IVS - Profiling

DreamTeam

2. dubna 2025

1 Naměřené hodnoty, metodika měření

Cílem této analýzy bylo identifikovat časově nejnáročnější části programu a navrhnout možnosti jeho optimalizace. K profilování byl použit nástroj cProfile, který je vestavěný přímo v Pythonu (použitá verze Pythonu 3.11.9). Tento nástoj analyzuje daný program, aniž bychom museli dělat jakékoli úpravy v samotném kódu. V tabulce se nachází šest funkcí, jejichž provedení zabralo nejvíce seřazeny podle tottime (total time (s)). Ostatní funkce byly vyřazeny z této tabulky, jelikož jejich úprava by nevedla k žádnému významnému zrychlení programu.

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
2	0.003	0.002	0.005	0.003	profiling.py:29(sum)
2001	0.003	0.000	0.003	0.000	math_lib.py:66(expon)
4000	0.002	0.000	0.002	0.000	math_lib.py:12(addition)
1	0.002	0.002	0.005	0.005	profiling.py:58(<listcomp >)
1	0.001	0.001	0.001	0.001	{method 'split' of 're.Pattern' objects}
1	0.001	0.001	0.001	0.001	profiling.py:19(<listcomp >)

Tabulka 1: Výsledky Profilování

Seřazení podle tottime je podle mého názoru nejlepší, jelikož bere pouze čas strávený na provádění jednotlivé funkce, narozdíl třeba od cumtime (cumulative time), jenž je celkový čas strávený ve funkci, tudíž i čas strávených ve funkcích, jež tato funkce volá. Kdybychom seřadili naměřené hodnoty podle parametru cumtime, tak bycho zjistili že nejvíce času (v našem né-moc dlouhém programu) zaberou ty nejkomplexnější funkce. Tudíž se v našem případě řazení tímto způsobem nehodí.

2 Analýza naměřených hodnot, možná zrychlení programu

Nejvíce času na provedení zabrala funkce sum() (vizte Tabulka 1). Z toho vyplývá, že při optimalizaci programu bychom se měli zaměřit právě na tuto funkci. Jedno z možných zrychlení je použít vestavěnou funkci sum(0) v pythonu, ta bude určitě rychlejší než námi definovaná funkce sum() využívající naši vlastní matematickou knihovnu. Dále nejvíce času zabírá provedení funkcí expon() a addition(), na tyto funcke, přesněji jejich optimalizaci bychom se měli obvzláště zaměřit, jelikož jsou v našem programu volány nejčastěji. Zde bychom taky mohli dosáhnout menšího zrychlení, kdybychom použili vestavěné operátory "**"a "+". Toto platí pro všechny funkce naší matematické knihovny. Čtvrtý řádek tabulky poukazuje na tvorbu listu čísel na druhou. Zde bychom zase mohli použí vestavěný operátor. Dále bychom mohli použít pole z knihovny NumPy, se kterou je práce rychlejší než se seznamem. Pátý řádek poukazuje na metodu split() používající regulární výraz pro definování vzoru rozdělení řetězce. V tomto případě si myslím, že zde už žádného zrychlení asi nebudeme schopni dosáhnout. Poslední řádek tabulky nám ukazuje na konverzi načtených řetězců na čísla datového typu float. Zde bychom mohli dosáhnout zrychlení, kdybychom přetypování nahradili za funkci map().

3 Zhodnocení

V našem programu je značný prostor pro zlepšení. Určitě se zde také nachází nějaká místa, která se profilovacímu nástroji nepovedlo odhalit.