resitev

January 28, 2024

0.1 Ogrevanje: Zaporedje števil

Ob takem zaporedju bi zavil z očmi celo Collatz: naslednji člen zaporedja izračunamo tako, da trenutni člen pomnožimo s 13, prištejemo 1 in nato izračunamo ostanek po deljenju s 16, se pravi $x_i = (13x_{i-1} + 1) \,\%\, 16$, pri čemer `%` pomeni ostanek po deljenju. Začetni člen zaporedja naj bo 0.

Napiši program, ki izračuna in izpiše prvih 16 členov tega zaporedja. Izpisati mora torej

```
0
1
14
7
12
13
10
3
8
9
6
15
4
5
2
11
```

0.1.1 Rešitev

Običajni vzorec za štetje je

```
i = 0
while i < 16:
    print(i)
    i += 1
```

le da tu ne bomo izpisovali števca, `i`-ja, ker nas ne zanima, potrebujemo ga le za štetje. Pač pa bomo imeli še eno spremenljivko, ki bo vsebovala vrednost člena: `x`. Ta bo v začetku 0; v vsakem koraku ga bomo izpisali in irzačunali naslednjega.

[1]:
```
i = 0
x = 0
while i < 16:
```

```
    print(x)
    x = (13 * x + 1) % 16
    i += 1
```

```
0
1
14
7
12
13
10
3
8
9
6
15
4
5
2
11
```

0.2 Naloga (nadaljevanje)

Zdaj pa spremeni program tako, da bo izpisal 1000 členov zaporedja. Hitro boš videl(a), da se stalno ponavlja istih 16 členov, kar je dolgčas, zato ga spremeni program tako, da uporablja drugačno formulo za izračun naslednjega člena: $x_i = (1664525\, x_{i-1} + 1013904223) \ \% \ 2^{32}$. Zdaj se zaporedje začne z

```
0
1013904223
1196435762
3519870697
2868466484
1649599747
2670642822
1476291629
2748932008
2180890343
```

0.2.1 Rešitev

[2]:
```
i = 0
x = 0
while i < 10:  # V teh zapiskih bomo izpisali samo prvih 10 členov...
    print(x)
    x = (1664525 * x + 1013904223) % 2 ** 32
    i += 1
```

```
0
1013904223
1196435762
3519870697
2868466484
1649599747
2670642822
1476291629
2748932008
2180890343
```

0.3 Glavni del: Zaporedje sob

V nekem hodniku je 10 sob. Ker je lastnik računalnikar, so označene s številkami od 0 do 9 in ne od 1 do 10.

Ana obiskuje sobe v naključnem vrstnem redu, ki ga dobi tako, da računa člene zaporedja iz prejšnje naloge (zadnjega, tistega z velikimi koeficienti). Za vsak člen pogleda skrajno desno števko in gre v sobo s to številko. Pri gornjem zaporedju gre torej v sobe 0, 1, 6, 1, 8, 9, …

Ker ima preveč časa (najbrž je v samoizolaciji?), to ponovi tisočkrat. Program naj ne izpisuje členov ali sob, temveč naj izpiše, kolikokrat je bila Ana v sobi številka 6.

(Če se nisem zmotil, je pravilni odgovor 96.)

0.3.1 Rešitev

Dodamo še novo spremenljivko (rekli ji bomo `soba6`), v kateri bomo šteli, kolikokrat je zadnja števka `x` enaka 6. Kako pa dobimo zadnjo števko `x`? Ostanek po deljenju z 10.

[3]:
```
i = 0
x = 0
soba6 = 0
while i < 1000:
    x = (1664525 * x + 1013904223) % 2 ** 32
    if x % 10 == 6:
        soba6 += 1
    i += 1
print(soba6)
```

```
96
```

Najbrž se nisem motil. :)

0.3.2 Dodatna naloga: Srečanja

Tudi Berta je v samoizolaciji in hodi po istih sobah, le drugačno formulo uporablja: $x_i = (22695477\, x_{i-1} + 1) \ \% \ 2^{32}$. Tudi ona gre vedno v sobo, ki jo določi glede na ostanek po deljenju člena z 10. Tudi ona to stori tisočkrat.

Obe računata enako hitro, se pravi, obe gresta najprej v sobo, ki ustreza prvemu členu, nato obe v istem trenutku v sobo, ki ustreza drugemu členu, tretjemu …

Napiši program, ki izračuna, kolikokrat se bosta srečali.

(Če se nisem zmotil, je pravilni odgovor 195.)

[4]:
```
skupaj = 0
x1 = 0
x2 = 0
for i in range(1000):
    if x1 % 10 == x2 % 10:
        skupaj += 1
    x1 = (1664525 * x1 + 1013904223) % 2 ** 32
    x2 = (22695477 * x2 + 1) % 2 ** 32
print(skupaj)
```

```
195
```

Ker nekatere študente take stvari zanimajo:

[5]:
```
from functools import reduce

print(
    sum(
        map(lambda x: 2 - len(set(map(lambda t: t % 10, x))),
            zip(reduce(lambda s, _: s + [(1664525 * s[-1] + 1013904223) % (2 **␣
 ↪32)], range(1000), [0]),
                reduce(lambda s, _: s + [(22695477 * s[-1] + 1) % (2 ** 32)],␣
 ↪range(1000), [0])))))
```

```
195
```

Ampak ni prav lepo. So tudi lepši onelinerji. Ampak ne za ta problem.

0.4 Ozadje naloge

Na ta način so dolgo delale funkcije, ki vračajo naključna števila. Začetna vrednost ni 0, temveč jo izberemo na nek dovolj "naključen" način, na primer na podlagi trenutnega časa ali pa na podlagi natančnega merjenja razmikov med uporabnikovimi akcijami (npr. pritiski na tipke). Od tam naprej pa vsako naslednje "naključno" število izračunamo iz prejšnjega s pomočjo formule v obliki $(ax + c)$. Vrnjeno naključno število je pri tem del tega, naračunanega števila. Konstante morajo biti a, c in m izbrane tako, da se števila ne začnejo prehitro ponavljati. Prej ko slej pa se morajo: ker računamo ostanek po deljenju z m, teh ostankov pa je le m, se bo člen po največ m korakih ponovil. Ko se ponovi en člen, pa se seveda tudi vsi za njim.

Ker so ta števila izračunana z neko formulo, seveda niso naključna, čeprav se zdijo takšna. Rečemo jim psevdo-naključna števila. Resnično naključnih števil računalnik tako ali tako ne more proizvajati, saj v njem ni ničesar naključnega (brez kakšne dodatne opreme, seveda).

Več o tem si lahko preberete na strani Wikipedije o linearnih kongruenčnih generatorjih.

Moderni generatorji naključnih števil temeljijo na drugačni, malo bolj zapleteni matematiki Mersennovega vrtinca.