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

Выполнил студент группы 08-208 МАИ *Ибрагимов Далгат*.

Условие

Общая 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:

- 1. Дневника выполонения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы.
- 2. Выводов о найденных недочётах.
- 3. 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

Для исследования потребления памяти я использовала утилиту Valgrind. Это инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, проверки потокобезопасности, а также профилирования. Наиболее используемым инструментом в этой утилите является Memcheck. Проблемы, которые может обнаружить Memcheck, включают в себя:

- 1. Попытки использования неинициализированной памяти
- 2. Чтение/запись в память после её освобождения
- 3. Чтение/запись за границами выделенного блока
- 4. Утечки памяти

Для отображения профильной статистики, которая накапливается во время приложения я использовала утилиту gprof. Профиллирование позволяет понять, где программа расходует свое время и какие функции вызывали другие функции, пока программа исполнялась. Эта информация может указать на ту часть программы, которая исполняется медленнее, чем ожидалось.

Дневник отладки

Удалим деструтор для Treap, чтобы создать утечку памяти. Для работы с утилитой Valgrind и gprof скомпилируем программу с флагами -g и -pg, а затем пропишем в консоли команды для их использования:

```
    lockr@lockR:~/projects/DA_LABS/lab3$ cat test.txt
+ a 1
+ A 2
+
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
    18446744073709551615
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

A
- A
a
! Save tree
+ dad 121212
dad
! Load tree
dad
+ dad 555
- dad
- DaD
+ a 1
+ A 2

lockr@lockR:~/projects/DA_LABS/lab3$ valgrind ./a.out < test.txt
==13232== Memcheck, a memory error detector
==13232== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward
    et al.
==13232== Using Valgrind-3.18.1 and LibVEX; rerun with -h for
   copyright info
==13232== Command: ./a.out
==13232==
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
OK
```

```
OK
OK: 121212
OK
NoSuchWord
OK
OK
NoSuchWord
OK
Exist
==13232==
==13232== HEAP SUMMARY:
==13232==     in use at exit: 339 bytes in 4 blocks
==13232==   total heap usage: 21 allocs, 17 frees, 123,583 bytes
   allocated
==13232==
==13232== LEAK SUMMARY:
==13232==    definitely lost: 40 bytes in 1 blocks
==13232==    indirectly lost: 299 bytes in 3 blocks
==13232==      possibly lost: 0 bytes in 0 blocks
==13232==    still reachable: 0 bytes in 0 blocks
==13232==         suppressed: 0 bytes in 0 blocks
==13232== Rerun with --leak-check=full to see details of leaked
   memory
==13232==
==13232== For lists of detected and suppressed errors, rerun with:
    -s
==13232== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0
   from 0)
lockr@lockR:~/projects/DA_LABS/lab3$ gprof
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00       49     0.00     0.00  toLowerCase(
     char*)
  0.00      0.00     0.00       15     0.00     0.00  Treap::find(
     Node*, char const*)
  0.00      0.00     0.00        7     0.00     0.00  Treap::add(
     char const*, unsigned long)
  0.00      0.00     0.00        6     0.00     0.00  Node::Node(
     char const*, unsigned long, unsigned long)
  0.00      0.00     0.00        5     0.00     0.00  Treap::
     insert(Node*, Node*)
  0.00      0.00     0.00        5     0.00     0.00  Treap::
     search(char const*)
  0.00      0.00     0.00        4     0.00     0.00  Node::~Node
     ()
  0.00      0.00     0.00        3     0.00     0.00  Treap::del(
     char const*)
  0.00      0.00     0.00        3     0.00     0.00  Treap::split
     (Node*, char const*, Node*&, Node*&)
  0.00      0.00     0.00        2     0.00     0.00  Treap::erase
     (Node*, char const*)
  0.00      0.00     0.00        2     0.00     0.00  Treap::merge
     (Node*, Node*)
  0.00      0.00     0.00        1     0.00     0.00
     __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  Treap::
     saveToFile(Node*, std::basic_ofstream<char, std::char_traits<
     char> >&)
  0.00      0.00     0.00        1     0.00     0.00  Treap::
     loadFromFile(std::basic_ifstream<char, std::char_traits<char>
      >&)
  0.00      0.00     0.00        1     0.00     0.00  Treap::load(
     char const*)
  0.00      0.00     0.00        1     0.00     0.00  Treap::save(
     char const*)
  0.00      0.00     0.00        1     0.00     0.00  Treap::clear
     (Node*)
  0.00      0.00     0.00        1     0.00     0.00  Treap::Treap
     ()

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.
```

```
calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       2/49          Treap::del(char
                   const*) [15]
                0.00    0.00       4/49          Treap::erase(Node
                   *, char const*) [17]
                0.00    0.00       6/49          Treap::split(Node
                   *, char const*, Node*&, Node*&) [16]
                0.00    0.00       7/49          Treap::add(char
                   const*, unsigned long) [10]
                0.00    0.00      30/49          Treap::find(Node
                   *, char const*) [9]
[8]     0.0     0.00    0.00      49         toLowerCase(char*)
   [8]
-----------------------------------------------
                                   8             Treap::find(Node
                                     *, char const*) [9]
                0.00    0.00       3/15          Treap::del(char
                   const*) [15]
                0.00    0.00       5/15          Treap::search(
                   char const*) [13]
                0.00    0.00       7/15          Treap::add(char
                   const*, unsigned long) [10]
[9]     0.0     0.00    0.00      15+8       Treap::find(Node*,
   char const*) [9]
                0.00    0.00      30/49          toLowerCase(char
                   *) [8]
                                   8             Treap::find(Node
                                     *, char const*) [9]
-----------------------------------------------
                0.00    0.00       7/7           main [6]
[10]    0.0     0.00    0.00       7         Treap::add(char const
   *, unsigned long) [10]
                0.00    0.00       7/49          toLowerCase(char
                   *) [8]
                0.00    0.00       7/15          Treap::find(Node
                   *, char const*) [9]
                0.00    0.00       5/6           Node::Node(char
                   const*, unsigned long, unsigned long) [11]
                0.00    0.00       5/5           Treap::insert(
                   Node*, Node*) [12]
-----------------------------------------------
                0.00    0.00       1/6           Treap::
                   loadFromFile(std::basic_ifstream<char, std::
                   char_traits<char> >&) [21]
                0.00    0.00       5/6           Treap::add(char
                   const*, unsigned long) [10]
[11]    0.0     0.00    0.00       6         Node::Node(char const
   *, unsigned long, unsigned long) [11]
-----------------------------------------------
                                   1             Treap::insert(
                                     Node*, Node*) [12]
                0.00    0.00       5/5           Treap::add(char
                   const*, unsigned long) [10]
[12]     0.0    0.00    0.00       5+1       Treap::insert(Node*,
   Node*) [12]
                0.00    0.00       3/3           Treap::split(Node
                   *, char const*, Node*&, Node*&) [16]
                                   1             Treap::insert(
                                      Node*, Node*) [12]
-----------------------------------------------
                0.00    0.00       5/5           main [6]
[13]     0.0    0.00    0.00       5         Treap::search(char
   const*) [13]
                0.00    0.00       5/15          Treap::find(Node
                   *, char const*) [9]
-----------------------------------------------
                0.00    0.00       2/4           Treap::clear(Node
                   *) [24]
                0.00    0.00       2/4           Treap::erase(Node
                   *, char const*) [17]
[14]     0.0    0.00    0.00       4         Node::~Node() [14]
-----------------------------------------------
                0.00    0.00       3/3           main [6]
[15]     0.0    0.00    0.00       3         Treap::del(char const
   *) [15]
                0.00    0.00       3/15          Treap::find(Node
                   *, char const*) [9]
                0.00    0.00       2/49          toLowerCase(char
                   *) [8]
                0.00    0.00       2/2           Treap::erase(Node
                   *, char const*) [17]
-----------------------------------------------
                                   3             Treap::split(Node
                                      *, char const*, Node*&, Node
                                      *&) [16]
                0.00    0.00       3/3           Treap::insert(
                   Node*, Node*) [12]
[16]     0.0    0.00    0.00       3+3       Treap::split(Node*,
   char const*, Node*&, Node*&) [16]
                0.00    0.00       6/49          toLowerCase(char
                   *) [8]
                                   3             Treap::split(Node
                                      *, char const*, Node*&, Node
```

```
                                            *&) [16]
-----------------------------------------------
                0.00    0.00       2/2           Treap::del(char
                   const*) [15]
[17]     0.0    0.00    0.00       2         Treap::erase(Node*,
   char const*) [17]
                0.00    0.00       4/49          toLowerCase(char
                   *) [8]
                0.00    0.00       2/2           Treap::merge(Node
                   *, Node*) [18]
                0.00    0.00       2/4           Node::~Node()
                   [14]
-----------------------------------------------
                0.00    0.00       2/2           Treap::erase(Node
                   *, char const*) [17]
[18]     0.0    0.00    0.00       2         Treap::merge(Node*,
   Node*) [18]
-----------------------------------------------
                0.00    0.00       1/1
                   _GLOBAL__sub_I__Z11toLowerCasePc [26]
[19]     0.0    0.00    0.00       1
   __static_initialization_and_destruction_0(int, int) [19]
-----------------------------------------------
                0.00    0.00       1/1           Treap::save(char
                   const*) [23]
[20]     0.0    0.00    0.00       1         Treap::saveToFile(
   Node*, std::basic_ofstream<char, std::char_traits<char> >&)
   [20]
-----------------------------------------------
                0.00    0.00       1/1           Treap::load(char
                   const*) [22]
[21]     0.0    0.00    0.00       1         Treap::loadFromFile(
   std::basic_ifstream<char, std::char_traits<char> >&) [21]
                0.00    0.00       1/6           Node::Node(char
                   const*, unsigned long, unsigned long) [11]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[22]     0.0    0.00    0.00       1         Treap::load(char
   const*) [22]
                0.00    0.00       1/1           Treap::
                   loadFromFile(std::basic_ifstream<char, std::
                   char_traits<char> >&) [21]
                0.00    0.00       1/1           Treap::clear(Node
                   *) [24]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[23]     0.0    0.00    0.00       1         Treap::save(char
   const*) [23]
                0.00    0.00       1/1           Treap::saveToFile
                   (Node*, std::basic_ofstream<char, std::
                   char_traits<char> >&) [20]
-----------------------------------------------
                                   4             Treap::clear(Node
                                     *) [24]
                0.00    0.00       1/1           Treap::load(char
                   const*) [22]
[24]     0.0    0.00    0.00       1+4       Treap::clear(Node*)
   [24]
                0.00    0.00       2/4           Node::~Node()
                   [14]
                                   4             Treap::clear(Node
                                     *) [24]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[25]     0.0    0.00    0.00       1         Treap::Treap() [25]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted
     by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line
    with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table
          .
                Index numbers are sorted numerically.
                The index number is printed next to every function
                     name so
                it is easier to look up where the function is in
                   the table.
     % time     This is the percentage of the `total' time that was spent
                in this function and its children.  Note that due to
                different viewpoints, functions excluded by options, etc,
                these numbers will NOT add up to 100%.

     self       This is the total amount of time spent in this function.

     children   This is the total amount of time propagated into this
                function by its children.

     called     This is the number of times the function was called.
                If the function called itself recursively, the number
                only includes non-recursive calls, and is followed by
                a `+' and the number of recursive calls.

     name       The name of the current function.  The index number is
                printed after it.  If the function is a member of a
                cycle, the cycle number is printed between the
                function's name and the index number.


 For the function's parents, the fields have the following meanings:

     self       This is the amount of time that was propagated directly
                from the function into this parent.

     children   This is the amount of time that was propagated from
                the function's children into this parent.
```

called — This is the number of times this parent called the function '/' the total number of times the function was called. Recursive calls to the function are not included in the number after the '/'.

name — This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word '<spontaneous>' is printed in the 'name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self — This is the amount of time that was propagated directly from the child into the function.

children — This is the amount of time that was propagated from the child's children to the function.

called — This is the number of times the function called this child '/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the '/'.

name — This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the

```
 cycle (as parents) and the members of the cycle (as children.)
 The ‘+’ recursive calls entry shows the number of function calls
    that
 were internal to the cycle, and the calls entry for each member
    shows,
 for that member, how many times it was called from other members
    of
 the cycle.


Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without
   modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


Index by function name

   [8] toLowerCase(char*)       [10] Treap::add(char const*,
      unsigned long) [17] Treap::erase(Node*, char const*)
  [19] __static_initialization_and_destruction_0(int, int) (0.cpp)
      [15] Treap::del(char const*) [18] Treap::merge(Node*, Node*)
  [11] Node::Node(char const*, unsigned long, unsigned long) [9]
     Treap::find(Node*, char const*) [16] Treap::split(Node*, char
      const*, Node*&, Node*&)
  [14] Node::~Node()            [22] Treap::load(char const*) [12]
     Treap::insert(Node*, Node*)
  [20] Treap::saveToFile(Node*, std::basic_ofstream<char, std::
     char_traits<char> >&) [23] Treap::save(char const*) [13]
     Treap::search(char const*)
  [21] Treap::loadFromFile(std::basic_ifstream<char, std::
     char_traits<char> >&) [24] Treap::clear(Node*) [25] Treap::
     Treap()
```

Видим потери памяти и где они происходят. Теперь вернем деструтор для Treap:

```
lockr@lockR:~/projects/DA_LABS/lab3$ valgrind ./a.out < test.txt
==12838== Memcheck, a memory error detector
==12838== Copyright (C) 2002−2017, and GNU GPL'd, by Julian Seward et al.
==12838== Using Valgrind−3.18.1 and LibVEX; rerun with −h for copyright info
==12838== Command: ./a.out
==12838==
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
OK
OK
OK: 121212
OK
NoSuchWord
OK
OK
NoSuchWord
OK
Exist
==12838==
==12838== HEAP SUMMARY:
==12838==     in use at exit: 0 bytes in 0 blocks
==12838==   total heap usage: 21 allocs, 21 frees, 123,751 bytes allocated
==12838==
==12838== All heap blocks were freed −− no leaks are possible
==12838==
==12838== For lists of detected and suppressed errors, rerun with: −s
==12838== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
lockr@lockR:~/projects/DA_LABS/lab3$ gprof
Flat profile:

Each sample counts as 0.01 seconds.
```

```
no time accumulated

 %   cumulative   self              self     total
time   seconds   seconds    calls  Ts/call  Ts/call  name
 0.00      0.00     0.00       49     0.00     0.00  toLowerCase(
    char*)
 0.00      0.00     0.00       15     0.00     0.00  Treap::find(
    Node*, char const*)
 0.00      0.00     0.00        7     0.00     0.00  Treap::add(
    char const*, unsigned long)
 0.00      0.00     0.00        6     0.00     0.00  Node::Node(
    char const*, unsigned long, unsigned long)
 0.00      0.00     0.00        6     0.00     0.00  Node::~Node
    ()
 0.00      0.00     0.00        5     0.00     0.00  Treap::
    insert(Node*, Node*)
 0.00      0.00     0.00        5     0.00     0.00  Treap::
    search(char const*)
 0.00      0.00     0.00        3     0.00     0.00  Treap::del(
    char const*)
 0.00      0.00     0.00        3     0.00     0.00  Treap::split
    (Node*, char const*, Node*&, Node*&)
 0.00      0.00     0.00        2     0.00     0.00  Treap::clear
    (Node*)
 0.00      0.00     0.00        2     0.00     0.00  Treap::erase
    (Node*, char const*)
 0.00      0.00     0.00        2     0.00     0.00  Treap::merge
    (Node*, Node*)
 0.00      0.00     0.00        1     0.00     0.00
    __static_initialization_and_destruction_0(int, int)
 0.00      0.00     0.00        1     0.00     0.00  Treap::
    saveToFile(Node*, std::basic_ofstream<char, std::char_traits<
    char> >&)
 0.00      0.00     0.00        1     0.00     0.00  Treap::
    loadFromFile(std::basic_ifstream<char, std::char_traits<char>
     >&)
 0.00      0.00     0.00        1     0.00     0.00  Treap::load(
    char const*)
 0.00      0.00     0.00        1     0.00     0.00  Treap::save(
    char const*)
 0.00      0.00     0.00        1     0.00     0.00  Treap::Treap
    ()
```

```
  0.00      0.00     0.00        1     0.00     0.00  Treap::~
    Treap()

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without
   modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


                     Call graph (explanation follows)
```

```
granularity: each sample hit covers 4 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       2/49          Treap::del(char
   const*) [15]
                0.00    0.00       4/49          Treap::erase(Node
   *, char const*) [18]
                0.00    0.00       6/49          Treap::split(Node
   *, char const*, Node*&, Node*&) [16]
                0.00    0.00       7/49          Treap::add(char
   const*, unsigned long) [10]
                0.00    0.00      30/49          Treap::find(Node
   *, char const*) [9]
[8]      0.0    0.00    0.00      49         toLowerCase(char*)
   [8]
-----------------------------------------------
                                   8             Treap::find(Node
                                       *, char const*) [9]
                0.00    0.00       3/15          Treap::del(char
   const*) [15]
                0.00    0.00       5/15          Treap::search(
   char const*) [14]
                0.00    0.00       7/15          Treap::add(char
   const*, unsigned long) [10]
[9]      0.0    0.00    0.00      15+8       Treap::find(Node*,
   char const*) [9]
                0.00    0.00      30/49          toLowerCase(char
   *) [8]
                                   8             Treap::find(Node
                                       *, char const*) [9]
-----------------------------------------------
                0.00    0.00       7/7           main [6]
[10]     0.0    0.00    0.00       7         Treap::add(char const
   *, unsigned long) [10]
                0.00    0.00       7/49          toLowerCase(char
   *) [8]
                0.00    0.00       7/15          Treap::find(Node
   *, char const*) [9]
                0.00    0.00       5/6           Node::Node(char
   const*, unsigned long, unsigned long) [11]
                0.00    0.00       5/5           Treap::insert(
   Node*, Node*) [13]
```

```
-----------------------------------------------
                0.00    0.00     1/6          Treap::
                   loadFromFile(std::basic_ifstream<char, std::
                   char_traits<char> >&) [22]
                0.00    0.00     5/6          Treap::add(char
                   const*, unsigned long) [10]
[11]    0.0     0.00    0.00     6        Node::Node(char const
   *, unsigned long, unsigned long) [11]
-----------------------------------------------
                0.00    0.00     2/6          Treap::erase(Node
                   *, char const*) [18]
                0.00    0.00     4/6          Treap::clear(Node
                   *) [17]
[12]    0.0     0.00    0.00     6        Node::~Node() [12]
-----------------------------------------------
                                 1            Treap::insert(
                                    Node*, Node*) [13]
                0.00    0.00     5/5          Treap::add(char
                   const*, unsigned long) [10]
[13]    0.0     0.00    0.00     5+1      Treap::insert(Node*,
   Node*) [13]
                0.00    0.00     3/3          Treap::split(Node
                   *, char const*, Node*&, Node*&) [16]
                                 1            Treap::insert(
                                    Node*, Node*) [13]
-----------------------------------------------
                0.00    0.00     5/5          main [6]
[14]    0.0     0.00    0.00     5        Treap::search(char
   const*) [14]
                0.00    0.00     5/15         Treap::find(Node
                   *, char const*) [9]
-----------------------------------------------
                0.00    0.00     3/3          main [6]
[15]    0.0     0.00    0.00     3        Treap::del(char const
   *) [15]
                0.00    0.00     3/15         Treap::find(Node
                   *, char const*) [9]
                0.00    0.00     2/49         toLowerCase(char
                   *) [8]
                0.00    0.00     2/2          Treap::erase(Node
                   *, char const*) [18]
-----------------------------------------------
```

```
                                        3               Treap::split(Node
                                          *, char const*, Node*&, Node
                                          *&) [16]
                        0.00    0.00    3/3             Treap::insert(
                          Node*, Node*) [13]
[16]    0.0     0.00    0.00    3+3             Treap::split(Node*,
   char const*, Node*&, Node*&) [16]
                        0.00    0.00    6/49            toLowerCase(char
                          *) [8]
                                        3               Treap::split(Node
                                          *, char const*, Node*&, Node
                                          *&) [16]
-----------------------------------------------
                                        8               Treap::clear(Node
                                          *) [17]
                        0.00    0.00    1/2             Treap::~Treap()
                          [26]
                        0.00    0.00    1/2             Treap::load(char
                          const*) [23]
[17]    0.0     0.00    0.00    2+8             Treap::clear(Node*)
   [17]
                        0.00    0.00    4/6             Node::~Node()
                          [12]
                                        8               Treap::clear(Node
                                          *) [17]
-----------------------------------------------
                        0.00    0.00    2/2             Treap::del(char
                          const*) [15]
[18]    0.0     0.00    0.00    2               Treap::erase(Node*,
   char const*) [18]
                        0.00    0.00    4/49            toLowerCase(char
                          *) [8]
                        0.00    0.00    2/2             Treap::merge(Node
                          *, Node*) [19]
                        0.00    0.00    2/6             Node::~Node()
                          [12]
-----------------------------------------------
                        0.00    0.00    2/2             Treap::erase(Node
                          *, char const*) [18]
[19]    0.0     0.00    0.00    2               Treap::merge(Node*,
   Node*) [19]
-----------------------------------------------
```

```
                0.00    0.00       1/1
                   _GLOBAL__sub_I__Z11toLowerCasePc [27]
[20]     0.0    0.00    0.00       1
  __static_initialization_and_destruction_0(int, int) [20]
-----------------------------------------------
                0.00    0.00       1/1           Treap::save(char
                   const*) [24]
[21]     0.0    0.00    0.00       1         Treap::saveToFile(
  Node*, std::basic_ofstream<char, std::char_traits<char> >&)
  [21]
-----------------------------------------------
                0.00    0.00       1/1           Treap::load(char
                   const*) [23]
[22]     0.0    0.00    0.00       1         Treap::loadFromFile(
  std::basic_ifstream<char, std::char_traits<char> >&) [22]
                0.00    0.00       1/6           Node::Node(char
                   const*, unsigned long, unsigned long) [11]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[23]     0.0    0.00    0.00       1         Treap::load(char
  const*) [23]
                0.00    0.00       1/1           Treap::
                   loadFromFile(std::basic_ifstream<char, std::
                   char_traits<char> >&) [22]
                0.00    0.00       1/2           Treap::clear(Node
                   *) [17]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[24]     0.0    0.00    0.00       1         Treap::save(char
  const*) [24]
                0.00    0.00       1/1           Treap::saveToFile
                   (Node*, std::basic_ofstream<char, std::
                   char_traits<char> >&) [21]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[25]     0.0    0.00    0.00       1         Treap::Treap() [25]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[26]     0.0    0.00    0.00       1         Treap::~Treap() [26]
                0.00    0.00       1/2           Treap::clear(Node
                   *) [17]
-----------------------------------------------
```

```
This table describes the call tree of the program, and was sorted
    by
the total amount of time spent in each function and its children.

Each entry in this table consists of several lines.  The line
   with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.
This line lists:
    index       A unique number given to each element of the table
       .
                Index numbers are sorted numerically.
                The index number is printed next to every function
                    name so
                it is easier to look up where the function is in
                   the table.

    % time      This is the percentage of the 'total' time that
       was spent
                in this function and its children.  Note that due
                   to
                different viewpoints, functions excluded by
                   options, etc,
                these numbers will NOT add up to 100%.

    self        This is the total amount of time spent in this
       function.

    children    This is the total amount of time propagated into
       this
                function by its children.

    called      This is the number of times the function was
       called.
                If the function called itself recursively, the
                   number
                only includes non-recursive calls, and is followed
                    by
                a '+' and the number of recursive calls.
```

name   The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self   This is the amount of time that was propagated directly from the function into this parent.

children   This is the amount of time that was propagated from the function's children into this parent.

called   This is the number of times this parent called the function ‘/’ the total number of times the function was called. Recursive calls to the function are not included in the number after the ‘/’.

name   This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word ‘<spontaneous>’ is printed in the ‘name’ field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self   This is the amount of time that was propagated directly

```
                from the child into the function.

     children   This is the amount of time that was propagated
        from the
                child's children to the function.

     called     This is the number of times the function called
                this child '/' the total number of times the child
                was called.  Recursive calls by the child are not
                listed in the number after the '/'.

     name       This is the name of the child.  The child's index
                number is printed after it.  If the child is a
                member of a cycle, the cycle number is printed
                between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The '+' recursive calls entry shows the number of function calls
    that
 were internal to the cycle, and the calls entry for each member
    shows,
 for that member, how many times it was called from other members
    of
 the cycle.
```

```
Index by function name

   [8] toLowerCase(char*)       [15] Treap::del(char const*) [16]
      Treap::split(Node*, char const*, Node*&, Node*&)
  [20] __static_initialization_and_destruction_0(int, int) (0.cpp)
      [9] Treap::find(Node*, char const*) [13] Treap::insert(Node
   *, Node*)
[11] Node::Node(char const*, unsigned long, unsigned long) [23]
   Treap::load(char const*) [14] Treap::search(char const*)
[12] Node::~Node()             [24] Treap::save(char const*) [25]
   Treap::Treap()
[21] Treap::saveToFile(Node*, std::basic_ofstream<char, std::
   char_traits<char> >&) [17] Treap::clear(Node*) [26] Treap::~
   Treap()
[22] Treap::loadFromFile(std::basic_ifstream<char, std::
   char_traits<char> >&) [18] Treap::erase(Node*, char const*)
[10] Treap::add(char const*, unsigned long) [19] Treap::merge(
   Node*, Node*)
```

Выводы

В результате данной лабораторной работы я познакомился с такими утилитами, как valgrind и gprof, которые позволяют отлаживать скомпилированные программы: находить ошибки, утечки памяти, на которые не может указать и заметить компилятор. Набор сведений, полученный этими утилитами, дает подробные сведения о программе, а также о её недостатках, которые можно исправить или устранить, тем самым оптимизировав код.