8))))
```

Pripadajoča slika:



3. naloga (Pretep s palico, 25 točk)

Andrej uči Nina programirati tako, da ga za vsako napako udari s palico. Palica ima več šibkih mest, na katerih se prelomi ob udarcu. Andrej lahko udarec izvede z mojstrsko natančnostjo, tako da se palica prelomi na tistem šibkem mestu, ki si ga zamisli vnaprej. Odlomljene koščke palic Andrej pobere in z njimi naprej tepe Nina. Ustavi se šele takrat, ko palico razlomi do konca (ni več nobenega koščka, ki bi imel kakšno šibko mesto). Pri vsakem udarcu je prizadejana bolečina sorazmerna z dolžino palice (1 cm ustreza 1 enoti bolečine).



Primer: Denimo, da ima Andrej 100 cm dolgo palico, ki ima šibka mesta na razdaljah 10 cm, 25 cm, 60 cm in 90 cm od levega roba. Prvi udarec povzroči 100 enot bolečine in Andrej ga izvede tako, da se palica razlomi na šibki točki, ki je od levega roba oddaljena 90 cm. Palica razpade na dva kosa, prvi je dolg 90 cm in drugi le 10 cm ter je neuporaben, saj nima več šibkih mest. Z uporabnim kosom Andrej ponovno udari, tako da se ta prelomi 25 cm od levega roba. S tem povzroči 90 enot bolečine, palica pa razpade na dva kosa; prvi je dolg 25 cm, drugi pa 65 cm. Ker vsak kos palice vsebuje po eno šibko mesto, lahko z vsakim še enkrat useka, s čimer povzroči 25 in 65 enot bolečine.

a) (10 točk) Sestavite funkcijo `mlatenje`(n,x,p), ki dobi naslednje tri argumente:

- število n predstavlja dolžino palice;
- seznam $x = [x_0, x_1, \ldots, x_{k-1}]$ so položaji šibkih točk, merjeno od levega roba, pri čemer je $0 < x_0 < x_1 < \cdots < x_{k-1} < n$;
- seznam p vsebuje števila med 0 in $k-1$ (vsako se pojavi natančno enkrat) in predstavlja vrstni red lomljenja palice.

Funkcija naj vrne seznam k števil in sicer za vsak udarec količino bolečine, ki je prizadejana. Zgled:

```
>>> mlatenje(100, [10, 25, 60, 90], [3, 1, 0, 2])
[100, 90, 25, 65]
```

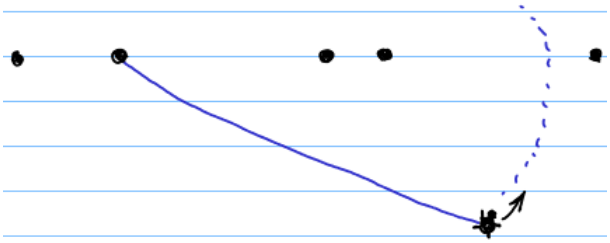
b) (15 točk) Andrej želi tepsti Nina tako, da bo skupna količina prizadejane bolečine čim večja. Napišite funkcijo `bolecina`(n,x), ki vrne skupno količino bolečine, če se Andrej pretepanja loti optimalno. Algoritem naj ima časovno zahtevnost $O(k^2)$, kjer je k dolžina seznama x.

```
>>> bolecina(100, [10, 25, 60, 90])
335
```

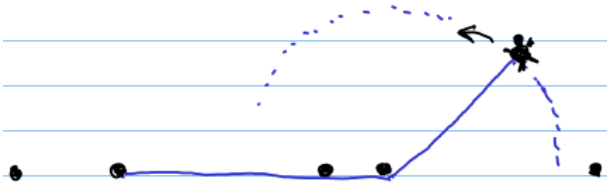
4. naloga (Kužek, 25 točk)

Na nekem velikem travniku je v ravno vrsto postavljenih n drogov z zastavami, ki si jih lahko predstavljamo kot točke na premici, če jih pogledamo s ptičje perspektive. Koordinatni sistem postavimo tako, da so vsi drogovi na osi x.

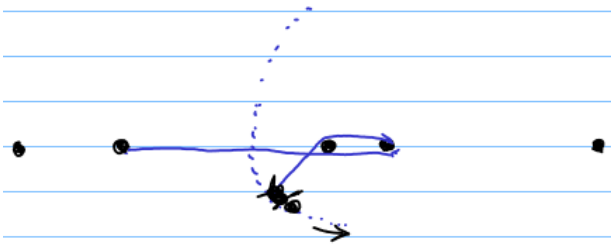
Na i-ti drog privežemo zanemarljivo majhnega kužka, ki je na l metrov dolgi vrvici, pri čemer je vrvica napeta. Kužek začne teči v nasprotni smeri urinega kazalca, tako da je vrvica ves čas napeta. Recimo, da kužek začne tek na spodnji polravnini. Tekel bo po krožnem loku, dokler ne bo prečkal osi x in prišel na zgornjo polravnino:



Vrvica se bo pričela ovijati okoli nekega drugega droga, kužek pa bo tekel po nekem manjšem krožnem loku, dokler ne bo spet prestopil na spodnjo polravnino:



Tek se konča tako, da se kužek prične ovijati okoli nekega droga (ker so ostali predaleč, da bi jih dosegel):



Če se vrvica ravno prav dolga, da pride do nekega droga, se šteje, da se tek konča tam.

a) (15 točk) Napišite funkcijo `naslednji_drog`(x,s,l,p), kjer je $x = [x_0, x_1, \ldots, x_{n-1}]$ seznam x-koordinat drogov, pri čemer velja $x_i < x_{i+1}$; s je zaporedna številka droga, za katerega privežemo kužka (najbolj levi drog ima indeks 0); l je dolžina vrvice; p je logična vrednost `True`, če kužek začne tek na spodnji polravnini in `False`, če prične na zgornji. Funkcija naj vrne zaporedno številko droga, ki bo postal središče novega krožnega loka, ko bo prestopil na nasprotno polravnino. Njena časovna zahtevnost naj bo $O(\log n)$.

```
>>> naslednji_drog([0, 2, 7, 8, 12], 1, 9, True)
3
>>> naslednji_drog([0, 2, 7, 8, 12], 1, 9, False)
0
```

b) (10 točk) Napišite funkcijo `konec_teka`(x,s,l,p), pri čemer so argumenti takšni kot v prejšnji nalogi. Funkcija naj vrne zaporedno številko droga, kjer bo kužek končal (okrog katerega se bo ovil). Časovna zahtevnost funkcije naj bo $O(n \log n)$.

```
>>> konec_teka([0, 2, 7, 8, 12], 1, 9, True)
2
```