resitev

January 28, 2024

1 Ne maram lihih števil

Vse te naloge je potrebno reševati z rekurzivnimi funkcijami. Nobenih zank `for`, nobenih izpeljanih seznamov. Testi bodo to (kolikor toliko) preverjali.

Funkcije iz ogrevalnega dela niso težje od teh iz obveznega, razlika je le, da vam v ogrevalnem že povem rekurzivno definicijo naloge, ki ustreza, temu, kar morate sprogramirati. Samo prevedete iz slovenščine v Python. :) Obenem pa funkcije iz ogrevalnega malo spominjajo, uvajajo v tiste iz obveznega.

1.1 Ogrevanje

- Napiši funkcijo `vsebuja_sama_soda(s)`, ki vrne `True`, če podani seznam števil vsebuje sama soda števila.

  Spomnimo se: seznam vsebuje sama soda števila, če je prazen ali pa je prvo število sodo in vsebuje tudi preostanek sama soda števila.

- Napiši funkcijo `obrni(s)`, ki prejme seznam števil in vrne seznam, ki vsebuje iste elemente, a v obratnem vrstnem redu.

  Seznam je najpreprosteje obrniti tako, da daš prvo število na konec obrnjenega seznama. Točneje, sešteješ seznam, ki vsebuje vsa ostala števila, vendar obrnjena, in seznam, ki vsebuje prvo število.

  Klic `obrni([5, 3, 8])` vrne `[8, 3, 5]`.

- Napiši funkcijo `samo_soda(s)`, ki prejme seznam števil in vrne seznam, ki vsebuje samo soda števila iz tega seznama.

  Očitno je potrebno vrniti seznam, ki v primeru, da je prvo število sodo, vrne to število in še vsa ostala, ki so soda. Če prvo število ni sodo, pa vrne soda števila iz ostanka.

  Klic `samo_soda([5, 3, 4, 8, 5, 6, 2])` vrne `[4, 8, 6, 2]`.

1.1.1 Rešitev

`vsebuje_sama_soda` … če je prazen ali pa je prvo število sodo in vsebuje tudi preostanek sama soda števila.

[1]:
```
def vsebuje_sama_soda(s):
    return s == [] or s[0] % 2 == 0 and vsebuje_sama_soda(s[1:])
```

[2]:
```
vsebuje_sama_soda([4, 6, 2, 12])
```

[2]:
```
True
```

[3]:
```
vsebuje_sama_soda([4, 6, 3, 2, 12])
```

[3]:
```
False
```

obrni

[4]:
```
def obrni(s):
    if s == []:
        return s
    else:
        return obrni(s[1:]) + [s[0]]
```

[5]:
```
obrni(["Ana", "Berta", "Cilka", "Dani"])
```

[5]:
```
['Dani', 'Cilka', 'Berta', 'Ana']
```

K seznamu moramo prišteti seznam. To mora biti seznam, ki vsebuje prvi element, torej `[s[0]]`. Lahko pa pišemo tudi `s[:1]`. Če ob tem še zamenjamo `if s == []` z `if not s`, ne bo funkcija nikjer več predpostavljala, da je `s` seznam.

[6]:
```
def obrni(s):
    if not s:
        return s
    else:
        return obrni(s[1:]) + s[:1]
```

[7]:
```
obrni(["Ana", "Berta", "Cilka", "Dani"])
```

[7]:
```
['Dani', 'Cilka', 'Berta', 'Ana']
```

[8]:
```
obrni((1, 2, 3, 4, 5))
```

[8]:
```
(5, 4, 3, 2, 1)
```

[9]:
```
obrni("janez demšar")
```

[9]:
```
'rašmed zenaj'
```

samo_soda Spet so že navodila povedala vse.

[10]:
```
def samo_soda(s):
    if not s:
        return s
    ostala_soda = samo_soda(s[1:])
```

```
    if s[0] % 2 == 0:
        return [s[0]] + ostala_soda
    else:
        return ostala_soda
```

[11]:
```
samo_soda([5, 3, 4, 8, 5, 6, 2])
```

[11]:
```
[4, 8, 6, 2]
```

1.2 Obvezni del

- Napiši funkcijo `prestej_liha(s)`, ki vrne število lihih števil v podanem seznamu števil.

  Klic `prestej_liha([5, 3, 4, 8, 5, 6, 2])` vrne `3`.

- Napiši funkcijo `ena_vec(s)`, ki prejme seznam števil in vrne seznam, v katerem je vsako od števil za ena večje kot v podanem seznamu.

  Klic `ena_vec([5, 2, 8])` vrne `[6, 3, 9]`.

- In zdaj naš končni cilj: napiši funkcijo `zbij_liho(s)`, ki prejme seznam števil in vrne nov seznam z enakimi elementi, le vsem lihim številom odštejemo 1, da postanejo soda.

  Klic `zbij_liha([5, 2, 8, 7, 1, 6])` vrne `[4, 2, 8, 6, 0, 6]`.

1.2.1 Rešitev

`prestej_liha` Če je seznam prazen, vsebuje 0 lihih števil. Sicer pa vsebuje toliko lihih števil, kot jih je v preostanku (brez prvega), in še eno več, če je tudi prvo liho.

[12]:
```
def prestej_liha(s):
    if not s:
        return 0
    lihih = prestej_liha(s[1:])
    if s[0] % 2 == 1:
        lihih += 1
    return lihih
```

[13]:
```
prestej_liha([5, 3, 4, 8, 5, 6, 2])
```

[13]:
```
3
```

Zazrevši se v program opazimo, da k `lihih` prištejemo `1`, takrat ko je `s[0] % 2` enak `1`. Ko ni enak `1`, pa je enak `0` in takrat prištejemo `0`. Zategadelj lahko k `lihih` prištejemo, preprosto, `s[0] % 2`, ki je pač kolikorsižebodi.

[14]:
```
def prestej_liha(s):
    if not s:
        return 0
    return s[0] % 2 + prestej_liha(s[1:])
```

[15]:
```
prestej_liha([5, 3, 4, 8, 5, 6, 2])
```

[15]:
```
3
```

ena_vec Ako je seznam prazen, tu ni kaj: tak bode tudi ostal. Sicer pa sestavimo seznam tako, da k seznamu, ki vsebuje prvi element, povečan za 1, prištejemo seznam, ki vsebuje ostale elemente, povečane za 1.

[16]:
```
def ena_vec(s):
    if not s:
        return s
    return [s[0] + 1] + ena_vec(s[1:])
```

[17]:
```
ena_vec([5, 3, 4, 8, 5, 6, 2])
```

[17]:
```
[6, 4, 5, 9, 6, 7, 3]
```

zbij_liha Ta funkcija pa je zmes vsega, kar smo počeli doslej. Če je prazen, vrnemo prazen seznam. Sicer pa prvega zmanjšamo za 1, če je njegov ostanek po deljenju z 1 enak 1. Če je enak 0, pa ga pustimo pri miru, ali pa se pohecamo in ga zmanjšamo za `0`. Hec je v tem primeru na mestu, ker skrajša program: število zmanjšamo za ostanek tega števila po deljenju z 2.

[18]:
```
def zbij_liha(s):
    if not s:
        return s
    return [s[0] - (s[0] % 2)] + zbij_liha(s[1:])
```

[19]:
```
zbij_liha([5, 3, 4, 8, 5, 6, 2])
```

[19]:
```
[4, 2, 4, 8, 4, 6, 2]
```

Omembe vredno je tole: `s[0] - (s[0] % 2)`. Pisali bi lahko tudi kar `s[0] - s[0] % 2` in bi bilo isto, saj ima deljenje (in prav tako ostanek po deljenju) prednost pred odštevanjem. Vendar je oblika z oklepaji preglednejša, oblika brez pa … čudna.

1.3 Dodatni del

- Napiši funkcijo `zadnje_liho(s)`, ki prejme seznam števil in vrne zadnje liho število v njem. Če v seznamu ni lihih števil, pa naj vrne `False`.

  Klic `zadnje_liho([5, 2, 8, 7, 1, 6])` vrne `1`.

Rešitev Če je seznam prazen, vrnemo `False`. Nato poiščemo zadnje liho število v preostanku seznama. Če obstaja, ga vrnemo, saj je to pač zadnje liho število, ne glede na to, ali je prvi element lih ali ne. Če ne obstaja (in funkcija za ostanek seznama vrne `False`, pa preverimo, ali je prvo število morda liho. Če, potem je to zadnje liho število `s`, saj v preostanku ni več lihih števil. Če pa tudi prvo število ni liho, v celem `s` ni lihih števil, torej vrnemo `False`. Z drugimi besedami, v drugem jeziku:

[20]:
```
def zadnje_liho(s):
    if not s:
        return False
    zadnje = zadnje_liho(s[1:])
    if zadnje != False:
        return zadnje
    if s[0] % 2 == 1:
        return s[0]
    return False
```

[21]:
```
zadnje_liho([5, 2, 8, 7, 1, 6])
```

[21]:
```
1
```

Funkcija ima neko nenavadnost: drznil sem si napisati `if zadnje != False`, čeprav vas že skoraj od 1. oktobra prepričujem, da je to grozno in da se piše `if not zadnje`. Vendar je tule stvar nekoliko specifična: tu je zadnje lahko bodisi število (`int`) bodisi logična vrednost (`bool`). Tu me *v resnici* zanima, ali ni `False`.

Nadalje se tu skriva past:

[22]:
```
False == 0
```

[22]:
```
True
```

[23]:
```
False != 0
```

[23]:
```
False
```

Če bi bilo zadnje liho število v preostanku seznama slučajno enako 0, bi program zmotno mislil, da je funkcija vrnila `False`, saj je `0` in `False` eno in isto. Vendar `0` iz nekih razlogov, ki imajo korenine globoko v matematiki, ne more biti zadnje liho število v seznamu. (Tu se ne bomo preveč poglabljali v matematiko, tako da bom samo namignil: osnovni trik je v tem, da `0` ni liho število. :)-.

Vseeno pa je lepo tule pisati takole:

[24]:
```
def zadnje_liho(s):
    if not s:
        return False
    zadnje = zadnje_liho(s[1:])
    if zadnje is False:
        return zadnje
    if s[0] % 2 == 1:
        return s[0]
    return False
```

Zato ker

[25]:
```
False == 0
```

[25]:
```
True
```

[26]:
```
False is 0
```

[26]:
```
False
```

Neželena alternativna rešitev Skoraj vsi študenti so izziv te naloge videli drugje. V vseh nalogah, kjer smo šli z rekurzijo prek seznama, smo šli od začetka proti koncu. Prvo liho število smo poiskali z

[27]:
```
def prvo_liho(s):
    if not s:
        return False
    if s[0] % 2 == 1:
        return s[0]
    else:
        return prvo_liho(s[1:])
```

[28]:
```
prvo_liho([2, 4, 5, 3, 4, 8, 7, 2])
```

[28]:
```
5
```

Za zadnje liho so vse skupaj le obrnili in gledali s konca seznama.

[29]:
```
def zadnje_liho(s):
    if not s:
        return False
    if s[-1] % 2 == 1:
        return s[-1]
    else:
        return zadnje_liho(s[:-1])
```

[30]:
```
zadnje_liho([2, 4, 5, 3, 4, 8, 7, 2])
```

[30]:
```
7
```

OK. Čeprav je bila ideja v tem, da znamo najti zadnje liho ob iskanju naprej, ne nazaj.

Ob sestavljanju naloge na to nisem niti pomislil. Ves hec rekurzije na seznamih namreč prihaja iz Lispa. To je jezik, ki nima spremenljivk in zato tudi ne zank, prav tako nima seznamov, temveč pare. Seznam `[2, 4, 5, 3, 4, 8, 7, 2]` bi se v Lispu zapisal kot

[31]:
```
s = (2, (4, (5, (3, (4, (8, (7, (2, None)))))))))
```

Za iskanje prvega lihega v takšnem seznamu bi napisali funkcijo

[32]:
```
def prvo_liho_lisp(s):
    prvi, ostali = s
    if prvi % 2 == 1:
        return prvi
    else:
        return prvo_liho_lisp(ostali)
```

[33]:
```
prvo_liho_lisp(s)
```

[33]: 5

(Samo da bi namesto `prvi` in `ostali` pisali `car` in `cdr`. Dolga zgodba.)

Za iskanje zadnjega lihega trik z `s[-1]` zdaj ne vžge več, ker nimamo več nobenega pametnega načina za dostopanje do zadnjega elementa (izvzemši zanko), predvsem pa nimamo res nobenega pametnega načina, da odrežemo zadnji element (`s[:-1]`). Zato bi iskanje zadnjega nujno sprogramirali s funkcijo, podobno tisti v prvi rešitvi.

Ob tem je potrebno omeniti, da je Lisp eden prvih računalniških jezikov in ga pogosto povezujejo tudi z umetno inteligenco. Ker je bil namenjen preprostim računalnikom, ima zelo malo elementov, torej je v nekem smislu zelo preprost. In prav zato zelo zapleten: sestavni deli, ki jih imamo na voljo, so zelo "primitivni", predvsem pa programiranje v njem zahteva zelo drugačen način razmišljanja kot v večini programskih jezikov, ki jih uporabljamo. To povedši pa moramo vseeno dodati, da je Lisp eden tistih starih jezikov, ki so najmanj mrtvi: ne le, da je inspiriral marsikaj v sodobnih jezikih, temveč je zaradi drugačnega načina razmišljanja uporaben za predstavljanje določenih konceptov in tudi za splošno rekreacijo programerjev.

Napačna rešitev Tole ne deluje.

[49]:
```
def zadnje_liho(s):
    if not s:
        return False
    t = obrni(s)
    if t[0] % 2 == 1:
        return t[0]
    else:
        return zadnje_liho(t[1:])
```

[50]:
```
zadnje_liho([2, 4, 5, 6, 8, 9, 10, 12, 14])
```

[50]: 5

Problem je v tem, da se seznam obrne ob vsakem klicu. O tem se hitro prepričamo z dodatnim `print`-om.

[52]:
```
def zadnje_liho(s):
    print(s)
    if not s:
        return False
```

```
    t = obrni(s)
    if t[0] % 2 == 1:
        return t[0]
    else:
        return zadnje_liho(t[1:])
```

[53]:
```
zadnje_liho([2, 4, 5, 6, 8, 9, 10, 12, 14])
```

```
[2, 4, 5, 6, 8, 9, 10, 12, 14]
[12, 10, 9, 8, 6, 5, 4, 2]
[4, 5, 6, 8, 9, 10, 12]
[10, 9, 8, 6, 5, 4]
[5, 6, 8, 9, 10]
[9, 8, 6, 5]
```

[53]:
```
5
```

Funkcija bo vrnila prvo ali zadnje liho število, odvisno od tega, kateri je bližji robu.

Ta napaka je zanimiva, ker kaže na to, da ne razumemo ali pa nismo dobro razmislili, kaj se dogaja v rekurziji.