2020_profile

July 1, 2022

1 Tech - profiling

Le profilage de code ou *profiling* en anglais signifie qu'on mesure le temps passé dans chaque partie d'un programme pour en découvrir les parties les plus coûteuses et les améliorer. On souhaite toujours accélérer un programme trop lent, le profiling permet de savoir sur quelles parties se concentrer.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

Autre lecture : Exemple de profiling.

1.1 Enoncé

On applique le profiling au code suivant pour découvrir laquelle de ces fonctions est la plus lente. Vous pouvez changer ce code pour un de ceux que vous avez écrits précédemment.

[2]:
```
def _gini_sort(Y):
    return list(sorted(Y))


def _gini_init(Y):
    return [[1, y] for y in Y]


def _gini_cumsum(couples):
    for i in range(1, len(couples)):
        couples[i][0] += couples[i-1][0]
        couples[i][1] += couples[i-1][1]
    for i in range(0, len(couples)):
        couples[i][0] /= couples[-1][0]
        couples[i][1] /= couples[-1][1]
    return couples


def _gini_final(couples):
    g = couples[0][0] * couples[0][1]
    n = len(couples)

    for i in range(1, n):
        dx = couples[i][0] - couples[i-1][0]
        y = couples[i-1][1] + couples[i][1]
```

```
        g += dx * y

    return 1. - g / 2


def gini(Y):
    Y = _gini_sort(Y)
    couples = _gini_init(Y)
    couples = _gini_cumsum(couples)
    return _gini_final(couples)


gini([1, 1, 1, 1, 1]), gini([0, 0, 0, 0, 100000])
```

[2]: (0.5, 0.9)

Pour ces deux exemples caractéristiques, on retrouve bien des valeurs attendues.

1.1.1 Exercice 1 : profiler le code précédent

Le langage python dispose d'un module qui mesure le temps passé dans chaque fonction profile. Il faut l'utiliser.

[3]:

1.1.2 Exercice 2 : changer la fonction _gini_final si possible en plus rapide

[4]:

1.1.3 Exercice 3 : vous améliorez la fonction _gini_final, profilez pour savoir de combien ?

[5]:

1.1.4 Exercice 4 : utiliser d'autres modules de profiling

Si c'est possible, comme pyinstrument ou py-spy. Il y a deux façons de faire du profiling :

- **déterministe :** on mesure le temps passé dans chaque fonction, à chaque appel, ce type de profilage n'est souvent possible que pour des langages interprétés,
- **statistique :** tous les centièmes de secondes, on regarde quelle ligne de quelle fonction le programme exécute. On compte ensuite combien de fois la ligne exécutée était dans une fonction pour déterminer le temps passé dans chaque fonction.

Les deux modules proposés sont statistiques, le premier est déterministe. Bien évidemment, le fait de mesurer le temps passé prend du temps également, plus on s'arrête souvant, plus l'exécution est ralentie.

[6]:

1.1.5 Exercice 5 : la fonction _gini_cumsum contient deux boucles. Quelle est la plus rapide ?

[7]:

1.2 Réponses

1.2.1 Exercice 1 : profiler le code précédent

On reprend un des exemples de cette page The Python Profilers.

[8]:
```
import cProfile, pstats, io
from pstats import SortKey
pr = cProfile.Profile()
pr.enable()

gini([1, 1, 1, 1, 1])

pr.disable()
s = io.StringIO()
sortby = SortKey.CUMULATIVE
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
ps.print_stats()
print(s.getvalue())
```

```
         55 function calls in 0.000 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\interactiveshell.py:3400(run_code)
        2    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 <ipython-
input-3-405f7b24fbdc>:8(<module>)
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\codeop.py:142(__call__)
        2    0.000    0.000    0.000    0.000 {built-in method builtins.compile}
        1    0.000    0.000    0.000    0.000 <ipython-
input-3-405f7b24fbdc>:6(<module>)
        1    0.000    0.000    0.000    0.000 <ipython-
input-2-6d8b3d4c048c>:31(gini)
        1    0.000    0.000    0.000    0.000 <ipython-
input-2-6d8b3d4c048c>:9(_gini_cumsum)
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:242(helper)
        4    0.000    0.000    0.000    0.000 {built-in method builtins.next}
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:86(__init__)
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:112(__enter__)
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:121(__exit__)
        1    0.000    0.000    0.000    0.000 <ipython-
input-2-6d8b3d4c048c>:19(_gini_final)
        4    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\compilerop.py:166(extra_flags)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\hooks.py:103(__call__)
        1    0.000    0.000    0.000    0.000 <ipython-
```

```
input-2-6d8b3d4c048c>:1(_gini_sort)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\traitlets\traitlets.py:564(__get__)
        1    0.000    0.000    0.000    0.000 <ipython-
input-2-6d8b3d4c048c>:5(_gini_init)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\interactiveshell.py:3338(compare)
        4    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\traitlets\traitlets.py:533(get)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\utils\ipstruct.py:125(__getattr__)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.sorted}
        1    0.000    0.000    0.000    0.000 <ipython-
input-2-6d8b3d4c048c>:6(<listcomp>)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\interactiveshell.py:1278(user_global_ns)
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
        3    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\hooks.py:168(pre_run_code_hook)
```

Tous les temps sont nuls. Le code est trop rapide. Il faut soit exécuter le code plusieurs fois, soit prendre un tableau plus grand.

[9]:
```
import cProfile, pstats, io
from pstats import SortKey
pr = cProfile.Profile()
pr.enable()

for i in range(1000):
    gini([1 for i in range(1000)])

pr.disable()
s = io.StringIO()
sortby = SortKey.CUMULATIVE
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
ps.print_stats()
print(s.getvalue())
```

```
         11045 function calls in 1.352 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        2    0.000    0.000    1.352    0.676 c:\python395_x64\lib\site-
packages\IPython\core\interactiveshell.py:3400(run_code)
        2    0.000    0.000    1.352    0.676 {built-in method builtins.exec}
        1    0.040    0.040    1.352    1.352 <ipython-
input-4-b1075e275d99>:6(<module>)
```

```
     1000    0.003    0.000    1.266    0.001 <ipython-
input-2-6d8b3d4c048c>:31(gini)
     1000    0.706    0.001    0.707    0.001 <ipython-
input-2-6d8b3d4c048c>:9(_gini_cumsum)
     1000    0.357    0.000    0.357    0.000 <ipython-
input-2-6d8b3d4c048c>:19(_gini_final)
     1000    0.001    0.000    0.185    0.000 <ipython-
input-2-6d8b3d4c048c>:5(_gini_init)
     1000    0.184    0.000    0.184    0.000 <ipython-
input-2-6d8b3d4c048c>:6(<listcomp>)
     1000    0.047    0.000    0.047    0.000 <ipython-
input-4-b1075e275d99>:7(<listcomp>)
     1000    0.005    0.000    0.013    0.000 <ipython-
input-2-6d8b3d4c048c>:1(_gini_sort)
     1000    0.008    0.000    0.008    0.000 {built-in method builtins.sorted}
     3000    0.001    0.000    0.001    0.000 {built-in method builtins.len}
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\codeop.py:142(__call__)
        2    0.000    0.000    0.000    0.000 {built-in method builtins.compile}
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:242(helper)
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:86(__init__)
        4    0.000    0.000    0.000    0.000 {built-in method builtins.next}
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:121(__exit__)
        2    0.000    0.000    0.000    0.000
c:\python395_x64\lib\contextlib.py:112(__enter__)
        4    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\compilerop.py:166(extra_flags)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\hooks.py:103(__call__)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\traitlets\traitlets.py:564(__get__)
        1    0.000    0.000    0.000    0.000 <ipython-
input-4-b1075e275d99>:9(<module>)
        4    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\interactiveshell.py:3338(compare)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\utils\ipstruct.py:125(__getattr__)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\traitlets\traitlets.py:533(get)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\interactiveshell.py:1278(user_global_ns)
        2    0.000    0.000    0.000    0.000 c:\python395_x64\lib\site-
packages\IPython\core\hooks.py:168(pre_run_code_hook)
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
```

C'est un peu illisible. On filtre.

[10]:
```
def filtre(text):
    lines = text.split("\n")
    return "\n".join(filter(lambda t: "gini" in t, lines))

print(filtre(s.getvalue()))
```

```
     1000    0.003    0.000    1.266    0.001 <ipython-
input-2-6d8b3d4c048c>:31(gini)
     1000    0.706    0.001    0.707    0.001 <ipython-
input-2-6d8b3d4c048c>:9(_gini_cumsum)
     1000    0.357    0.000    0.357    0.000 <ipython-
input-2-6d8b3d4c048c>:19(_gini_final)
     1000    0.001    0.000    0.185    0.000 <ipython-
input-2-6d8b3d4c048c>:5(_gini_init)
     1000    0.005    0.000    0.013    0.000 <ipython-
input-2-6d8b3d4c048c>:1(_gini_sort)
```

1.2.2 Exercice 2 : changer la fonction _gini_final si possible en plus rapide

[11]:
```
def _gini_final_faster(couples):
    g = couples[0][0] * couples[0][1]
    n = len(couples)

    # par construction, tous les dx sont identiques, cela fait des calculs en moins
    # le code suivant est plus rapide mais peut encore être optimisé
    dx = couples[0][0]
    sy = couples[0][0]
    for i in range(1, n):
        sy = couples[i-1][1] + couples[i][1]
        g += dx * sy

    return 1. - g / 2


def gini_faster(Y):
    Y = _gini_sort(Y)
    couples = _gini_init(Y)
    couples = _gini_cumsum(couples)
    return _gini_final_faster(couples)


(gini_faster([0, 0, 0, 0, 5000]), gini_faster([1, 1, 1, 1, 1]),
 gini([0, 0, 0, 0, 5000]), gini([1, 1, 1, 1, 1]))
```

[11]:
```
(0.9, 0.5, 0.9, 0.5)
```

1.2.3 Exercice 3 : vous améliorez la fonction _gini_final, profilez pour savoir de combien ?

On appelle les deux fonctions pour pouvoir comparer.

[12]:
```
import cProfile, pstats, io
from pstats import SortKey
pr = cProfile.Profile()
pr.enable()
```

```
for i in range(1000):
    gini([1 for i in range(1000)])
    gini_faster([1 for i in range(1000)])

pr.disable()
s = io.StringIO()
sortby = SortKey.CUMULATIVE
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
ps.print_stats()
print(filtre(s.getvalue()))
```

```
     2000    1.085    0.001    1.086    0.001 <ipython-
input-2-6d8b3d4c048c>:9(_gini_cumsum)
     1000    0.003    0.000    0.999    0.001 <ipython-
input-2-6d8b3d4c048c>:31(gini)
     1000    0.003    0.000    0.882    0.001 <ipython-
input-6-a3c3cc48873e>:16(gini_faster)
     2000    0.002    0.000    0.315    0.000 <ipython-
input-2-6d8b3d4c048c>:5(_gini_init)
     1000    0.275    0.000    0.275    0.000 <ipython-
input-2-6d8b3d4c048c>:19(_gini_final)
     1000    0.174    0.000    0.175    0.000 <ipython-
input-6-a3c3cc48873e>:1(_gini_final_faster)
     2000    0.009    0.000    0.024    0.000 <ipython-
input-2-6d8b3d4c048c>:1(_gini_sort)
```

La seconde version est plus rapide.

1.2.4 Exercice 4 : utiliser d'autres modules de profiling

pyinstrument n'est pas peut-être pas installé. Il fonctionne de la même façon.

[13]:
```
from pyinstrument import Profiler
from IPython.display import HTML

profiler = Profiler()
profiler.start()

for i in range(1000):
    gini([1 for i in range(1000)])
    gini_faster([1 for i in range(1000)])

profiler.stop()

HTML(profiler.output_html())
```

[13]:
```
<IPython.core.display.HTML object>
```

C'est plus facile à lire.

1.2.5 Exercice 5 : la fonction _gini_cumsum contient deux boucles. Quelle est la plus rapide ?

Pour ce faire, on découpe la fonction en deux.

[14]:
```
def _gini_cumsum2a(couples):
    for i in range(1, len(couples)):
        couples[i][0] += couples[i-1][0]
        couples[i][1] += couples[i-1][1]
    return couples


def _gini_cumsum2b(couples):
    for i in range(0, len(couples)):
        couples[i][0] /= couples[-1][0]
        couples[i][1] /= couples[-1][1]
    return couples


def _gini_cumsum2(couples):
    _gini_cumsum2a(couples)
    _gini_cumsum2b(couples)
    return couples


def gini2(Y):
    Y = _gini_sort(Y)
    couples = _gini_init(Y)
    couples = _gini_cumsum2(couples)
    return _gini_final(couples)


gini2([1, 1, 1, 1, 1]), gini2([0, 0, 0, 0, 100000])
```

[14]:
```
(0.5, 0.9)
```

[15]:
```
import cProfile, pstats, io
from pstats import SortKey
pr = cProfile.Profile()
pr.enable()

for i in range(1000):
    gini2([1 for i in range(1000)])
    gini_faster([1 for i in range(1000)])

pr.disable()
s = io.StringIO()
sortby = SortKey.CUMULATIVE
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
ps.print_stats()
print(filtre(s.getvalue()))
```

```
     1000    0.003    0.000    1.144    0.001 <ipython-
input-9-2fd5204eb0f7>:21(gini2)
     1000    0.003    0.000    1.038    0.001 <ipython-
input-6-a3c3cc48873e>:16(gini_faster)
     1000    0.001    0.000    0.641    0.001 <ipython-
input-9-2fd5204eb0f7>:15(_gini_cumsum2)
```

```
     1000    0.637    0.001    0.638    0.001 <ipython-
input-2-6d8b3d4c048c>:9(_gini_cumsum)
     2000    0.002    0.000    0.341    0.000 <ipython-
input-2-6d8b3d4c048c>:5(_gini_init)
     1000    0.332    0.000    0.332    0.000 <ipython-
input-2-6d8b3d4c048c>:19(_gini_final)
     1000    0.326    0.000    0.326    0.000 <ipython-
input-9-2fd5204eb0f7>:1(_gini_cumsum2a)
     1000    0.314    0.000    0.314    0.000 <ipython-
input-9-2fd5204eb0f7>:8(_gini_cumsum2b)
     1000    0.200    0.000    0.200    0.000 <ipython-
input-6-a3c3cc48873e>:1(_gini_final_faster)
     2000    0.009    0.000    0.024    0.000 <ipython-
input-2-6d8b3d4c048c>:1(_gini_sort)
```

Elles sont aussi rapides l'une que l'autre. On peut néanmoins en accélérer une puisqu'on divise tous les éléments d'un tableau par une même valeur. On peut la stocker dans une variable plutôt que d'aller la chercher à chaque fois dans le tableau.

[16]:
```
def _gini_cumsum3b(couples):
    total0 = couples[-1][0]
    total1 = couples[-1][1]
    for i in range(0, len(couples)):
        couples[i][0] /= total0
        couples[i][1] /= total1
    return couples


def _gini_cumsum3(couples):
    _gini_cumsum2a(couples)
    _gini_cumsum3b(couples)
    return couples


def gini3(Y):
    Y = _gini_sort(Y)
    couples = _gini_init(Y)
    couples = _gini_cumsum3(couples)
    return _gini_final(couples)


gini2([1, 1, 1, 1, 1]), gini2([0, 0, 0, 0, 100000])
```

[16]:
```
(0.5, 0.9)
```

[17]:
```
import cProfile, pstats, io
from pstats import SortKey
pr = cProfile.Profile()
pr.enable()

for i in range(1000):
    gini2([1 for i in range(1000)])
    gini3([1 for i in range(1000)])

pr.disable()
```

```
s = io.StringIO()
sortby = SortKey.CUMULATIVE
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
ps.print_stats()
print(filtre(s.getvalue()))
```

```
     1000    0.002    0.000    1.002    0.001 <ipython-
input-9-2fd5204eb0f7>:21(gini2)
     1000    0.002    0.000    0.935    0.001 <ipython-
input-11-492a2c865fd9>:16(gini3)
     2000    0.568    0.000    0.568    0.000 <ipython-
input-2-6d8b3d4c048c>:19(_gini_final)
     2000    0.567    0.000    0.567    0.000 <ipython-
input-9-2fd5204eb0f7>:1(_gini_cumsum2a)
     1000    0.001    0.000    0.561    0.001 <ipython-
input-9-2fd5204eb0f7>:15(_gini_cumsum2)
     1000    0.001    0.000    0.496    0.000 <ipython-
input-11-492a2c865fd9>:10(_gini_cumsum3)
     2000    0.002    0.000    0.285    0.000 <ipython-
input-2-6d8b3d4c048c>:5(_gini_init)
     1000    0.275    0.000    0.275    0.000 <ipython-
input-9-2fd5204eb0f7>:8(_gini_cumsum2b)
     1000    0.213    0.000    0.213    0.000 <ipython-
input-11-492a2c865fd9>:1(_gini_cumsum3b)
     2000    0.008    0.000    0.022    0.000 <ipython-
input-2-6d8b3d4c048c>:1(_gini_sort)
```

C'est mieux même si l'amélioration ne paraît pas nécessairement significative par rapport au temps total, le résultat l'est si on regarde le temps fonction par fonction.

[18]: