# Лабораторная работы № 3 по курсу дискретного анализа: Исследование качества программ

Выполнил студент группы 08-210 МАИ *Некрасов Константин*

## 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.

Результатом лабораторной работы является отчёт, состоящий из:

- Дневника выполнения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы.
- Выводов о найденных недочётах.
- Сравнение работы исправленной программы с предыдущей версией.
- Общих выводов о выполнении лабораторной работы, полученном опыте.

Минимальный набор используемых средств должен содержать утилиту ***gprof*** и библиотеку ***dmalloc***, однако их можно заменять на любые другие аналогичные или более развитые утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov).

## Метод решения

Valgrind – инструментальное ПО, которое используют для контроля использования памяти и обнаружения утечек памяти. С её помощью возможно обнаружить попытки использования или обращения к неинициализированной памяти, взаимодействие с памятью после её освобождения и утечки инициализированной памяти.

Листинг обращения к valgrind с данными самого первого теста системы:

```
root@RedTheCat:~/study/discran/lab2# valgrind --leak-check=full ./lab2.exe < test.txt | cat > out.txt

==59036== Memcheck, a memory error detector

==59036== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
```

```
==59036== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==59036== Command: ./lab2.exe
==59036==
==59036==
==59036== HEAP SUMMARY:
==59036==     in use at exit: 0 bytes in 0 blocks
==59036==   total heap usage: 8 allocs, 8 frees, 94,970 bytes allocated
==59036==
==59036== All heap blocks were freed -- no leaks are possible
==59036==
==59036== For lists of detected and suppressed errors, rerun with: -s
==59036== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind вывел информацию о том, что утечек памяти и ошибок не произошло. В процессе написания кода, я сам постоянно отслеживаю выделения и очищения памяти, поэтому итоговый код не имеет утечек. Поэтому введу их искусственно.

Листинг после удаления освобождения памяти в `Remove`:

```
root@RedTheCat:~/study/discran/lab2# valgrind --leak-check=full ./lab2.exe < test.txt | cat > out.txt
==61324== Memcheck, a memory error detector
==61324== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==61324== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==61324== Command: ./lab2.exe
==61324==
==61324==
```

```
==61324== Process terminating with default action of signal 27 (SIGPROF)
==61324==    at 0x4BC5A1A: __open_nocancel (open64_nocancel.c:39)
==61324==    by 0x4BD456F: write_gmon (gmon.c:370)
==61324==    by 0x4BD4DDE: _mcleanup (gmon.c:444)
==61324==    by 0x4AF1A55: __cxa_finalize (cxa_finalize.c:83)
==61324==    by 0x1093A6: ??? (in /root/study/discran/lab2/lab2.exe)
==61324==    by 0x400624D: _dl_fini (dl-fini.c:142)
==61324==    by 0x4AF1494: __run_exit_handlers (exit.c:113)
==61324==    by 0x4AF160F: exit (exit.c:143)
==61324==    by 0x4AD5D96: (below main) (libc_start_call_main.h:74)
==61324==
==61324== HEAP SUMMARY:
==61324==     in use at exit: 94,577 bytes in 6 blocks
==61324==   total heap usage: 8 allocs, 2 frees, 94,874 bytes allocated
==61324==
==61324== 297 (40 direct, 257 indirect) bytes in 1 blocks are definitely lost in loss record 2 of 6
==61324==     at 0x4849013: operator new(unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==61324==    by 0x109BDA: TTreap::Insert(char const*, unsigned long) (in /root/study/discran/lab2/lab2.exe)
==61324==    by 0x10952C: main (in /root/study/discran/lab2/lab2.exe)
==61324==
==61324== LEAK SUMMARY:
==61324==    definitely lost: 40 bytes in 1 blocks
```

```
==61324==    indirectly lost: 257 bytes in 1 blocks
==61324==      possibly lost: 0 bytes in 0 blocks
==61324==    still reachable: 94,280 bytes in 4 blocks
==61324==         suppressed: 0 bytes in 0 blocks
==61324== Reachable blocks (those to which a pointer was found) are not shown.
==61324== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==61324==
==61324== For lists of detected and suppressed errors, rerun with: -s
==61324== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Таким образом, valgrind с флагом -leak-check=full вывел для меня информацию об утечке памяти после выделения памяти в `Insert`, и соответственно я могу исправить ошибку в операции, противоположной `Insert` (то есть `Remove`).

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

Результат теста с миллионом строк 1000000, оформленный в виде таблицы:

| Процент времени | Всего секунд | Секунд на функцию | Число вызовов | Ср t на вызов, мс | Ср t на вызов потомков, мс | Название функции |
|---|---|---|---|---|---|---|
| 28.57 | 0.02 | 0.02 | 760000 | 26.32 | 26.32 | TTreap::Split(TTreap::Node*, char const*, TTreap::Node*&, TTreap::Node*&) |
| 28.57 | 0.04 | 0.02 | - | - | - | main |
| 14.29 | 0.05 | 0.01 | 760025 | 13.16 | 13.16 | TTreap::Merge(TTreap::Node*, TTreap::Node*) |
| 14.29 | 0.06 | 0.01 | 410000 | 24.39 | 48.78 | TTreap::Find(char const*) |
| 14.29 | 0.07 | 0.01 | 410000 | 24.39 | 24.39 | TTreap::Search(TTreap::Node*, char const*) |
| 0.00 | 0.07 | 0.00 | 1590001 | 0.00 | 0.00 | std::basic_istream<char, std::char_traits<char> > |
| 0.00 | 0.07 | 0.00 | 1000000 | 0.00 | 0.00 | ToLower(char*) |
| 0.00 | 0.07 | 0.00 | 420000 | 0.00 | 0.00 | TTreap::Min(TTreap::Node*) |
| 0.00 | 0.07 | 0.00 | 420000 | 0.00 | 44.17 | TTreap::Insert(char const*, unsigned long) |
| 0.00 | 0.07 | 0.00 | 170000 | 0.00 | 67.34 | TTreap::Remove(char*) |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.00 | 0.07 | 0.00 | 150025 | 0.00 | 0.00 | TTreap::Node::Node(char const*, unsigned long) |
| 0.00 | 0.07 | 0.00 | 150025 | 0.00 | 0.00 | TTreap::Node::~Node() |
| 0.00 | 0.07 | 0.00 | 1 | 0.00 | 0.00 | __static_initialization_and_destruction_0(int, int) |
| 0.00 | 0.07 | 0.00 | 1 | 0.00 | 0.00 | TTreap::Free(TTreap::Node*) |
| 0.00 | 0.07 | 0.00 | 1 | 0.00 | 0.00 | TTreap::TTreap() |
| 0.00 | 0.07 | 0.00 | 1 | 0.00 | 0.00 | TTreap::~TTreap() |

Из таблицы видно, что чаще всего вызываются функции `Split` и `Merge`, и это логично, поскольку на них основываются функции вставки и удаления. Для ускорения работы нужно минимизировать количество вызовов функции `Split`, поскольку она имеет наибольшее время одного вызова.

## Выводы

При выполнении лабораторной работы я ознакомился с профилированием, крайне важным для качественно разработки программ, узнал возможные инструменты и использовал их на практике. Я знал о valgrind и возможности отслеживания утечек памяти и сейчас использовал на практике, а также узнал о gprof с очень наглядным показанием использования функций, однако для полной наглядности необходимы большие тесты. Тем не менее, полезно знать, какая функция используется чаще всего и как много времени требует в среднем.