Отчет по домашнему заданию №9

1. Сравнение использования weakref и слотов

Для чистоты эксперимента произведем несколько запусков программы.

Результаты запуска программы с использованием обычных атрибутов смотри на Рисунке 1.

```
(.venv) (base) →  09 git:(main) × python3 ordinary.py
time of creating 2000000 objects --- 1.4142708778381348 seconds ---
time of access to 2000000 objects --- 0.6458780765533447 seconds ---
time of modifying 2000000 objects --- 0.5183689594268799 seconds ---
time of deleting 2000000 objects --- 0.36330389976501465 seconds ---
(.venv) (base) →  09 git:(main) × python3 ordinary.py
time of creating 2000000 objects --- 1.4175007343292236 seconds ---
time of access to 2000000 objects --- 0.37233829498291016 seconds ---
time of modifying 2000000 objects --- 1.8045639991760254 seconds ---
time of deleting 2000000 objects --- 0.8134441375732422 seconds ---
(.venv) (base) →  09 git:(main) × python3 ordinary.py
time of creating 2000000 objects --- 1.4070122241973877 seconds ---
time of access to 2000000 objects --- 0.45262694358825684 seconds ---
time of modifying 2000000 objects --- 0.5474660396575928 seconds ---
time of deleting 2000000 objects --- 0.3507518768310547 seconds ---
(.venv) (base) →  09 git:(main) × python3 ordinary.py
time of creating 2000000 objects --- 1.5022950172424316 seconds ---
time of access to 2000000 objects --- 0.28736019134521484 seconds ---
time of modifying 2000000 objects --- 0.4759228229522705 seconds ---
time of deleting 2000000 objects --- 0.5058109760284424 seconds ---
(.venv) (base) →  09 git:(main) × python3 ordinary.py
time of creating 2000000 objects --- 1.5754668712615967 seconds ---
time of access to 2000000 objects --- 0.3530080318450928 seconds ---
time of modifying 2000000 objects --- 0.48546886444091797 seconds ---
time of deleting 2000000 objects --- 0.5776171684265137 seconds ---
```

Рисунок 1 – Время работы программы с использованием обычных атрибутов

Результаты запуска программы с использованием слотов смотри на Рисунке 2.

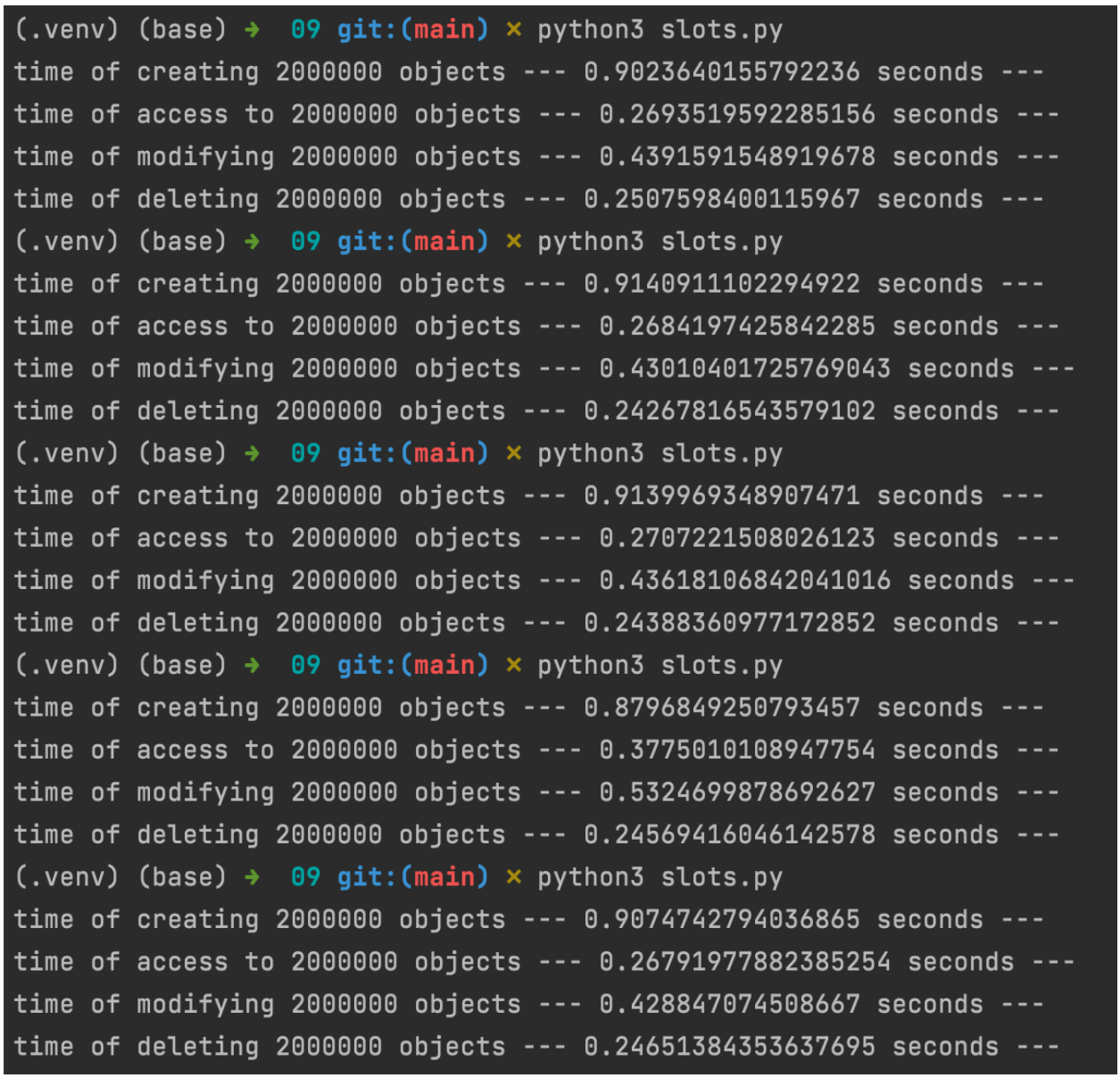

Рисунок 2 – Время работы программы с использованием слотов

Результаты запуска программы с использованием weakref смотри на Рисунке 3.

```
(.venv) (base) ➜  09 git:(main) ✗ python3 weakrefs.py
time of creating 2000000 objects --- 6.709449052810669 seconds ---
time of access to 2000000 objects --- 0.8732359409332275 seconds ---
time of modifying 2000000 objects --- 1.59647798538208 seconds ---
time of deleting 2000000 objects --- 1.616779088973999 seconds ---
(.venv) (base) ➜  09 git:(main) ✗ python3 weakrefs.py
time of creating 2000000 objects --- 3.2852728366851807 seconds ---
time of access to 2000000 objects --- 0.6943681240081787 seconds ---
time of modifying 2000000 objects --- 1.4159801006317139 seconds ---
time of deleting 2000000 objects --- 1.4873299598693848 seconds ---
(.venv) (base) ➜  09 git:(main) ✗ python3 weakrefs.py
time of creating 2000000 objects --- 5.950564861297607 seconds ---
time of access to 2000000 objects --- 0.9690542221069336 seconds ---
time of modifying 2000000 objects --- 1.8319199085235596 seconds ---
time of deleting 2000000 objects --- 1.7476670742034912 seconds ---
(.venv) (base) ➜  09 git:(main) ✗ python3 weakrefs.py
time of creating 2000000 objects --- 4.208540916442871 seconds ---
time of access to 2000000 objects --- 0.5751402378082275 seconds ---
time of modifying 2000000 objects --- 0.9855737686157227 seconds ---
time of deleting 2000000 objects --- 1.105882167816162 seconds ---
(.venv) (base) ➜  09 git:(main) ✗ python3 weakrefs.py
time of creating 2000000 objects --- 7.156334161758423 seconds ---
time of access to 2000000 objects --- 0.9145429134368896 seconds ---
time of modifying 2000000 objects --- 1.796738862991333 seconds ---
time of deleting 2000000 objects --- 1.4857780933380127 seconds ---
```

Рисунок 3 – Время работы программы с использованием weakref

Вывод: по замерам видно, что с использованием слотов создание и удаление объектов происходит значительно быстрее. С использованием weakref все операции сильно замедляются.

Профилирование вызовов и памяти в программе.

Профилирование вызовов и памяти в программе и использованием слотов смотри на рисунке 4.

```
49     44.7 MiB   -142.7 MiB      200004       flats = [Flat(all_windows[i], all_floors[i], all_rooms[i], all_doors[i])
50     44.7 MiB    -75.0 MiB      100001                for i in range(times_of_run)
51                                                       ]
52
53     44.7 MiB     -0.1 MiB           1       time_2 = time.time()
54     44.8 MiB      0.1 MiB           2       print(f"time of creating {times_of_run} objects "
55     44.7 MiB      0.0 MiB           1             f"--- {time_2 - time_1} seconds ---")
56
57     44.8 MiB -980180.6 MiB      100001       for i in range(times_of_run):
58     44.8 MiB -980167.9 MiB      100000           if flats[i].floor is not None:
59                                                     pass
```

Рисунок 4 – Профилирование вызовов и памяти в программе и использованием слотов

Профилирование вызовов и памяти в программе и использованием обычных атрибутов смотри на рисунке 5.

```
48     56.8 MiB  -4192.2 MiB      200004       flats = [Flat(all_windows[i], all_floors[i], all_rooms[i], all_doors[i])
49     56.8 MiB  -2104.0 MiB      100001                for i in range(times_of_run)
50                                                       ]
51
52     56.8 MiB      0.0 MiB           1       time_2 = time.time()
53     56.9 MiB      0.1 MiB           2       print(f"time of creating {times_of_run} objects "
54     56.8 MiB      0.0 MiB           1             f"--- {time_2 - time_1} seconds ---")
55
56     57.0 MiB -87405.3 MiB      100001       for i in range(times_of_run):
57     57.0 MiB -87404.4 MiB      100000           if flats[i].floor is not None:
58                                                     pass
```

Рисунок 5 – Профилирование вызовов и памяти в программе и использованием обычных атрибутов

Профилирование вызовов и памяти в программе и использованием weakref смотри на рисунке 6.

```
49     88.2 MiB     47.5 MiB      200004       flats = [Flat(all_windows[i], all_floors[i], all_rooms[i], all_doors[i])
50     88.2 MiB      0.0 MiB      100001                for i in range(times_of_run)
51                                                       ]
52
53     88.2 MiB      0.0 MiB           1       time_2 = time.time()
54     88.2 MiB      0.0 MiB           2       print(f"time of creating {times_of_run} objects "
55     88.2 MiB      0.0 MiB           1             f"--- {time_2 - time_1} seconds ---")
56
57     88.2 MiB -409300.9 MiB      100001       for i in range(times_of_run):
58     88.2 MiB -409295.9 MiB      100000           if flats[i].floor is not None:
59                                                     pass
```

Рисунок 6 – Профилирование вызовов и памяти в программе и использованием weakref

Вывод: по замерам видно, что для использования памяти справедливо следующее неравенство: (память с использованием слотов) < (память с использованием обычных атрибутов) < (память с использованием weakref).

Профилирование вызовов при помощи cProfile

Профилирование вызовов в программе с помощью cProfile с использованием слотов смотри на рисунке 7.

```
(base) ➜  09 git:(main) ✗ python3 slots.py
time of creating 200000 objects --- 0.09624600410461426 seconds ---
time of access to 200000 objects --- 0.029459238052368164 seconds ---
time of modifying 200000 objects --- 0.04452681541442871 seconds ---
time of deleting 200000 objects --- 0.025628089904785156 seconds ---
         1000012 function calls in 0.401 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.283    0.283    0.401    0.401 /Users/artem/Documents/BMSTU/vk_2_sem/python/09/slots.py:31(run)
        1    0.077    0.077    0.096    0.096 /Users/artem/Documents/BMSTU/vk_2_sem/python/09/slots.py:48(<listcomp>)
   800000    0.022    0.000    0.022    0.000 {method 'append' of 'list' objects}
   200000    0.019    0.000    0.019    0.000 /Users/artem/Documents/BMSTU/vk_2_sem/python/09/slots.py:24(__init__)
        4    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        5    0.000    0.000    0.000    0.000 {built-in method time.time}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Рисунок 7 – Профилирование вызовов в программе с помощью cProfile с использованием слотов

Профилирование вызовов в программе с помощью cProfile с использованием обычных атрибутов смотри на рисунке 8.

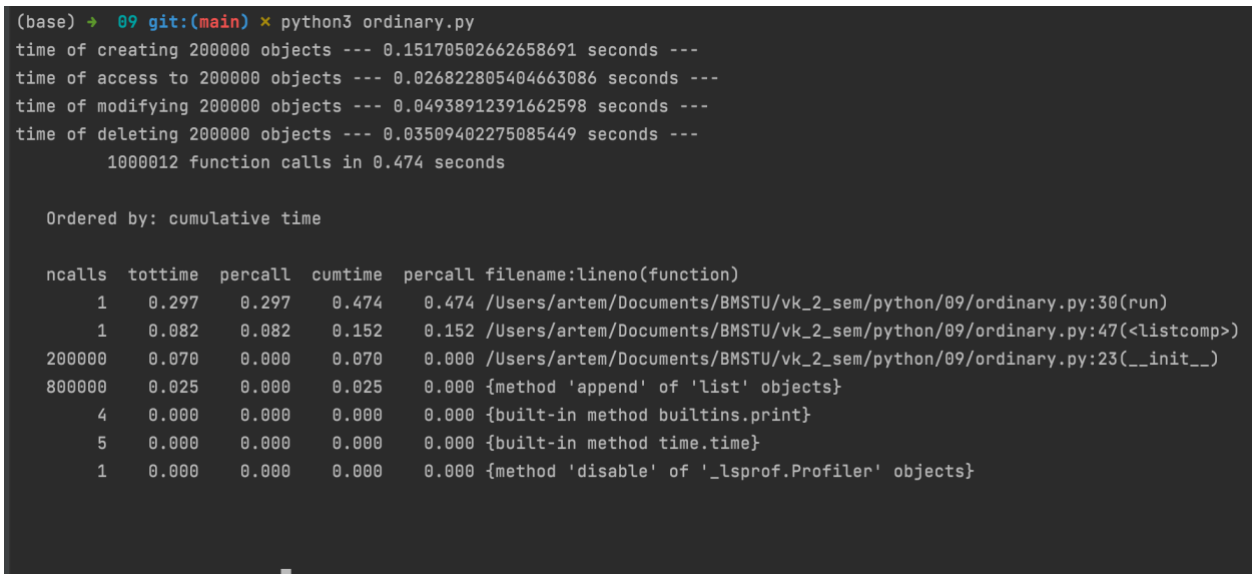

Рисунок 8 – Профилирование вызовов в программе с помощью cProfile с использованием обычных атрибутов

Профилирование вызовов в программе с помощью cProfile с использованием weakref смотри на рисунке 9.

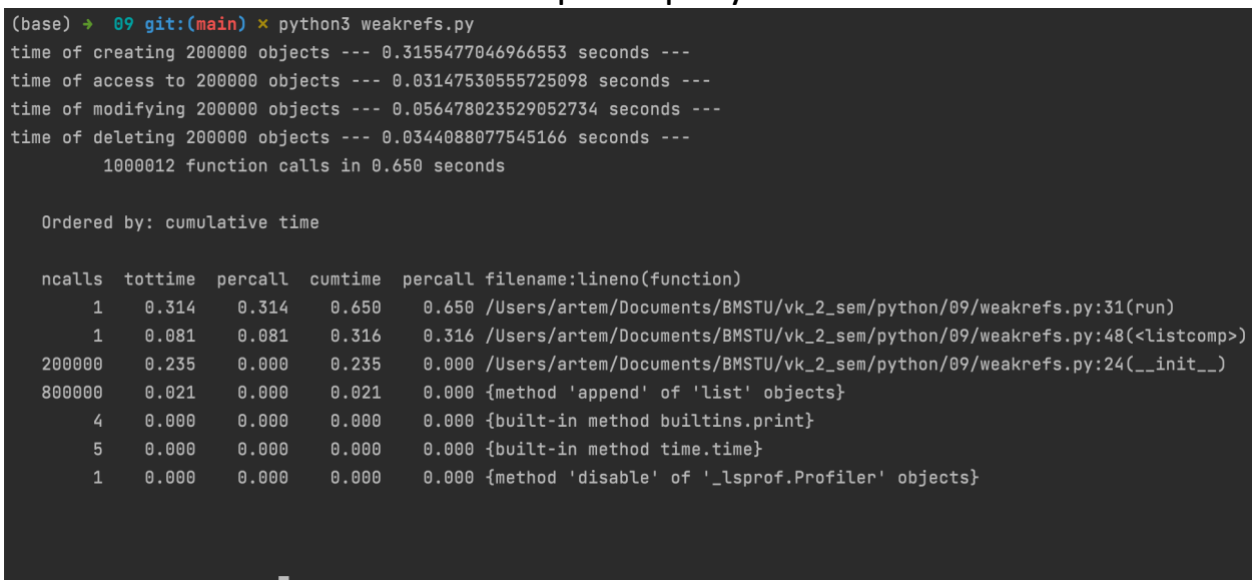

Рисунок 9 – Профилирование вызовов в программе с помощью cProfile с использованием weakref

Вывод: в данном случае следует смотреть на cumtime. По результатам замеров видно, что с использованием слотов создание и удаление объектов происходит значительно быстрее. С использованием weakref все операции сильно замедляются.