Vysoké učení technické v Brně

Fakulta informačních technologií

IVS – 2. projekt

Profiling: Výstupní zpráva

18. 4. 2021 R. Marek, J. Dědič, V. Dvořák, T. Dvořák

Obsah zprávy

[Úvod 3](#_Toc69670697)

[Gprof 3](#_Toc69670698)

[Překlad programu 3](#_Toc69670699)

[Profiling 3](#_Toc69670700)

[Výstup 3](#_Toc69670701)

[Callgrind 5](#_Toc69670702)

[Překlad programu 5](#_Toc69670703)

[Profiling 5](#_Toc69670704)

[Výstup 5](#_Toc69670705)

[Interpretace výsledků 10](#_Toc69670706)

[Závěr 10](#_Toc69670707)

[Příloha č. 1 – Skript pro generování vzorků 11](#_Toc69670708)

[Příloha č. 2 – Úplné grafy vygenerované nástroji callgrind a graph2dot 12](#_Toc69670709)

[10 hodnot 12](#_Toc69670710)

[100 hodnot 13](#_Toc69670711)

[1000 hodnot 14](#_Toc69670712)

# Úvod

V následujícím textu je popsán profiling programu *math\_prof,* který byl vytvořen v rámci vývoje kalkulačky *dumdumCalc*. Tento program načítá ze standartního vstupu hodnoty (celá nebo desetinná čísla) a následně z nich pomocí funkcí z matematické knihovny *math\_lib* (taktéž využívané v kalkulačce) počítá tzv. [výběrovou směrodatnou odchylku](https://cs.wikipedia.org/wiki/Sm%C4%9Brodatn%C3%A1_odchylka), kterou poté tiskne na standartní výstup.

Pro profiling byly využity, pro větší směrodatnost výsledků, hned dva programy – *callgrind* a *gprof*. Program byl profilován třikrát (s 10, 100 a 1000 hodnotami) na platformě Ubuntu 20.04 (64bit).

# Gprof

## Překlad programu

Program *math\_prof* byl přeložen pro gprof následujícím způsobem:

g++ -pg -g -O2 -Werror -Wall -pedantic -std=c++11 -c -o build/math\_prof.o math\_prof.cpp

g++ -pg -g -O2 build/math\_prof.o build/math\_lib.o -lm -o build/math\_prof

Přepínač *-O2* byl přidán za účelem automatické optimalizace (a lepších výsledků).

## Profiling

Samotné profiler byl spuštěn touto sekvencí příkazů:

./build/math\_prof < profiling\_sample

gprof -b ./build/math\_prof

Kde soubor *profiling\_sample* byl vzorek s náhodně vygenerovanými 10/100/1000 hodnotami oddělenými náhodným počtem a typem bílých znaků (mezery, tabulátory, nové řádky). Pro účely tvorby vzorků jsme vytvořili jednoduchý skript v jazyce Python (viz [Příloha č. 1](#_Příloha_č._1)).

## Výstup

Po profilování s 10, 100, 1000 byly výstupy programu gprof spojeny do souboru *vystup.txt* a odděleny oddělovači. Pro větší přehlednost zde uvádíme pouze prvních pět řádků Flat profile z každého běhu programu (kompletní výstup je dostupný ve výše zmíněném souboru).

10 hodnot:

% cumulative self self total

time seconds seconds calls Ts/call Ts/call name

0.00 0.00 0.00 20 0.00 0.00 add(double, double)

0.00 0.00 0.00 11 0.00 0.00 f\_pow(double, double)

0.00 0.00 0.00 10 0.00 0.00 void std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >::\_M\_construct<char\*>(char\*, char\*, std::forward\_iterator\_tag)

0.00 0.00 0.00 5 0.00 0.00 void std::vector<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >, std::allocator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > > >::\_M\_realloc\_insert<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&>(\_\_gnu\_cxx::\_\_normal\_iterator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >\*, std::vector<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >, std::allocator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > > > >, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&)

0.00 0.00 0.00 5 0.00 0.00 void std::vector<double, std::allocator<double> >::\_M\_realloc\_insert<double>(\_\_gnu\_cxx::\_\_normal\_iterator<double\*, std::vector<double, std::allocator<double> > >, double&&)

100 hodnot:

% cumulative self self total

time seconds seconds calls Ts/call Ts/call name

0.00 0.00 0.00 200 0.00 0.00 add(double, double)

0.00 0.00 0.00 101 0.00 0.00 f\_pow(double, double)

0.00 0.00 0.00 100 0.00 0.00 void std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >::\_M\_construct<char\*>(char\*, char\*, std::forward\_iterator\_tag)

0.00 0.00 0.00 8 0.00 0.00 void std::vector<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >, std::allocator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > > >::\_M\_realloc\_insert<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&>(\_\_gnu\_cxx::\_\_normal\_iterator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >\*, std::vector<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >, std::allocator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > > > >, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&)

0.00 0.00 0.00 8 0.00 0.00 void std::vector<double, std::allocator<double> >::\_M\_realloc\_insert<double>(\_\_gnu\_cxx::\_\_normal\_iterator<double\*, std::vector<double, std::allocator<double> > >, double&&)

1000 hodnot:

% cumulative self self total

time seconds seconds calls Ts/call Ts/call name

0.00 0.00 0.00 2000 0.00 0.00 add(double, double)

0.00 0.00 0.00 1001 0.00 0.00 f\_pow(double, double)

0.00 0.00 0.00 1000 0.00 0.00 void std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >::\_M\_construct<char\*>(char\*, char\*, std::forward\_iterator\_tag)

0.00 0.00 0.00 11 0.00 0.00 void std::vector<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >, std::allocator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > > >::\_M\_realloc\_insert<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&>(\_\_gnu\_cxx::\_\_normal\_iterator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >\*, std::vector<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >, std::allocator<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > > > >, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&)

0.00 0.00 0.00 11 0.00 0.00 void std::vector<double, std::allocator<double> >::\_M\_realloc\_insert<double>(\_\_gnu\_cxx::\_\_normal\_iterator<double\*, std::vector<double, std::allocator<double> > >, double&&)

# Callgrind

## Překlad programu

Program *math\_prof* byl přeložen pro callgrind následujícím způsobem:

g++ -O2 -Werror -Wall -pedantic -std=c++11 -c -o build/math\_prof.o math\_prof.cpp

g++ -O2 build/math\_prof.o build/math\_lib.o -lm -o build/math\_prof

Oproti překladu pro gprof byly odebrány přepínače *-g* a *-pg*, které pro profilování callgrindem nejsou potřeba a naopak byly příčinou občasných chybových hlášení při jeho běhu.

## Profiling

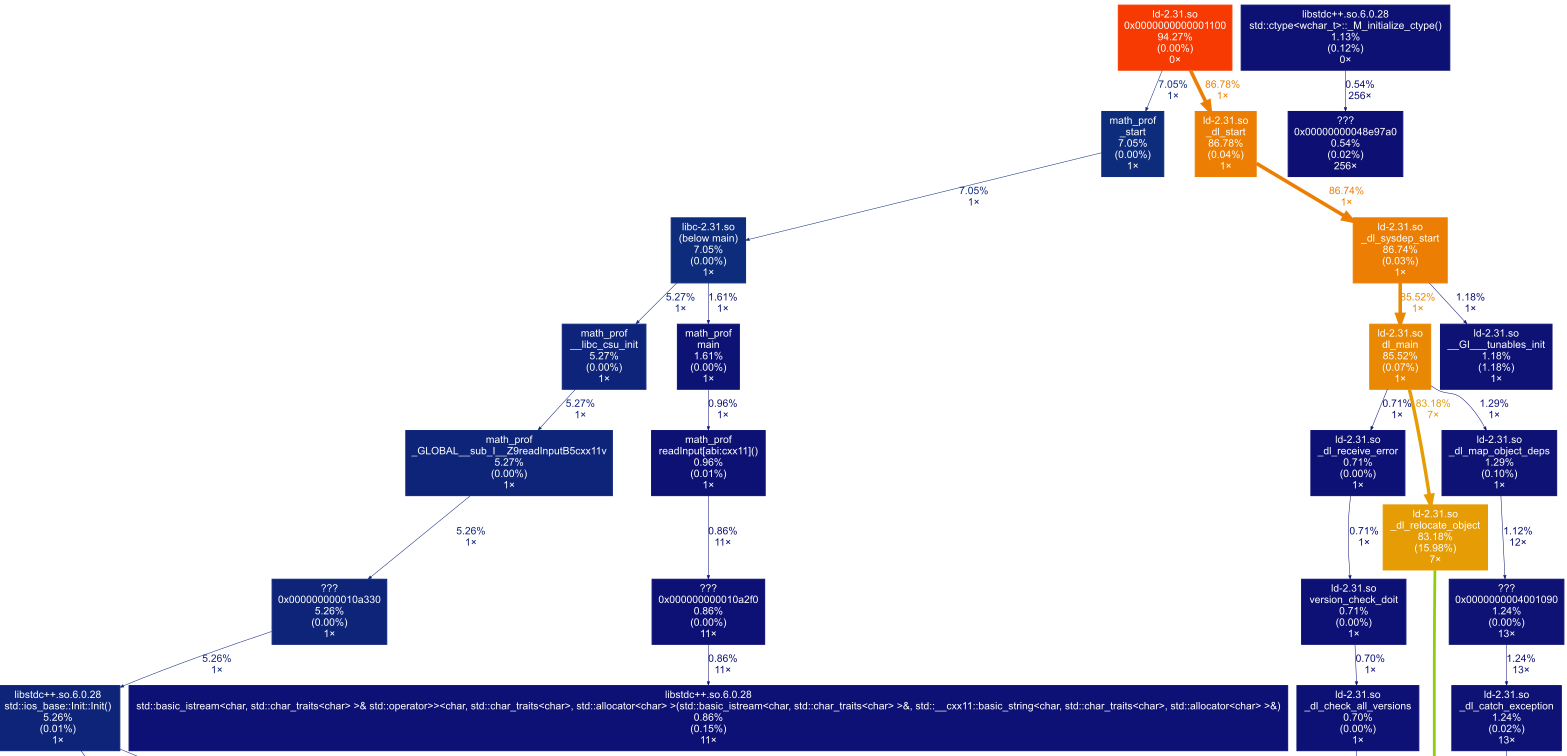
Nástroj callgrind byl po kompilaci spuštěn tímto příkazem:

valgrind --tool=callgrind --callgrind-out-file=callgrind-prof.out\ ./build/math\_prof < profiling\_sample

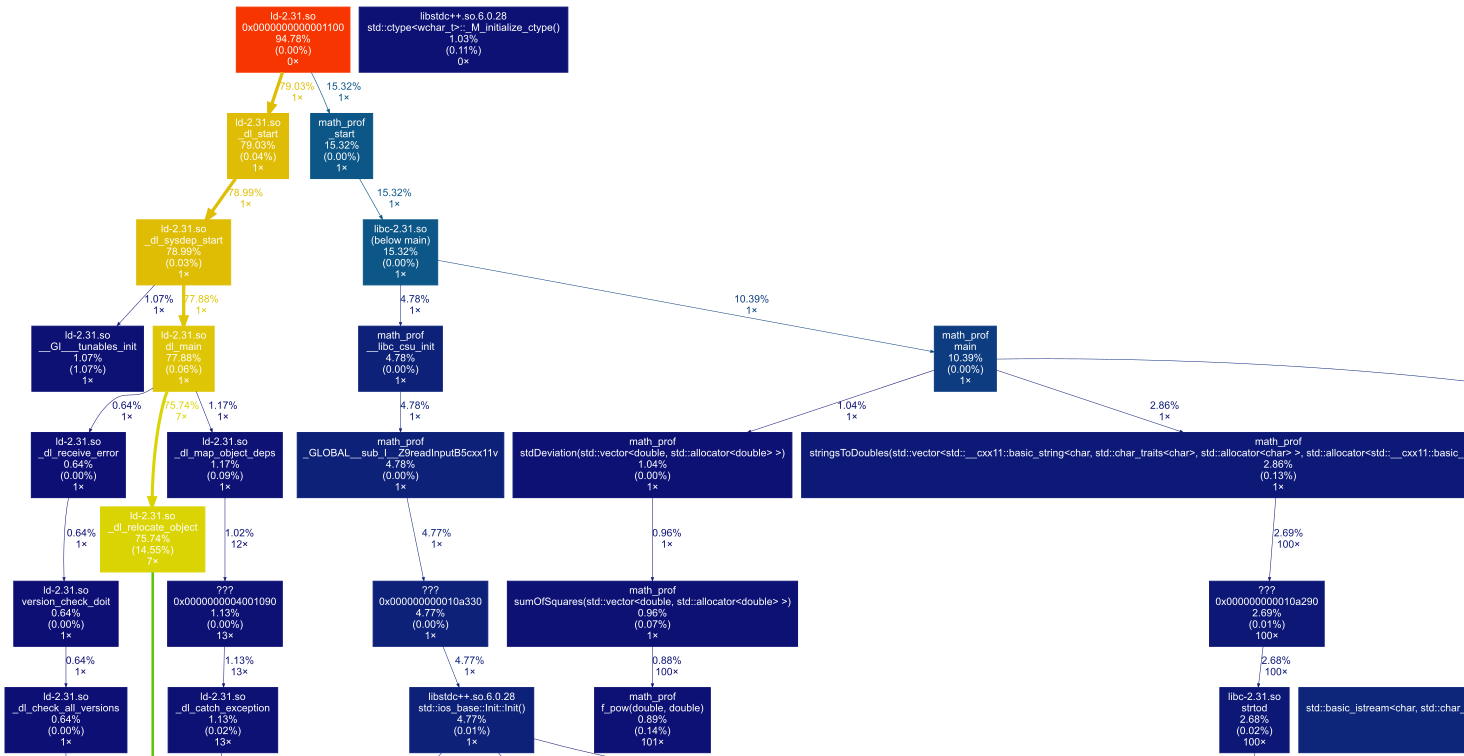
Kde soubor *profiling\_sample* byl vzorek s náhodně vygenerovanými 10/100/1000 hodnotami oddělenými náhodným počtem a typem bílých znaků (mezery, tabulátory, nové řádky).

## Výstup

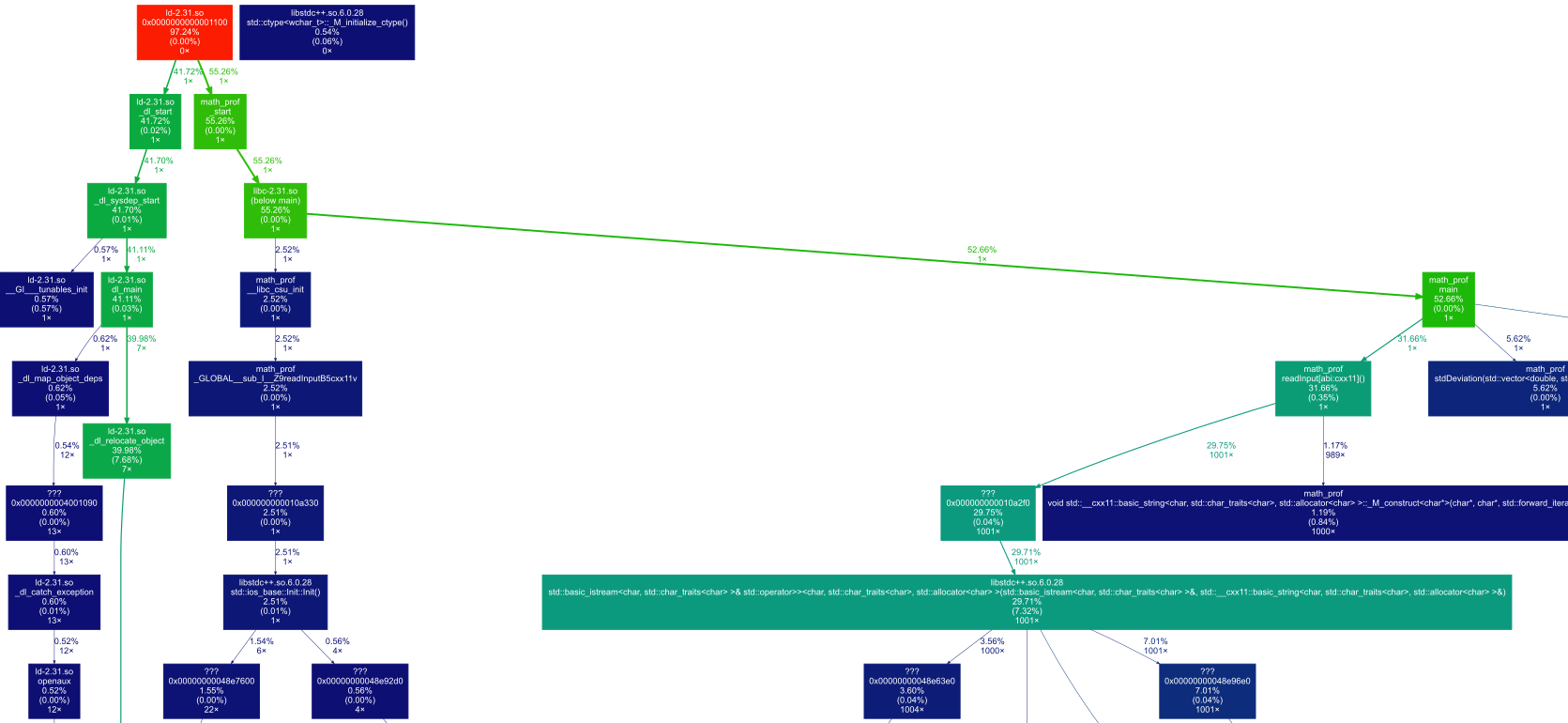
Výsledné soubory v callgrind formátu byly pro lepší čitelnost následně konvertovány nástrojem graph2dot.py do grafické podoby. Pro větší přehlednost uvádíme výstřižky z grafů s kořeny příp. částmi grafu, které zachycují volání matematických funkcí z knihovny (viz následující stránky). Celé grafy je možné si prohlédnout [v příloze](#_Příloha_č._2).



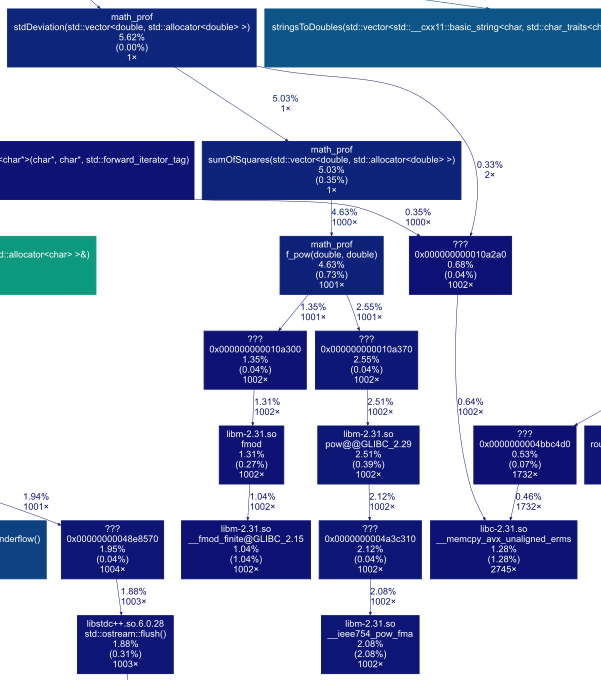
Obrázek 1 - Výsledek profilingu s 10 hodnotami (callgrind)



Obrázek 2 - Výsledek profilingu se 100 hodnotami (callgrind)



Obrázek 3 - Výsledek profilingu s 1000 hodnotami (callgrind)



Obrázek 4 - Výsledek profilingu s 1000 hodnotami (callgrind) - volání matematických funkcí

# Interpretace výsledků

Vzhledem ke krátkému běhu programu, program gprof nenaměřil žádný čas. Z jeho výstupů však poměrně přehledně vidíme nejčastěji volané funkce.

Nejčastěji byly podle gprofu volány funkce *add* (tj. sčítání dvou reálných čísel) a *f\_pow* (umocňování s přirozeným exponentem) z matematické knihovny. S přibývajícím počtem hodnot roste pochopitelně také počet jejich volání, protože jsou tyto funkce volány pro každou načtenou hodnotu ze stdin. Dále jsou pak často volány funkce, které souvisejí s režií tříd *string* a *vector* (realokace, konstruktory)*,* jež slouží pro uložení načtených hodnot.

Absence časových údajů způsobila nedůvěru v gprof a proto jsme se rozhodli získat podrobnější data prostřednictvím callgrindu. Výsledné grafy ukázaly, že se program velkou část doby svého běhu zabývá vyhledáváním funkcí ve sdílených knihovnách a voláním funkcí s tím spojených (např. check\_match). S rostoucím počtem hodnot ve vzorku se však podíl této operace na celkové době běhu snižuje (resp. počet volání zůstává stejný jak pro 10, tak i pro 1000 hodnot), protože se nejspíše jedná o jednorázovou akci.

# Závěr

Při optimalizaci se doporučujeme zaměřit na umocňování, které v tomto případě (pouze umocňování druhou mocninou), lze snadno realizovat např. násobením, jež bude zřejmě ke zdrojům šetrnější.

Dále by bylo vhodné se při optimalizaci zabývat třídami sloužícími pro uchování a čtení hodnot (vector a string). Navrhujeme předávat hodnoty funkcím zásadně prostřednictvím ukazatele (nikoliv skrze hodnoty) nebo např. implementovat vlastní jednodušší struktury. Tyto úpravy by mohly vést ke snížení počtu volaných funkcí (a ke snížení nároků na zásobník).

Nakonec navrhujeme se ujistit, že všechna volání funkcí ze sdílených knihoven (byť standartních) jsou opravdu nutná, popř. počet těchto funkcí zredukovat na minimum. Mohl by se pak snížit čas potřebný k jejich vyhledání pomocí *ld-2.31.so*.

# Příloha č. 1 – Skript pro generování vzorků

Pro generování vzorků využitých při profilování byl využit následující skript v jazyce Python. Přijímá právě jeden argument, tj. počet náhodných hodnot, které mají být vygenerovány, oddělených náhodným počtem a typem bílých znaků.

#!/usr/bin/python

import sys

import random

separators = ["\n", " ", "\t"]

if len(sys.argv) == 1:

print("Missing argument!")

quit(1)

for x in range(int(sys.argv[1])):

number = random.random()\*pow(10, random.randint(0,3))

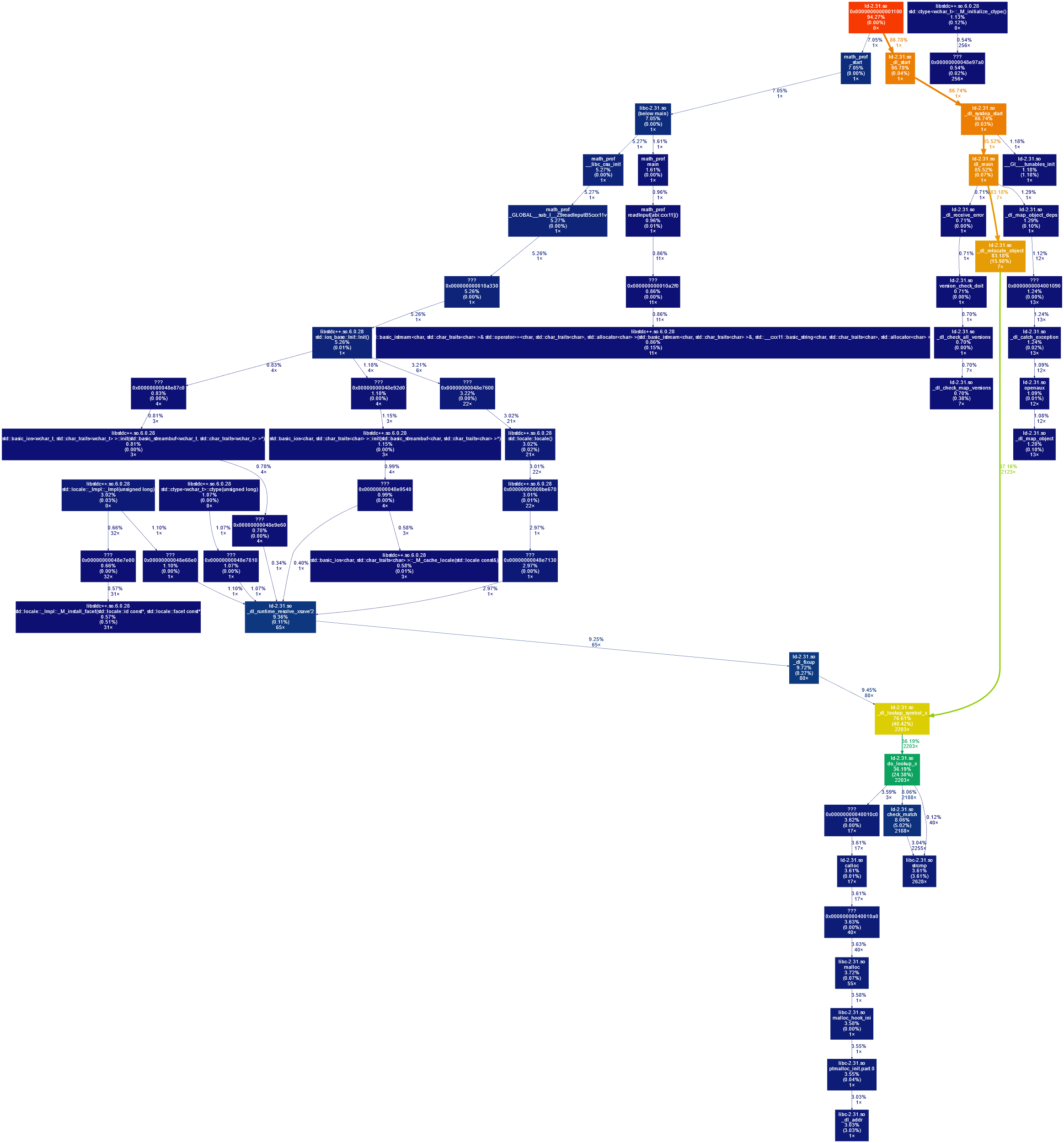
number = round(number, random.randint(0,5))

print(number, end=random.choice(separators))

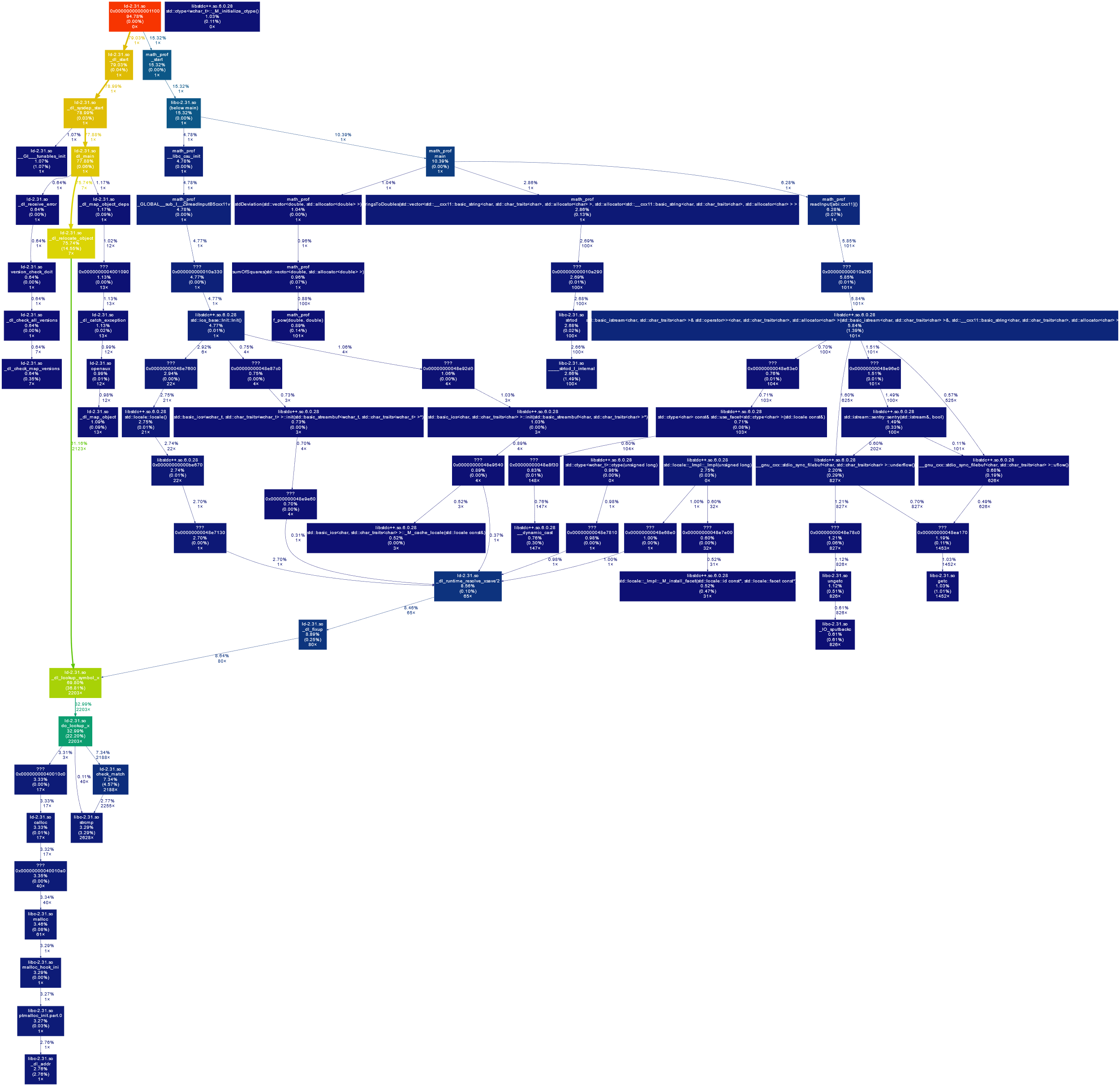
print()

# Příloha č. 2 – Úplné grafy vygenerované nástroji callgrind a graph2dot

## 10 hodnot



## 100 hodnot



## 1000 hodnot

