Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: «Информационных технологий и прикладной математики»
Кафедра: 806 «Вычислительная математика и программирование»

# Лабораторная работа № 3

## по курсу «Дискретный анализ»

| | |
|---|---|
| Студент: | Королев И.М. |
| Группа: | М8О-208Б-19 |
| Преподаватель: | Капралов Н.С. |
| Дата: | |
| Оценка: | |

Москва 2020

# 1. Постановка задачи

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 недочётов.
- Сравнение работы исправленной программы с предыдущей версии.
- Общих выводов о выполнении лабораторной работы, полученном опыте.
- Минимальный набор используемых средст должен содержать утилиту gprof и библиотеку dmalloc, однако их можно заменять на любые другие аналогичные или более известные утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov).

**Вариант структуры данных:** Красно-чёрное дерево

# 2. Описание

Для выполнения лабораторной работы были использованы утилиты valgrind и gprof. Valgrind – утилита для поиска ошибок работы с памятью. С помощью неё можно находить утечки памяти. Также vallgrind можно использовать для профилирования вместе с callgrind. Для профилирования программы была использована утилита gprof. Она позволяет узнать расходуемое программой время в разных её областях, какие функции вызывали другие функции, пока программа исполнялась. Утилита gprof помогает узнать какая часть программы исполняется медленнее, чем ожидалось, после эту часть можно переписать для увеличения эффективности программы. Профилятор использует информацию, собранную в процессе реального времени исполнения программы. Он может быть использован для исследования программ, которые слишком большие или сложные для анализа.

# 3. Протокол

## Valgrind

```
harry@LAPTOP-Q9RGM4HT:~/MAI/DA/da_lab2$ valgrind --tool=memcheck --leak-
check=full ./solution < tests/big_test.txt > result
==1059== Memcheck, a memory error detector
==1059== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1059== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1059== Command: ./solution
==1059==
==1059== error calling PR_SET_PTRACER, vgdb might block
==1059==
==1059== HEAP SUMMARY:
==1059==     in use at exit: 8,845 bytes in 58 blocks
==1059==   total heap usage: 67,716 allocs, 67,658 frees, 10,428,209 bytes
allocated
==1059==
==1059== 8,845 (96 direct, 8,749 indirect) bytes in 2 blocks are definitely lost
in loss record 3 of 3
==1059==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==1059==    by 0x10B048: Menu() (in /home/harry/MAI/DA/da_lab2/solution)
==1059==    by 0x10A7A2: main (in /home/harry/MAI/DA/da_lab2/solution)
==1059==
==1059== LEAK SUMMARY:
==1059==    definitely lost: 96 bytes in 2 blocks
==1059==    indirectly lost: 8,749 bytes in 56 blocks
==1059==      possibly lost: 0 bytes in 0 blocks
==1059==    still reachable: 0 bytes in 0 blocks
==1059==         suppressed: 0 bytes in 0 blocks
==1059==
==1059== For lists of detected and suppressed errors, rerun with: -s
==1059== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Во время диагностики было выявлено, что программа выполняет действия с удалённой памятью, что не должно происходить. Оказалось, что при удалении узла из древа узел перекрашивался в красный, после чего с ним выполнялись определённые действия. Это было ошибкой, так как изначально он должен был быть перекрашен в чёрный цвет. Неверное написание кода дерева и повлекло за собой утечки памяти.

```
root@Harry:~/work/MAI/DA/da_lab2# make clean
rm -rf *.o solution
root@Harry:~/work/MAI/DA/da_lab2# make
g++ --std=c++17 -Wall -c -O3 -std=c++17 -pg main.cpp
g++ --std=c++17 -Wall -c -O3 -std=c++17 -pg functions.cpp
g++ --std=c++17 -Wall -c -O3 -std=c++17 -pg menu.cpp
g++ -std=c++17 -pg main.o functions.o menu.o -o solution
root@Harry:~/work/MAI/DA/da_lab2# valgrind ./solution < tests/big_test.txt >
result
==9286== Memcheck, a memory error detector
==9286== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9286== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==9286== Command: ./solution
```

```
==9286==
==9286== error calling PR_SET_PTRACER, vgdb might block
==9286==
==9286== HEAP SUMMARY:
==9286==     in use at exit: 0 bytes in 0 blocks
==9286==   total heap usage: 67,716 allocs, 67,716 frees, 10,435,377 bytes
allocated
==9286==
==9286== All heap blocks were freed -- no leaks are possible
==9286==
==9286== For lists of detected and suppressed errors, rerun with: -s
==9286== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Недочёт был исправлен, после чего программа работала без ошибок и выполняла нужное количество выделений и освобождений памяти.

# Gprof

```
harry@harry-VirtualBox:~/DA/lab3/da_lab2$ g++ -pg lab3.cpp
harry@harry-VirtualBox:~/DA/lab3/da_lab2$ ./a.out < tests/test_55 > result
harry@harry-VirtualBox:~/DA/lab3/da_lab2$ ./a.out -p ./a.out
harry@harry-VirtualBox:~/DA/lab3/da_lab2$ ./a.out < tests/test_55 > result
harry@harry-VirtualBox:~/DA/lab3/da_lab2$ gprof ./a.out -p ./gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 24.89      6.42     6.42  4604241     0.00     0.00  Node<char*, unsigned long
long>::GetKey(char*)
 22.82     12.30     5.88  4816460     0.00     0.00  Str_copy(char const*,
char*)
  9.13     14.65     2.35 79276296     0.00     0.00  Equal_strings(char const*,
char const*)
  8.47     16.84     2.18 163106217     0.00     0.00  Node<char*, unsigned long
long>::FindKey()
  6.31     18.46     1.63 140793522     0.00     0.00  Node<char*, unsigned long
long>::FindParent()
  5.79     19.95     1.49  4604241     0.00     0.00  Tree<char*, unsigned long
long>::Insert(Node<char*, unsigned long long>*)
  5.59     21.40     1.44 59502341     0.00     0.00  Node<char*, unsigned long
long>::FindLeft()
  4.43     22.54     1.14 93013268     0.00     0.00  Node<char*, unsigned long
long>::FindRight()
  1.44     22.91     0.37  4600917     0.00     0.00  Tree<char*, unsigned long
long>::Fixup(Node<char*, unsigned long long>*)
  1.26     23.23     0.33 20556339     0.00     0.00  Node<char*, unsigned long
long>::GetRight(Node<char*, unsigned long long>*)
  1.22     23.55     0.32 31867395     0.00     0.00  Node<char*, unsigned long
long>::GetColor(NodeColor)
  1.17     23.85     0.30     3124     0.00     0.00  Tree<char*, unsigned long
long>::AllTreeDelete()
  1.05     24.12     0.27     3239     0.00     0.00  void Tree_save<char*,
unsigned long long>(std::ostream&, Tree<char*, unsigned long long>*, Node<char*,
unsigned long long>*)
  0.97     24.37     0.25 22741501     0.00     0.00  Node<char*, unsigned long
long>::FindColor()
```

```
  0.85     24.59     0.22  4540256     0.00     0.00  Tree<char*, unsigned long
long>::LeftRotation(Node<char*, unsigned long long>*)
  0.76     24.78     0.20 15936558     0.00     0.00  Node<char*, unsigned long
long>::GetParent(Node<char*, unsigned long long>*)
  0.70     24.97     0.18                             Strings_compare(char
const*, char const*)
  0.68     25.14     0.18     3123     0.00     0.01  void Load_tree<char*,
unsigned long long>(std::basic_ifstream<char, std::char_traits<char> >*,
Tree<char*, unsigned long long>*)
  0.56     25.29     0.15 11445672     0.00     0.00  Node<char*, unsigned long
long>::GetLeft(Node<char*, unsigned long long>*)
  0.52     25.42     0.14      122     0.00     0.00  Tree<char*, unsigned long
long>::TreeMinimum(Node<char*, unsigned long long>*)
  0.39     25.52     0.10  4607365     0.00     0.00  Node<char*, unsigned long
long>::~Node()
  0.35     25.61     0.09  4607365     0.00     0.00  Node<char*, unsigned long
long>::Node()
  0.23     25.67     0.06  9617699     0.00     0.00  Tree<char*, unsigned long
long>::FindTNull()
  0.19     25.72     0.05  4807356     0.00     0.00  Node<char*, unsigned long
long>::FindValue()
  0.16     25.76     0.04     3124     0.00     0.00  Tree<char*, unsigned long
long>::Tree()
  0.10     25.79     0.03  4604241     0.00     0.00  Node<char*, unsigned long
long>::GetValue(unsigned long long)
  0.06     25.80     0.02        1     0.02    25.63  Menu()
  0.04     25.81     0.01     6478     0.00     0.00  Tree<char*, unsigned long
long>::FindRoot()
  0.00     25.81     0.00    12366     0.00     0.00  Tree<char*, unsigned long
long>::Search(char*)
  0.00     25.81     0.00     3239     0.00     0.00
std::operator|(std::_Ios_Iostate, std::_Ios_Iostate)
  0.00     25.81     0.00     3124     0.00     0.00  Tree<char*, unsigned long
long>::~Tree()
  0.00     25.81     0.00      282     0.00     0.00  Tree<char*, unsigned long
long>::Transplant(Node<char*, unsigned long long>*, Node<char*, unsigned long
long>*)
  0.00     25.81     0.00      223     0.00     0.00  Tree<char*, unsigned long
long>::Delete(Node<char*, unsigned long long>*)
  0.00     25.81     0.00      221     0.00     0.00  Tree<char*, unsigned long
long>::DeleteFixup(Node<char*, unsigned long long>*)
  0.00     25.81     0.00       43     0.00     0.00  Tree<char*, unsigned long
long>::RightRotation(Node<char*, unsigned long long>*)
  0.00     25.81     0.00        1     0.00     0.00
_GLOBAL__sub_I__Z15Strings_comparePKcS0_
  0.00     25.81     0.00        1     0.00     0.00
__static_initialization_and_destruction_0(int, int)
```

Из работы утилиты gprof видно, что программа тратит большую часть своего времени на копирование ключа в виде строки в значение узла, просто копирование строк, а также сравнение строк. Выполнение самих функций дерева происходит за значительно меньшее время.

# 4. Дневник отладки

1. 04.12.2020 Использовал утилиту Valgrind для выявления утечек памяти при работе программы.
2. 10.12.2020 Ознакомился с утилитой Gprof, а также с Callgrind.
3. 15.12.2020 Выполнил профилирование программы с помощью утилиты Gprof.

# 5. Выводы

Выполняя лабораторную работу были получены навыки работы с утилитой Valgrind, которая позволяет отслеживать работу программы с памятью. Благодаря этой утилите можно отлавливать ошибки при выделении и освобождении памяти. Из-за того, что я ошибся при написании функции удаления узла из дерева в цвете узла, происходили утечки памяти и неправильная работа программы. С помощью этой утилиты я смог решить эту проблему. Также были получены навыки работы с утилитой Gprof, которая позволяет узнавать количество вызовов отдельных функций программы, время их работы за всё время работы программы. С помощью этой информации можно узнать какая часть кода программы работает не так эффективно, как хотелось бы. Поэтому, если возможно увеличить эффективность работы программы, то эту часть программы можно переписать. В итоге эти утилиты помогают оптимизировать код программы и время её работы.

# Список используемых источников

1. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* – Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. – 1296 с. (ISBN 5-8459-0857-4 (рус.))
2. Роберт Лафоре. *Объектно-ориентированное программирование в C++*. Классика Computer Science, 4-е издание. — Издательский дом «Питер», 2018. Перевод с английского: А. Кузнецов, М. Назаров, В.Шрага. — 928 с. (ISBN 978-5-596-00353-7 (рус.))
3. *Valgrind – Википедия.*
URL: https://ru.wikipedia.org/wiki/Valgrind (дата обращения: 16.11.2020).
3. *Профилирование – Википедия.*
URL: https://ru.wikipedia.org/wiki/Профилирование_(информатика) (дата обращения: 16.11.2020).
4. *Информация о работе с valgrind* – [Электронный ресурс]. – URL: http://alexott.net/ru/linux/valgrind/Valgrind.html (дата обращения: 16.11.2020).
5. *Информация о работе с gprof* – [Электронный ресурс]. – URL: https://www.opennet.ru/docs/RUS/gprof/ (дата обращения: 16.11.2020).