Лабораторная работа № 3 по курсу дискрeтного анализа: Исследование качества программ

Выполнил студент группы М8О-208Б-20 Фаттяхетдинов Сильвестр Динарович

# Условие

Для реализации словаря из предыдущей лабораторной работы,

необходимо провести исследование скорости выполнения и потребления

оперативной памяти. В случае выявления ошибок или явных недочётов,

требуется их исправить.

# gprof

## Основная информация

Утилита gprof позволяет измерить время работы всех функций, методов и операторов программы, количество их вызовов и долю от общего времени работы программы в процентах.

## Команды для работы с утилитой

Шаг 1) скомпилировать программу с флагом *−pg*:

g++ main . cpp *−*pg *−*o prog

Шаг 2) запустить программу, передав ей на ввод файл *test.txt*, в котором содержится по 5000 команд на вставку, поиск и удаление, вывод сохранить в out.txt:

prog.exe <t e s t . txt >out . txt

Кроме файла *out.txt*, в котором содержатся результаты выполнения команд из *test.txt*, также в каталоге появился файл *gmon.out*, в котором содержится вся информация, предоставленная утилитой gprof.

Шаг 3) получить всю информацию в текстовом виде

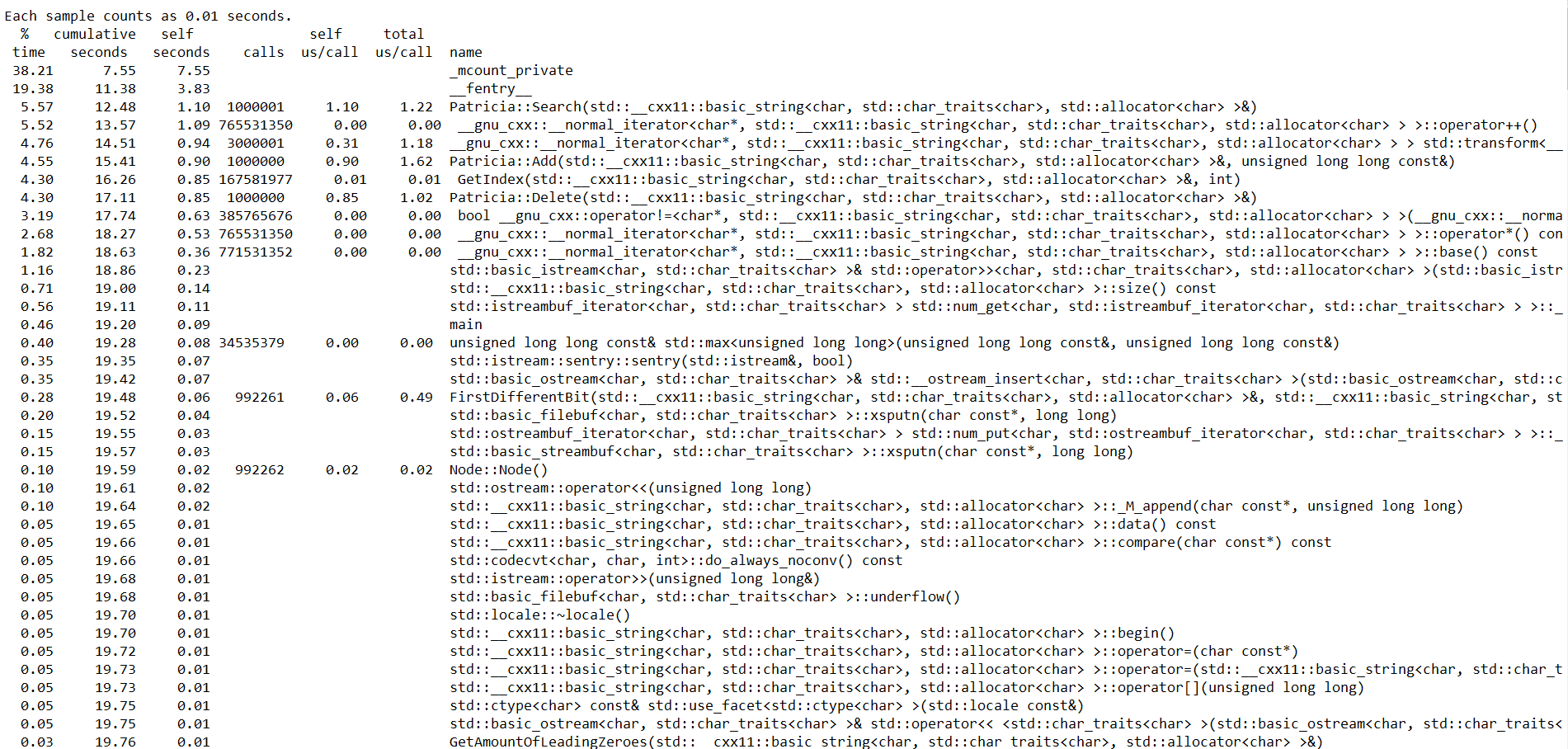
gprof prog.exe gmon . out > p r o f i l e *−*data . txt

Таким образом, получен текстовый файл с подробной информацией о времени работы и вызовах всех функций и операторов, которые использовались в программе.

## Результат работы утилиты

Ниже приведена таблица, в которую перенесены данные из файла *profile data.txt*, полученного с помощью утилиты gprof.

*−*



Все остальные функции по данным gprof работали 0% времени, поэтому на скриншоте они отсутствуют. \_mcount\_private, \_\_fentry\_\_ - в программе отсутствуют, запускаются при работе утилиты gprof, их можно опустить. Больше всего времени заняла функция поиска элемента, за ней идут инкремент и std::transform, также много времени задали Add, Delete и функция работы со строками GetIndex. Меньше времени заняли различные операторы работы со строками, а также со вводом-выводом.

# valgrind

Valgrind является самым распространённым и одним из самых удобных инструментов для отслеживания утечек памяти и других подобных ошибок. Для проверки программы на ошибки, связанные с памятью, необходимо ввести следующую команду в терминал:

v a l g r i n d ./ main <t e s t . txt >out . txt

В результате выполнения этой команды получаем следующее сообщение:

==13594== Memcheck, a memory error detector

==13594== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==13594== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info

==13594== Command: ./main

==13594==

==13594==

==13594== HEAP SUMMARY:

==13594== in use at exit: 122,880 bytes in 6 blocks

==13594== total heap usage: 10,900 allocs, 10,894 frees, 1,457,413 bytes allocated

==13594==

==13594== LEAK SUMMARY:

==13594== definitely lost: 0 bytes in 0 blocks

==13594== indirectly lost: 0 bytes in 0 blocks

==13594== possibly lost: 0 bytes in 0 blocks

==13594== still reachable: 122,880 bytes in 6 blocks

==13594== suppressed: 0 bytes in 0 blocks

==13594== Rerun with --leak-check=full to see details of leaked memory

==13594==

==13594== For lists of detected and suppressed errors, rerun with: -s

==13594== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Утечек обнаружено не было. Остались лишь still reachable из-за следующих строк:  
std::ios::sync\_with\_stdio(false);

std::cin.tie(nullptr);

std::cout.tie(nullptr);

Но, так как они ускоряют программу и всё же не вызывают утечек, они были оставлены в программе.

# Выводы

Выполнив данную лабораторную работу, я узнал о существовании такой утилиты как gprof и научился её использовать, также вспомнил valgrind и закрепил навыки работы с ним.