03b še nekaj primerov

January 28, 2024

0.1 Zaporedje števil

Nekje smo našli tale skrivnostni program. Kaj računa?

[1]:
```
s = 1
for i in range(10):
    s = (s * 165 + 1) % 256
    print(s)
```

```
166
255
92
77
162
107
248
217
222
23
```

Kr neki. Številke, brez posebnega reda, opaznega pravila. Morda bi celo koga preslepili, da so naključne. (Čeprav niso, ta števila očiotno niso naključna, saj so izračunana po neki formuli.)

V resnici tako delujejo generatorji naključnih števil; temu, konkretno, se reče linearni kongruenčni generator. Linearni, ker računa linearno funkcijo, kongruenčni zaradi modula.

Če bi hoteli z njim, recimo, simulirati met kocke, bi ga uporabili tako, da bi namesto vrednosti `s` izpisovali ostanek po deljenju `s` z 6.

[2]:
```
s = 1
for i in range(10):
    s = (s * 165 + 1) % 256
    print(s % 6)
```

```
4
3
2
5
0
5
2
```

```
1
0
5
```

Hmnja, prav, ta kocka ima številke od 0 do 5 namesto od 1 do 6. Pa recimo, da je to kocka, kakršno uporabljajo programerji v Pythonu (in večini drugih jezikov); ti namreč štejejo od 0, ne od 1. Program, ki namesto tega izpisuje za 1 večja števila bi bilo namreč grozljivo težko napisati. ;)

Zdaj me zanima, ali je kocka poštena. So vse številke enako pogoste?

To bom preveril tako, da bom pripravil seznam s šestimi elementi - ničti bo štel, kolikokrat je padla 0, prvi, kolikokrat ena, drugi kolikokrat dva in tako naprej. Saj poznamo vajo, od prejšnjič.

[3]:
```
s = 1
meti = [0] * 6
for i in range(1000):
    s = (s * 165 + 1) % 256
    meti[s % 6] += 1

print(meti)
```

```
[167, 168, 167, 167, 166, 165]
```

Super, kocka je poštena, vse številke so praktično enako verjetne.

(Mimogrede, to je kar zelo neres. Če dobro razmislimo, je ta kocka takšna, da se izmenjujejo lihe in sode številke. Takšne kocke še nikoli nisem videl. V kriptografiji se pogosto uporabljajo naključna števila - vsakič, ko greste na kakšno stran, katere URL je https, ne http, si mora vaš računalnik izmisliti naključno število, ki bo služilo kot ključ … in tako naprej. Kriptografi zato cenijo takšne generatorje naključnih števil, pri katerih na podlagi enega števila (ali nekaj zadnjih) ne izvemo čisto nič o naslednjih. Tale očitno ni takšen.)

Za preštevanje sem uporabil seznam, ne slovarja. Tule štejem pojavitve (majhnih) števil. Števila lahko uporabim kot indekse v seznam, zato sem vzel seznam. Z nizi pa lahko indeksiram samo slovarje. Pa bi lahko tudi tu uporabil slovar? Da, lahko, vendar je tule bolj praktičen seznam, saj sem smel v začetku napisati `meti = [0] * 6`, pa sem dobil pripravljen seznam s šestimi ničlami, v katerega sem potem samo prišteval.

Seveda bi šlo tudi s slovarji. Popaziti bi bilo potrebno na začetno nastavljanje elementov ali pa uporabiti `defaultdict`. Lahko pa se znajdemo z neko čudno funkcijo, `dict.fromkeys`. (Zakaj čudno? Ker ima tako čudno ime. Kot da bi bil `dict` modul.) Funkcija `dict.fromkeys` prejme kot argument ključe (recimo seznam) in privzeto vrednost; če je ne podamo, je privzeta vrednost `None`.

[4]:
```
dict.fromkeys(["Ana", "Berta", "Cilka"])
```

[4]:
```
{'Ana': None, 'Berta': None, 'Cilka': None}
```

[5]:
```
dict.fromkeys("Berta")
```

[5]:
```
{'B': None, 'e': None, 'r': None, 't': None, 'a': None}
```

[6]:
```
dict.fromkeys("Berta", 42)
```

[6]:
```
{'B': 42, 'e': 42, 'r': 42, 't': 42, 'a': 42}
```

[7]:
```
dict.fromkeys(range(6), 0)
```

[7]:
```
{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0}
```

Pa imamo: isti program, le s slovarjem:

[8]:
```
s = 1
meti = dict.fromkeys(range(6), 0)
for i in range(1000):
    s = (s * 165 + 1) % 256
    meti[s % 6] += 1

print(meti)
```

```
{0: 167, 1: 168, 2: 167, 3: 167, 4: 166, 5: 165}
```

Zdaj pa si raje zastavimo bolj zanimivo vprašanje: se začnejo te številke kdaj ponavljati?

Kot običajno, pridemo do odgovora s tehniko, ki jo imenujemo *razmišljanje*. :) Po krajšem razmišljanju torej ugotovimo, da se, prav gotovo: ker ima `s` samo 256 različnih možnih vrednosti (pač številke med 0 in 255, saj računamo ostanke po deljenju z 256), bomo v 257. koraku (ali prej, če imamo smolo, naleteli na številko, ki smo jo že videli. In ko prvič dobimo nek `s`, ki smo ga že imeli, bo očitno enak tudi naslednji `s` in za njim naslednji in tako naprej.

Pa recimo, da nismo tako pametni, da bi ugotovili, da ima `s` samo 256 vrednost: recimo, da natuhtamo samo, da se bodo `s`-ji, čim se ponovijo enkrat, ponavljali v nedogled. Ne vemo pa, ali se bodo kdaj ponovili. Radi bi torej spremenili program tako, da se bo ustavil, če se `s` v 1000 korakih kdaj ponovi.

[9]:
```
s = 1
meti = []
for i in range(1000):
    s = (s * 165 + 1) % 256
    if s in meti:
        print("Številka", s, "se je ponovila")
        break
    meti.append(s)
else:
    print("V 1000 korakih se s ni ponovila")
```

```
Številka 166 se je ponovila
```

Program najbrž ne potrebuje posebne razlage razen, spet, onega hecnega `else` za `for`-om, ki se izvede, če znotraj `for` ne pride do `break`a (vendar seveda pride).

Vedno, kadar vidim, da nekdo preverja, ali seznam vsebuje določen element, se mi Python, ki mora preletati ves seznam, zasmili. Zato tule raje uporabimo slovarje. Če bo `meti` slovar, bo znal `s in`

`meti` brez iskanja povedati, ali `meti` vsebuje ključ `s` ali ne. Kaj torej uporabimo tu? Množice.

[10]:
```
s = 1
meti = set()
for i in range(1000):
    s = (s * 165 + 1) % 256
    if s in meti:
        print("Številka", s, "se je ponovila")
        break
    meti.add(s)
else:
    print("V 1000 korakih se s ni ponovila")
```

```
Številka 166 se je ponovila
```

Vse je tako kot prej, le namesto `meti = []` imamo `meti = set()` in namesto `meti.append(s)` imamo `meti.add(s)`.

0.2 Primer: Končnice datotek

Po množicah kar kličejo naloge tipa "Poišči končnice vseh datotek, ki se nahajajo v danem direktoriju. Množice namreč same skrbijo za to, da se vsaka stvar pojavi le enkrat.

[11]:
```
import os

koncnice = set()
for ime in os.listdir("."):
    konc = ime.split(".")[-1]
    koncnice.add(konc)

for konc in sorted(koncnice):
    print(konc)
```

```
ipynb
ipynb_checkpoints
```

0.3 Še en primer

Preseki in unije so uporabne reči. Ena vaših kolegic se je pred leti na vajah lotila računanja največjega skupnega delitelja dveh števil tako, da je najprej izračunala vse delitelje enega, nato vse delitelje drugega, potem pa si je želela med njimi poiskati največjega.

```
def vsi_delitelji(a):
    delitelji = []
    for i in range(1, a):
        if a % i == 0:
            delitelji.append(i)
    return delitelji

def gcd(a, b):
```

```
    delitelji_a = vsi_delitelji(a)
    delitelji_b = vsi_delitelji(b)
    in kaj zdaj?!?!
```

Zdaj imamo dva seznama. Kdo bo našel največje število, ki se pojavi v obeh? Za vsako število iz enega seznama bi bilo potrebno preveriti, ali obstaja tudi v drugem in ali je večje od največjega doslej.

Čisto za vajo naredimo to reč učinkovito.

[12]:
```
def vsi_delitelji(a):
    delitelji = []
    for i in range(1, a):
        if a % i == 0:
            delitelji.append(i)
    return delitelji

def gcd(a, b):
    delitelji_a = vsi_delitelji(a)
    delitelji_b = vsi_delitelji(b)
    ia = ib = -1
    while True:
        if delitelji_a[ia] == delitelji_b[ib]:
            return delitelji_a[ia]
        if delitelji_a[ia] > delitelji_b[ib]:
            ia -= 1
        else:
            ib -= 1
```

[13]:
```
gcd(2 * 7 * 13 * 5, 2 * 7 * 3 * 11 * 17)
```

[13]:
```
14
```

To je lepo, a ni?

Aja, moram prej razložiti, kako deluje. Seznama vsebujeta delitelje `a` in `b`, urejene po velikosti. Z enim prstom pokažemo na zadnji element prvega seznama, z drugim na zadnji element drugega. Preverimo, kateri prst kaže na večjo število in ga premaknemo za en element levo, na manjše število. To ponavljamo, dokler ne kažeta oba prsta na enako število - v skrajnem primeru bo to 1. To število je potem največji skupni delitelj.

To je bila čista stranpot, pokazal sem samo zato, ker je tako lepo. Oni študentki tega nisem kazal, spomnil sem jo na stari, dobri in učinkovitejši Evklidov algoritem. Če bi že vedeli za množice, pa bi jo lahko vprašal, če ne bi bilo morda boljše uporabiti množic… Kolegica bi rekla, ah, seveda in jadrno napisala

[14]:
```
def vsi_delitelji(a):
    delitelji = []
    for i in range(1, a):
        if a % i == 0:
```

```
            delitelji.append(i)
    return delitelji

def gcd(a, b):
    delitelji_a = set(vsi_delitelji(a))
    delitelji_b = set(vsi_delitelji(b))
    return max(delitelji_a & delitelji_b)
```

[15]:
```
gcd(2 * 7 * 13 * 5, 2 * 7 * 3 * 11 * 17)
```

[15]: 14

Ali, z enim zamahom.

[16]:
```
def vsi_delitelji(a):
    delitelji = []
    for i in range(1, a):
        if a % i == 0:
            delitelji.append(i)
    return delitelji

def gcd(a, b):
    delitelji_a = set(vsi_delitelji(a))
    delitelji_b = set(vsi_delitelji(b))
    return max(delitelji_a & delitelji_b)
```

[17]:
```
gcd(2 * 7 * 13 * 5, 2 * 7 * 3 * 11 * 17)
```

[17]: 14

Drugo funkcijo lahko očitno brez posebnega znanja stlačimo v eno vrstico. Zanimivo je, da lahko tudi prvo. Kot reklamo za to, kar nas čaka v prihodnjih tednih:

[18]:
```
def vsi_delitelji(a):
    return {i for i in range(1, a) if a % i == 0}

def gcd(a, b):
    return max(vsi_delitelji(a) & vsi_delitelji(b))
```

[19]:
```
gcd(2 * 7 * 13 * 5, 2 * 7 * 3 * 11 * 17)
```

[19]: 14

0.4 Malo bolj zapleteni slovarji

Vrnimo se k slovarjem, a ne pozabivši množic. V neki nalogi z vaj smo pomagali natakarju, ki je prejel naročila v takšnem seznamu:

[20]:
```
s = [("Ana", "kava"), ("Berta", "pita"), ("Ana", "pita"),
     ("Cilka", "caj"), ("Ana", "voda"), ("Berta", "voda")]
```

Napisati želimo funkcijo (ali program, tule nam ni mar), ki to predela v slovar, katerega ključi so imena strank, vrednosti pa povedo, kaj je stranka naročila. Na vajah smo rekli, naj bodo vrednosti *seznami* naročenih reči. Zdaj, ko vemo za množice, ni razloga, da ne bi uporabili množic. Dobiti hočemo torej

[21]:
```
narocila = {"Ana": {"kava", "pita", "voda"}, "Berta": {"pita", "voda"}, "Cilka":
 ↪ {"caj"}}
```

Stvar sploh ni tako težka, kot so mislili študenti FRI, ko jih je ta naloga zadela na izpitu. Drznil bi si trditi celo, da niso vedeli, da je čisto lahka in, v bistvu, nič drugačna od naloge, v kateri štejemo, kolikokrat je Benjamin poklical katero od An, Bert in Cilk. (In, naj izdam skrivnost: nič težja ali lažja od dveh izmed funkcij, ki ju morate narediti v naslednjem tednu! :)

Pripravimo si prazen slovar. Narediti nam je tole: gremo čez vse pare `(ime, narocilo)` v `s`. Za vsako ime pogledamo, ali je že v slovarju in če ga ni, ga dodamo, kot vrednost, pa mu damo - ne 0, kot pri klicih, temveč prazno množico, `set()`. Potem dodamo v seznam, kar je dotična gospa naročila.

[22]:
```
narocila = {}
for stranka, narocilo in s:
    if not stranka in narocila:
        narocila[stranka] = set()
    narocila[stranka].add(narocilo)

narocila
```

[22]:
```
{'Ana': {'kava', 'pita', 'voda'}, 'Berta': {'pita', 'voda'}, 'Cilka': {'caj'}}
```

Še lažje je, če se spomnimo na `defaultdict`.

[23]:
```
from collections import defaultdict

narocila = defaultdict(set)

for stranka, narocilo in s:
    narocila[stranka].add(narocilo)

narocila
```

[23]:
```
defaultdict(set,
            {'Ana': {'kava', 'pita', 'voda'},
             'Berta': {'pita', 'voda'},
             'Cilka': {'caj'}})
```