

PRAKTICKÉ ASPEKTY VÝVOJE SOFTWARE

2020/2021

Protokol o výsledcích profilování aplikace
pro výpočet směrodatné odchylky pomocí vlastní
matematické knihovny

Obsah

1	Úvod	2
2	Testovací data a profiler	2
3	Výsledky	2
4	Vyhodnocení	4
5	Ruční optimalizace	4
6	Závěr	5

1 Úvod

Cílem tohoto dokumentu je analyzovat a evaluovat výsledky profilingu výpočtu výběrové směrodatné odchylky pomocí vlastní matematické knihovny `math_lib.py` dle následujícího vzorce:

$$s = \sqrt{\frac{1}{N-1} \left(\sum_{i=1}^N x_i^2 - N\bar{x}^2 \right)}$$
$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

2 Testovací data a profiler

Dle zadání projektu byla aplikace profilována pro tři testovací sady čísel o 10, 100 a 1000 prvcích. Všechna čísla byla náhodně generována pomocí online nástroje¹ v rozsahu od -999999 do 999999 a následně vložena do textového souboru, který byl aplikaci předán na standardní vstup.

Pro samotný profiling byl použit nativní profiler vývojového prostředí *PyCharm Professional*², který vytvořil `.pstat` soubor s výsledky a zároveň umožnil udělat call graph.

3 Výsledky

Name	Call Count ▼	Time (ms)	Own Time (ms)
sum	150	0 0.0%	0 0.0%
<method 'isdigit' of 'str' objects>	67	0 0.0%	0 0.0%
<built-in method builtins.isinstance>	55	0 0.0%	0 0.0%
power	51	0 0.0%	0 0.0%
<method 'rstrip' of 'str' objects>	14	0 0.0%	0 0.0%
<method 'join' of 'str' objects>	8	0 0.0%	0 0.0%
<method 'rpartition' of 'str' objects>	7	0 0.0%	0 0.0%
_verbose_message	7	0 0.0%	0 0.0%
<listcomp>	6	0 0.0%	0 0.0%
_path_join	6	0 0.0%	0 0.0%
<built-in method builtins.round>	5	0 0.0%	0 0.0%
<built-in method _imp.acquire_lock>	5	0 0.0%	0 0.0%
<built-in method _imp.release_lock>	5	0 0.0%	0 0.0%
<built-in method builtins.hasattr>	3	0 0.0%	0 0.0%
<built-in method posix.stat>	3	0 0.0%	0 0.0%
<built-in method posix.fspath>	3	0 0.0%	0 0.0%
<built-in method _codecs.utf_8_decode>	3	0 0.0%	0 0.0%
decode	3	0 0.0%	0 0.0%

Obrázek 1: Část výsledků profileru pro 10 náhodně vygenerovaných čísel řazených dle počtu vykonání

¹Viz <https://www.random.org/integers/>

²Viz <https://www.jetbrains.com/help/pycharm/profiler.html>

Name	Call Count ▾	Time (ms)	Own Time (ms)
sum	1476	0 0.0%	0 0.0%
<method 'isdigit' of 'str' objects>	648	0 0.0%	0 0.0%
<built-in method builtins.isinstance>	497	0 0.0%	0 0.0%
power	493	0 0.0%	0 0.0%
<method 'rstrip' of 'str' objects>	14	0 0.0%	0 0.0%
<method 'join' of 'str' objects>	8	0 0.0%	0 0.0%
<method 'rpartition' of 'str' objects>	7	0 0.0%	0 0.0%
_verbose_message	7	0 0.0%	0 0.0%
<listcomp>	6	0 0.0%	0 0.0%
_path_join	6	0 0.0%	0 0.0%
<built-in method builtins.round>	5	0 0.0%	0 0.0%
<built-in method _imp.acquire_lock>	5	0 0.0%	0 0.0%
<built-in method _imp.release_lock>	5	0 0.0%	0 0.0%
<built-in method builtins.hasattr>	3	0 0.0%	0 0.0%
<built-in method posix.stat>	3	0 0.0%	0 0.0%
<built-in method posix.fspath>	3	0 0.0%	0 0.0%
<built-in method _codecs.utf_8_decode>	3	0 0.0%	0 0.0%
decode	3	0 0.0%	0 0.0%

Obrázek 2: Část výsledků profileru pro 100 náhodně vygenerovaných čísel řazených dle počtu vykonání

Name	Call Count	Time (ms)	Own Time (... ▾
profiling.py	1	16 100.0%	9 56.2%
sum	14676	2 12.5%	2 12.5%
power	4893	3 18.8%	2 12.5%
<method 'join' of 'str' objects>	8	0 0.0%	0 0.0%
<method 'rstrip' of 'str' objects>	14	0 0.0%	0 0.0%
<method 'rpartition' of 'str' objects>	7	0 0.0%	0 0.0%
<method 'isdigit' of 'str' objects>	6403	0 0.0%	0 0.0%
<method 'get' of 'dict' objects>	2	0 0.0%	0 0.0%
<built-in method builtins.hasattr>	3	0 0.0%	0 0.0%
<built-in method builtins.isinstance>	4897	0 0.0%	0 0.0%
<built-in method builtins.print>	1	0 0.0%	0 0.0%
<built-in method builtins.round>	5	0 0.0%	0 0.0%
<built-in method posix.stat>	3	0 0.0%	0 0.0%
<built-in method posix.fspath>	3	0 0.0%	0 0.0%
<method 'readlines' of '_io._IOBase' objects>	1	0 0.0%	0 0.0%
<built-in method _thread.allocate_lock>	2	0 0.0%	0 0.0%
<built-in method _thread.get_ident>	2	0 0.0%	0 0.0%
<built-in method _codecs.utf_8_decode>	3	0 0.0%	0 0.0%

Obrázek 3: Část výsledků profileru pro 1000 náhodně vygenerovaných čísel řazených dle času vykonání

4 Vyhodnocení

V případě testovacího vzorku s 10 čísly (obrázek 1) byl procesorový čas u všech použitých funkcí zanedbatelný a profiling v tomhle případě postrádal smysl.

Vstup obsahující 100 čísel (obrázek 2) vykazoval obdobné znaky jako předchozí vzorek a profiling při tak malém počtu vstupních dat neměl smysl.

Poslední testovací vzorek s 1000 čísly (obrázek 3) zaznamenal nejdelší procesorový čas u funkce `sum` (sčítání) a funkce `power` (n-tá mocnina). Funkce sčítání je výpočetně velmi nenáročná a její dlouhý procesorový čas byl primárně podmíněn počtem volání.

Zbylé funkce `sub` (odčítání), `multiply` (násobení), `divide` (dělení) a `nth_root` (n-tá odmocnina) byly volané pouze jednou nebo dvakrát a jejich časová náročnost je zanedbatelná. Z tohoto důvodu není třeba tyto funkce optimalizovat.

Při další optimalizaci programu pro výpočet směrodatné odchylky by se mělo především zaměřit na zrychlení funkce `power`, případně na zrychlení funkce `sum`.

5 Ruční optimalizace

Na základě výsledků prvního profilingu bylo zjištěno, že nejvyšší procesorový čas je stráven u funkce `power`. Za tímto účelem byla funkce optimalizována (ručně opraven její kód) a otestována znovu. Aby výsledky optimalizace byly patrnější, počet vstupních hodnot byl navýšen na 100000.

Name	Call Count	Time (ms)	Own Tim...	▼
profiling.py	1	1447 100.0%	911	63.0%
power	488641	276 19.1%	240	16.6%
sum	1465920	188 13.0%	188	13.0%
<method 'isdigit' of 'str' objects>	639269	67 4.6%	67	4.6%
<built-in method builtins.isinstance>	488645	35 2.4%	35	2.4%
<method 'readlines' of '_io._IOBase' objects>	1	3 0.2%	3	0.2%
<method 'join' of 'str' objects>	8	0 0.0%	0	0.0%
<method 'rstrip' of 'str' objects>	14	0 0.0%	0	0.0%
<method 'rpartition' of 'str' objects>	7	0 0.0%	0	0.0%
<method 'get' of 'dict' objects>	2	0 0.0%	0	0.0%
<built-in method builtins.hasattr>	3	0 0.0%	0	0.0%
<built-in method builtins.print>	1	0 0.0%	0	0.0%
<built-in method builtins.round>	5	0 0.0%	0	0.0%
<built-in method posix.stat>	3	0 0.0%	0	0.0%
<built-in method posix.fspath>	3	0 0.0%	0	0.0%
<built-in method _thread.allocate_lock>	2	0 0.0%	0	0.0%
<built-in method _thread.get_ident>	2	0 0.0%	0	0.0%
<built-in method _codecs.utf_8_decode>	82	0 0.0%	0	0.0%

Obrázek 4: Část výsledků profileru pro 100000 náhodně vygenerovaných čísel řazených dle času vykonání před ruční optimalizací funkce `power`

Name	Call Count	Time (ms)	Own Time ... ▼
profiling.py	1	1354 100.0%	843 62.3%
power	488641	265 19.6%	229 16.9%
sum	1465920	188 13.9%	188 13.9%
<method 'isdigit' of 'str' objects>	639269	52 3.8%	52 3.8%
<built-in method builtins.isinstance>	488645	35 2.6%	35 2.6%
<method 'readlines' of '_io._IOBase' objects>	1	3 0.2%	3 0.2%
<method 'join' of 'str' objects>	8	0 0.0%	0 0.0%
<method 'rstrip' of 'str' objects>	14	0 0.0%	0 0.0%
<method 'rpartition' of 'str' objects>	7	0 0.0%	0 0.0%
<method 'get' of 'dict' objects>	2	0 0.0%	0 0.0%
<built-in method builtins.hasattr>	3	0 0.0%	0 0.0%
<built-in method builtins.print>	1	0 0.0%	0 0.0%
<built-in method builtins.round>	5	0 0.0%	0 0.0%
<built-in method posix.stat>	3	0 0.0%	0 0.0%
<built-in method posix.fspath>	3	0 0.0%	0 0.0%
<built-in method _thread.allocate_lock>	2	0 0.0%	0 0.0%
<built-in method _thread.get_ident>	2	0 0.0%	0 0.0%
<built-in method _codecs.utf_8_decode>	82	0 0.0%	0 0.0%

Obrázek 5: Část výsledků profileru pro 100000 náhodně vygenerovaných čísel řazených dle času vykonání po ruční optimalizaci funkce power

6 Závěr

Z profilingu optimalizované funkce (obrázek 5) na vyšším počtu testovacích vzorků vyplynulo, že je efektivnější používat pro výpočet mocniny nativní operátor Pythonu pro exponent $a**b$, než postupné násobení jednolivých základů (obrázek 4), jakožto bylo použito v původní funkci. Přesto tato optimalizace není nijak markantní a není patrná při počítání s menším počtem vstupních čísel.