6. Generování náhodných čísel a testování generátorů

Zadání:

Tento úkol bude poněkud kreativnější charakteru. Vaším úkolem je vytvořit vlastní generátor semínka do pseudonáhodných algoritmů. Jazyk Python umí sbírat přes ovladače hardwarových zařízení různá fyzická a fyzikální data. Můžete i sbírat data z historie prohlížeče, snímaní pohybu myší, vyzvání uživatele zadat náhodné úhozy do klávesnice a jiná unikátní data uživatelů.

Řešení:

Vytvořil jsem generátor pseudonáhodných čísel, který se zakládá na čtení historie z prohlížeče Google Chrome. Načte předem do proměnné "results", pomocí SQL dotazu, určený počet URL z historie podle toho, kolik uživatel chce vygenerovat náhodných čísel.

```
# Sběr dat a jejich manipulace
2 usages
def collect_data():
    history_db_path = "C:/Users/XXX/AppData/Local/Google/Chrome/User Data/Default/History"

    # Připojení k databázi
    connection = sqlite3.connect(history_db_path)
    pointer = connection.cursor()

    # SQL dotaz pro získání posledních "no_count" navštívených stránek
    query = f"SELECT url FROM urls ORDER BY last_visit_time DESC LIMIT {no_count};"
    pointer.execute(query)
    results = pointer.fetchall() # Načtení výsledků do results

    # Ukončení spojení s databází historioe
    connection.close()
```

Před spuštěním programu je nutné naimportovat knihovny a nastavit hodnoty pro “no_count” (počet generovaných čísel) a “upper_limit“ (nejvyšší možné vygenerované číslo).

```
import sqlite3
import random
import time
import win32api
import cpuinfo

no_count = 1000
upper_limit = 5000

```

Po načtení všech hodnot z historie program pro každý výsledek odstraní prefix (https://) kvůli snížení počtu možných duplicit, jelikož informaci o protokolu nese téměř každá webová stránka. Po odstranění prefixu se zavolá funkce “generate_seed()”, kde mám aktuální webovou stránkou a horní limit pro čísla jako argumenty. Nakonec se uloží vrácený výsledek z funkce “generate_seed()” do seznamu “RNG_numbers”, kam z důvodu kontroly duplicit ukládám všechna čísla, která mi generátor dá.

```
    # Generování náhodných čísel na základě URL adres
    for result in results:
        to_strip = result[0]
        stripped_url = to_strip.lstrip("https://") # Odstraní prefix
        print(result[0])
        print(stripped_url)
        rng_seed = generate_seed(stripped_url, upper_limit)
        RNG_numbers.append(rng_seed)
        print(rng_seed)

```

Po zavolání funkce se vybere náhodný počet znaků z odkazu, opět kvůli vysokému počtu duplicit, které poté převede na jednotlivá čísla a sečte je. Dále vezme aktuální čas systému CPU a čas aktuálního procesu. Vzhledem k hodnotám, které tato funkce vrací je nutné je vynásobit, aby hodnoty mely vliv na chod generátoru. Poté je nutné je převést na celočíselnou hodnotu (funkce vrací float). Dále jsem pro vyšší randomizaci při vyšším počtu vygenerovaných čísel použil čtení pozice kurzoru na obrazovce uživatele.

```
def generate_seed(url, limit):
    selected_chars = url[:random.randint(1, len(url))]  # Vyber náhodný počet prvků z odkazu
    number = sum(ord(char) for char in selected_chars) # Převeď všechny charaktery na čísla a sečti je

    cpu_clock = int(time.process_time() * 1e9)  # CPU clock

    mouse_pos = win32api.GetCursorPos()  # Aktualni pozice myši
    mouse_seed = (mouse_pos[0] + mouse_pos[1])
```

Pokoušel jsem se použít i teplotu procesoru, jako další prvek náhody. Nicméně funkce “cpuinfo.get_cpu_info()” od verze knihovny py-cpuinfo 5.0.0 a vyšší čte hodnoty z CPU pomocí CPUID instrukce, kterou provede jednou, poté má 1 sekundu čekat, odečíst hodnoty znovu a vrátit jejich rozdíl. Z tohoto důvodu jsem musel metodu z generátoru odstranit.

```
    # Získání teploty procesoru pomocí knihovny py-cpuinfo -- Hodně zpomaluje RNG kvůli použité metodě (1s sleep time)
#     cpu_temp = cpuinfo.get_cpu_info().get("current_temp")
#     temperature_seed = int(cpu_temp) if cpu_temp else 0

    rng_seed = (number + mouse_seed + cpu_clock) % limit  # Výpočet čísla
    return rng_seed

```

Výpočet samotného semínka se skládá ze sumy převedených znaků na čísla, přičtené pozice myši a poté času CPU. Vrácenou hodnotu z funkce přidávám do seznamu RNG_numbers.

Po první várce takto vygenerovaných čísel se zavolá funkce “validate_numbers()”, která má za úkol, jak už název napovídá, zda je ono vygenerované číslo originální. Pokud ano, přidá jej do seznamu vygenerovaných

čísel, pokud se již v tomto seznamu nachází, přidá jej do seznamu duplicitních čísel. Zároveň tato funkce, kvůli možnému dalšímu užití po kontrole onoho čísla jej rovnou odstraní ze seznamu RNG_numbers, aby nedošlo k více kontrolám jednoho prvku seznamu.

```
33  # Kontrola duplicit
    3 usages
34  def validate_number():
35      index = 0
36      for number in RNG_numbers:
37          if number not in validatedNumbers:
38              validatedNumbers.append(number)
39              RNG_numbers.pop(index)
40              index += 1
41          else:
42              duplicity.append(number)
43              RNG_numbers.pop(index)
44              index += 1
45      if len(RNG_numbers) != 0:
46          validate_number()
47      validate_count()
48
```

Po rozházení čísel do seznamu se podívám, zda opravdu v RNG_numbers nezůstalo nějaké nepřiřazené číslo. Pokud ne, zavolá se funkce “validate_count()”, která má za úkol zkontrolovat, zda byl vygenerován požadovaný počet čísel porovnáním počtu originálních čísel s požadovaným počtem. Nakonec spočítám rozdíl, kolik čísel chybí, uložím jej do globální proměnné “no_count” a zavolám znovu funkce na generování čísel.

```
49
50  # Kontrola počtu vygenerovaných čísel
    1 usage
51  def validate_count():
52      global no_count
53      count = no_count
54      if len(validatedNumbers) < no_count:
55          no_count = count - len(validatedNumbers)
56          collect_data()
57          validate_number()
58
```

Závěr:

Testoval jsem generátor s různým nastavením počtu čísel, abych zjistil, jak je odolný vůči duplicitám. Upper_limit jsem pro testování nastavil na 10000. Při 100 číslech mi generátor vrátil 100 čísel a 2 duplicitní hodnoty, což odpovídá 2% duplicit. Při 500 číslech už ale generátor vygeneroval pouze 498 čísel a 44 duplicit, což je téměř 9,5%. Při testu na 2000 čísel již bylo pouze 1935 čísel a 405 duplicit – !20%!.

S rostoucím počtem čísel se bohužel počet vygenerovaných čísel vzdaluje od požadavku - možná chyba kódu. Protože při požadavku na 7000 čísel jich program vygeneroval pouze 5777 a k tomu 3443 duplicit – 50%. Z toho usuzuji, že zakládat generátor náhodných čísel na čtení dat z historie a práce s odkazy není nejlepší směr, kterým se pro kvalitní generátor pseudonáhodných čísel vydat, myslím si, že příčina je poměrně vysoká podobnost URL odkazů, tudíž je vysoká šance pro získání duplicit.