FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

2023

Profiling

PROFILING OF THE PROGRAM

IVS TADIFAN TEAM

```python
def main():
    if len(sys.argv) < 2:
        print("Error: Input file not specified")
        return
    filename = sys.argv[1]
    with open(filename, "r") as f:
        data = f.read()
        numbers = [float(x) for x in data.split()]
    result = stddev(numbers)
    print("sd =", result)
if __name__ == "__main__":
    main()
```

```
Terminal:  Local
      12    0.000    0.000    0.000    0.000 {method 'rfind' of 'str' objects}
      39    0.000    0.000    0.000    0.000 {method 'rpartition' of 'str' objects}
     328    0.000    0.000    0.000    0.000 {method 'rstrip' of 'str' objects}
       3    0.000    0.000    0.000    0.000 {method 'split' of 'str' objects}
     236    0.000    0.000    0.000    0.000 {method 'startswith' of 'str' objects}

PS C:\IVS\Repr\IVS2_PROJECT_TADIFAN\profiling> python profiling.py data.txt
sd = 63928.48023492956
PS C:\IVS\Repr\IVS2_PROJECT_TADIFAN\profiling>
```

The program "profiling.py" is a Python script that calculates the standard deviation of a set of numerical data provided in a file. The program uses the Calc_Library module, which includes a collection of mathematical functions such as addition, subtraction, multiplication, division, and power.

The "stddev" function in the program implements the standard deviation formula to calculate the statistical dispersion of the input data. The function first calculates the sample mean of the data by dividing the sum of the data by the number of values. The function then computes the difference between each data point and the sample mean and squares the result. The sum of these squared differences is then divided by the number of values to get the variance. Finally, the square root of the variance is calculated to obtain the standard deviation.

The "samplemean" and "summs" functions are helper functions used to calculate the sample mean and the sum of the data, respectively. The "main" function reads in the data from a file specified in the command-line argument and converts the data to a list of floating-point numbers. It then calls the "stddev" function to calculate the standard deviation of the data and prints the result.

The program can be run from the command line using the command "python profiling.py data.txt", where "data.txt" is the name of the file containing the data. To visualize profiling information for the program, the command "python -m cProfile profiling.py data.txt" can be used.

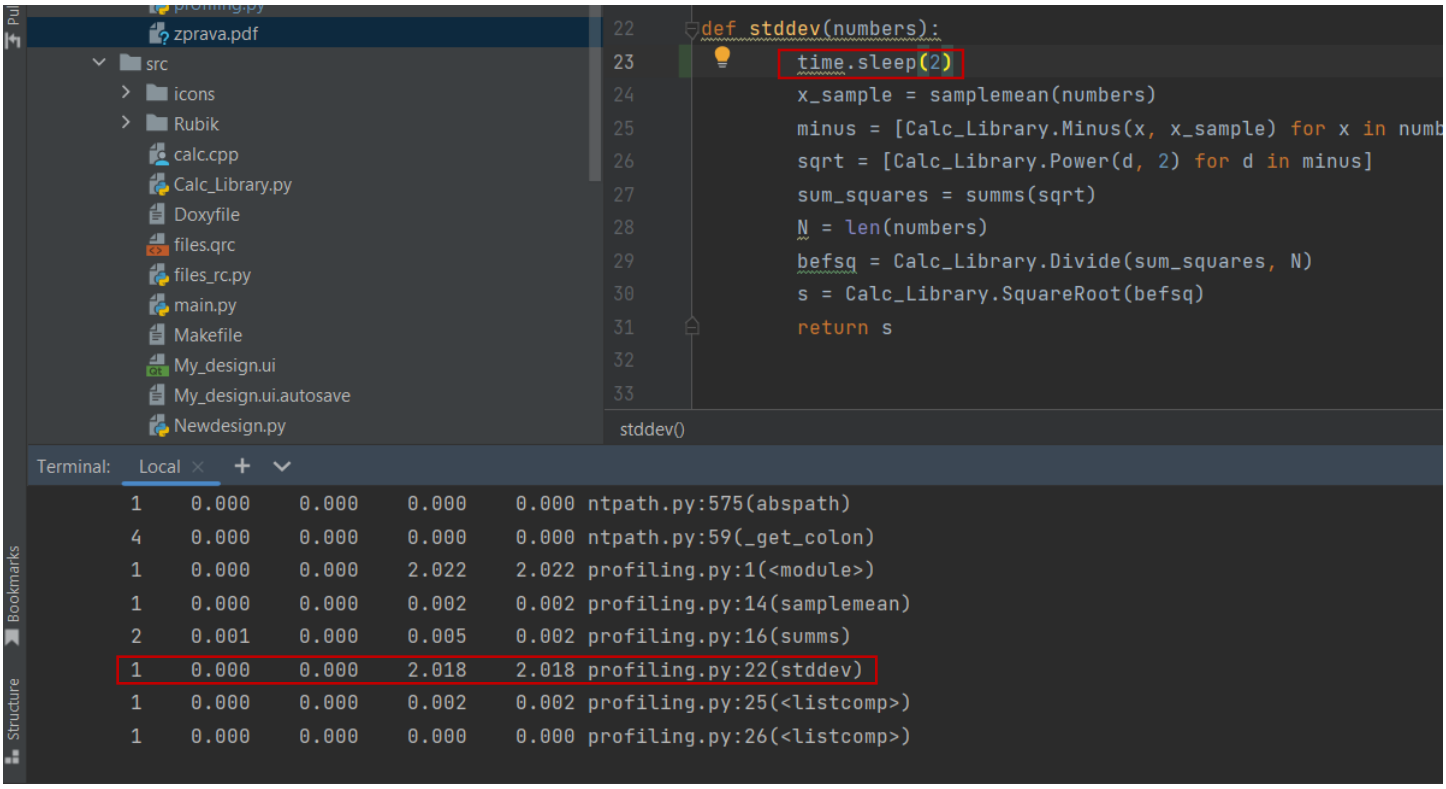
```
  2    0.000    0.000    0.000    0.000 ntpath.py:41(_get_bothseps)
  5    0.000    0.000    0.000    0.000 ntpath.py:47(_get_sep)
  1    0.000    0.000    0.000    0.000 ntpath.py:503(normpath)
  4    0.000    0.000    0.000    0.000 ntpath.py:53(_get_altsep)
  1    0.000    0.000    0.000    0.000 ntpath.py:575(abspath)
  4    0.000    0.000    0.000    0.000 ntpath.py:59(_get_colon)
  1    0.000    0.000    0.009    0.009 profiling.py:1(<module>)
  1    0.000    0.000    0.001    0.001 profiling.py:12(samplemean)
  2    0.000    0.000    0.004    0.002 profiling.py:14(summs)
  1    0.000    0.000    0.006    0.006 profiling.py:20(stddev)
  1    0.000    0.000    0.002    0.002 profiling.py:22(<listcomp>)
  1    0.000    0.000    0.000    0.000 profiling.py:23(<listcomp>)
  1    0.000    0.000    0.007    0.007 profiling.py:32(main)
  1    0.000    0.000    0.000    0.000 profiling.py:39(<listcomp>)
  1    0.000    0.000    0.000    0.000 {built-in method __new__ of type object at 0x00007ffd
  1    0.000    0.000    0.000    0.000 {built-in method _codecs.charmap_decode}
  1    0.000    0.000    0.000    0.000 {built-in method _imp._fix_co_filename}
 12    0.000    0.000    0.000    0.000 {built-in method _imp.acquire_lock}
  1    0.000    0.000    0.000    0.000 {built-in method _imp.is_builtin}
  2    0.000    0.000    0.000    0.000 {built-in method _imp.is_frozen}
 12    0.000    0.000    0.000    0.000 {built-in method _imp.release_lock}
  4    0.000    0.000    0.000    0.000 {built-in method _thread.allocate_lock}
  4    0.000    0.000    0.000    0.000 {built-in method _thread.get_ident}
  1    0.000    0.000    0.001    0.001 {built-in method builtins.__import__}
2/1    0.000    0.000    0.009    0.009 {built-in method builtins.exec}
 13    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
 13    0.000    0.000    0.000    0.000 {built-in method builtins.hasattr}
 30    0.000    0.000    0.000    0.000 {built-in method builtins.isinstance}
  1    0.000    0.000    0.000    0.000 {built-in method builtins.iter}
```

```
  2    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:1632(path_hook_for_FileFinder)
  3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:168(_path_isdir)
  3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:176(_path_isabs)
  2    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:384(cache_from_source)
  1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:514(_get_cached)
  1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:546(_check_name_wrapper)
  1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:589(_classify_pyc)
  1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:622(_validate_timestamp_pyc)
  1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:674(_compile_bytecode)
  8    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:71(_relax_case)
  1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:725(spec_from_file_location)
  1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:878(create_module)
  3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:88(_unpack_uint32)
  1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap_external>:881(exec_module)
  1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap_external>:954(get_code)
  2    0.000    0.000    0.000    0.000 <frozen zipimport>:64(__init__)
  1    0.000    0.000    0.000    0.000 Calc_Library.py:1(<module>)
1000    0.000    0.000    0.000    0.000 Calc_Library.py:11(Power)
2000    0.002    0.000    0.003    0.000 Calc_Library.py:14(Plus)
1000    0.001    0.000    0.002    0.000 Calc_Library.py:22(Minus)
  1    0.000    0.000    0.000    0.000 Calc_Library.py:39(Divide)
  1    0.000    0.000    0.000    0.000 Calc_Library.py:8(SquareRoot)
  2    0.000    0.000    0.000    0.000 __init__.py:89(find_spec)
  1    0.000    0.000    0.000    0.000 __init__.py:96(<lambda>)
  1    0.000    0.000    0.000    0.000 codecs.py:260(__init__)
  1    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
  1    0.000    0.000    0.000    0.000 ntpath.py:127(join)
  4    0.000    0.000    0.000    0.000 ntpath.py:169(splitdrive)
  1    0.000    0.000    0.000    0.000 ntpath.py:220(split)
```

Red highlights the functions that take the most time. As you can see the program runs very fast, I checked for changes in function execution time by adding 'time.sleep'.

Below I run the program with the same input arguments and call the 'time.sleep' function in the main 'stdevv' function. As you can see, the function execution time has changed to 2.

Thus, I checked whe ther the profiling was taking place correctly.


zprava.pdf
src
icons
Rubik
calc.cpp
Calc_Library.py
Doxyfile
files.qrc
files_rc.py
main.py
Makefile
My_design.ui
My_design.ui.autosave
Newdesign.py
22 def stddev(numbers):
23 time.sleep(2)
24 x_sample = samplemean(numbers)
25 minus = [Calc_Library.Minus(x, x_sample) for x in numb
26 sqrt = [Calc_Library.Power(d, 2) for d in minus]
27 sum_squares = summs(sqrt)
28 N = len(numbers)
29 befsq = Calc_Library.Divide(sum_squares, N)
30 s = Calc_Library.SquareRoot(befsq)
31 return s
stddev()
Terminal: Local
1 0.000 0.000 0.000 0.000 ntpath.py:575(abspath)
4 0.000 0.000 0.000 0.000 ntpath.py:59(_get_colon)
1 0.000 0.000 2.022 2.022 profiling.py:1(<module>)
1 0.000 0.000 0.002 0.002 profiling.py:14(samplemean)
2 0.001 0.000 0.005 0.002 profiling.py:16(summs)
1 0.000 0.000 2.018 2.018 profiling.py:22(stddev)
1 0.000 0.000 0.002 0.002 profiling.py:25(<listcomp>)
1 0.000 0.000 0.000 0.000 profiling.py:26(<listcomp>)