Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент:	М. М. Парфенов
Преподаватель:	С. А. Михайлова
Группа:	М8О-201Б-22
Дата:	
Оценка:	
Подпись:	

Москва, 2024

Лабораторная работа №3

Задача: **Задача**: 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.
Набор используемых средств: Valgrind и gprof

1 Дневник выполения работы

Исследование потребления памяти. Для этого будем использовать утилиту valgrind. Сначала проверим налачие утечек памяти. Для работы с valgrind желательно отключить оптимизацию, а также установить ключ отладки. Это позволит получить более достоверные и подробные данные:

```
[±main]-> g++ -O0 -g main.cpp -o output                       ~/Documents/MAI/ [20:42]
[±main]-> valgrind ./output < input.txt                       ~/Documents/MAI/ [20:42]
==390243== Memcheck, a memory error detector
==390243== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==390243== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==390243== Command: ./output
==390243==
==390243==
==390243== HEAP SUMMARY:
==390243==     in use at exit: 122,880 bytes in 6 blocks
==390243==   total heap usage: 2,917 allocs, 2,911 frees, 524,577 bytes allocated
==390243==
==390243== LEAK SUMMARY:
==390243==    definitely lost: 0 bytes in 0 blocks
==390243==    indirectly lost: 0 bytes in 0 blocks
==390243==      possibly lost: 0 bytes in 0 blocks
==390243==    still reachable: 122,880 bytes in 6 blocks
==390243==         suppressed: 0 bytes in 0 blocks
==390243== Rerun with --leak-check=full to see details of leaked memory
==390243==
==390243== For lists of detected and suppressed errors, rerun with: -s
==390243== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Утечек памяти обнаружено не было.

Давайте посмотрим на скорость работы. Для этого будем использовать утилиту gprof. Для её работы нужно скомпилировать программу с помощью ключа -pg, для более подробной информации следует использовать ключ -g, а также отключить оптимизацию. Затем запускаем программу как обычно. По завершению работы она создаст файл gmon.out:

```
[±main]-> g++ -pg main.cpp                                    ~/Documents/MAI/ [20:42]
[±main]-> ./a.out < input.txt                                 ~/Documents/MAI/ [20:45]
[±main]> gprof a.out                                         ~/Documents/MAI/ [20:45]
Flat profile:
```

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
100.05      0.01     0.01     2239     4.47     4.47  RBTree::insertValue(std::__cxx11
  0.00      0.01     0.00  1129472     0.00     0.00  __gnu_cxx::__normal_iterator<cha
  0.00      0.01     0.00  1120504     0.00     0.00  __gnu_cxx::__normal_iterator<cha
  0.00      0.01     0.00  1120504     0.00     0.00  __gnu_cxx::__normal_iterator<cha
  0.00      0.01     0.00   564736     0.00     0.00  bool __gnu_cxx::operator!=<char*
  0.00      0.01     0.00    38762     0.00     0.00  RBTree::compareString(std::__cxx
  0.00      0.01     0.00    11240     0.00     0.00  std::char_traits<char>::length(c
  0.00      0.01     0.00     6760     0.00     0.00  std::char_traits<char>::compare
  0.00      0.01     0.00     6760     0.00     0.00  bool std::operator==<char, std:
  0.00      0.01     0.00     4484     0.00     0.00  RBTree::toLower(std::__cxx11::ba
  0.00      0.01     0.00     4484     0.00     0.00  __gnu_cxx::__normal_iterator<cha
  0.00      0.01     0.00     4480     0.00     0.00  std::__new_allocator<char>::~__n
  0.00      0.01     0.00     4480     0.00     0.00  void std::__cxx11::basic_string<
  0.00      0.01     0.00     4480     0.00     0.00  std::__cxx11::basic_string<char
  0.00      0.01     0.00     4480     0.00     0.00  std::__cxx11::basic_string<char
  0.00      0.01     0.00     4480     0.00     0.00  std::__cxx11::basic_string<char
  0.00      0.01     0.00     2241     0.00     0.00  RBTree::deleteValue(std::__cxx11
  0.00      0.01     0.00     2240     0.00     0.00  Node::Node()
  0.00      0.01     0.00     2240     0.00     0.00  Node::~Node()
  0.00      0.01     0.00     2236     0.00     0.00  RBTree::fixDeleteRBTree(Node*)
  0.00      0.01     0.00     2233     0.00     0.00  RBTree::fixInsertRBTree(Node*)
  0.00      0.01     0.00     1118     0.00     0.00  RBTree::minRightNode(Node*)
  0.00      0.01     0.00     1067     0.00     0.00  RBTree::rotateLeft(Node*)
  0.00      0.01     0.00        4     0.00     0.00  RBTree::get(std::__cxx11::basic_
  0.00      0.01     0.00        3     0.00     0.00  RBTree::rotateRight(Node*)
  0.00      0.01     0.00        3     0.00     0.00  RBTree::saveFile(std::basic_ofst
  0.00      0.01     0.00        2     0.00     0.00  RBTree::erace(Node*)
  0.00      0.01     0.00        1     0.00     0.00  RBTree::loadFile(std::basic_ifst
  0.00      0.01     0.00        1     0.00     0.00  RBTree::RBTree()
  0.00      0.01     0.00        1     0.00     0.00  RBTree::~RBTree()

 %         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 2 byte(s) for 99.95% of 0.01 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.01                 main [1]
                0.01    0.00    2239/2239        RBTree::insertValue(std::__cxx11::ba
                0.00    0.00    6760/6760        bool std::operator==<char, std::char_
                0.00    0.00    4484/4484        RBTree::toLower(std::__cxx11::basic_
                0.00    0.00    2241/2241        RBTree::deleteValue(std::__cxx11::ba
                0.00    0.00       4/4           RBTree::get(std::__cxx11::basic_stri
                0.00    0.00       3/3           RBTree::saveFile(std::basic_ofstream
                0.00    0.00       1/1           RBTree::RBTree() [33]
                0.00    0.00       1/1           RBTree::loadFile(std::basic_ifstream
                0.00    0.00       1/1           RBTree::~RBTree() [34]
-----------------------------------------------
                0.01    0.00    2239/2239        main [1]
[2]    100.0    0.01    0.00    2239         RBTree::insertValue(std::__cxx11::basic_s
                0.00    0.00   23332/38762       RBTree::compareString(std::__cxx11::
                0.00    0.00    2239/2240        Node::Node() [22]
                0.00    0.00    2239/4480        std::__new_allocator<char>::~__new_a
                0.00    0.00    2239/4480        std::__cxx11::basic_string<char, std
                0.00    0.00    2233/2233        RBTree::fixInsertRBTree(Node*) [25]
-----------------------------------------------
                0.00    0.00 1129472/1129472     bool __gnu_cxx::operator!=<char*, st
[6]      0.0    0.00    0.00 1129472         __gnu_cxx::__normal_iterator<char*, std:
-----------------------------------------------
                0.00    0.00 1120504/1120504     __gnu_cxx::__normal_iterator<char*,
[7]      0.0    0.00    0.00 1120504         __gnu_cxx::__normal_iterator<char*, std:
-----------------------------------------------
                0.00    0.00 1120504/1120504     __gnu_cxx::__normal_iterator<char*,
[8]      0.0    0.00    0.00 1120504         __gnu_cxx::__normal_iterator<char*, std:
-----------------------------------------------
                0.00    0.00  564736/564736      __gnu_cxx::__normal_iterator<char*,
[9]      0.0    0.00    0.00  564736         bool __gnu_cxx::operator!=<char*, std::_
                0.00    0.00 1129472/1129472     __gnu_cxx::__normal_iterator<char*,
-----------------------------------------------
                0.00    0.00   15430/38762       RBTree::deleteValue(std::__cxx11::ba
                0.00    0.00   23332/38762       RBTree::insertValue(std::__cxx11::ba
[10]     0.0    0.00    0.00   38762         RBTree::compareString(std::__cxx11::basi
-----------------------------------------------
                0.00    0.00    4480/11240       std::__cxx11::basic_string<char, std
                0.00    0.00    6760/11240       bool std::operator==<char, std::char_
[11]     0.0    0.00    0.00   11240         std::char_traits<char>::length(char cons
-----------------------------------------------
                0.00    0.00    6760/6760        bool std::operator==<char, std::char_
[12]     0.0    0.00    0.00    6760         std::char_traits<char>::compare(char con
-----------------------------------------------
                0.00    0.00    6760/6760        main [1]
[13]     0.0    0.00    0.00    6760         bool std::operator==<char, std::char_tra
                0.00    0.00    6760/11240       std::char_traits<char>::length(char c
                0.00    0.00    6760/6760        std::char_traits<char>::compare(char
-----------------------------------------------
                0.00    0.00    4484/4484        main [1]
[14]     0.0    0.00    0.00    4484         RBTree::toLower(std::__cxx11::basic_strin
                0.00    0.00    4484/4484        __gnu_cxx::__normal_iterator<char*, s
-----------------------------------------------
                0.00    0.00    4484/4484        RBTree::toLower(std::__cxx11::basic_s
[15]     0.0    0.00    0.00    4484         __gnu_cxx::__normal_iterator<char*, std:
                0.00    0.00 1120504/1120504     __gnu_cxx::__normal_iterator<char*, s
                0.00    0.00 1120504/1120504     __gnu_cxx::__normal_iterator<char*, s
                0.00    0.00  564736/564736      bool __gnu_cxx::operator!=<char*, std
-----------------------------------------------
                0.00    0.00    2239/4480        RBTree::insertValue(std::__cxx11::bas
                0.00    0.00    2241/4480        RBTree::deleteValue(std::__cxx11::bas
[16]     0.0    0.00    0.00    4480         std::__new_allocator<char>::~__new_alloca
-----------------------------------------------
                0.00    0.00    4480/4480        std::__cxx11::basic_string<char, std
[17]     0.0    0.00    0.00    4480         void std::__cxx11::basic_string<char, std
                0.00    0.00    4480/4480        std::__cxx11::basic_string<char, std
                0.00    0.00    4480/4480        std::__cxx11::basic_string<char, std
-----------------------------------------------
                0.00    0.00    2239/4480        RBTree::insertValue(std::__cxx11::bas
                0.00    0.00    2241/4480        RBTree::deleteValue(std::__cxx11::bas
[18]     0.0    0.00    0.00    4480         std::__cxx11::basic_string<char, std::cha
                0.00    0.00    4480/11240       std::char_traits<char>::length(char c
                0.00    0.00    4480/4480        void std::__cxx11::basic_string<char
-----------------------------------------------
                0.00    0.00    4480/4480        void std::__cxx11::basic_string<char
[19]     0.0    0.00    0.00    4480         std::__cxx11::basic_string<char, std::cha
-----------------------------------------------
                0.00    0.00    4480/4480        void std::__cxx11::basic_string<char
[20]     0.0    0.00    0.00    4480         std::__cxx11::basic_string<char, std::cha
-----------------------------------------------
                0.00    0.00    2241/2241        main [1]
[21]     0.0    0.00    0.00    2241         RBTree::deleteValue(std::__cxx11::basic_s
                0.00    0.00   15430/38762       RBTree::compareString(std::__cxx11::b
                0.00    0.00    2241/4480        std::__cxx11::basic_string<char, std
                0.00    0.00    2241/4480        std::__new_allocator<char>::~__new_al
                0.00    0.00    2239/2240        Node::~Node() [23]
                0.00    0.00    2236/2236        RBTree::fixDeleteRBTree(Node*) [24]
                0.00    0.00    1118/1118        RBTree::minRightNode(Node*) [26]
```

```
-----------------------------------------------
                0.00    0.00       1/2240        RBTree::RBTree() [33]
                0.00    0.00    2239/2240        RBTree::insertValue(std::__cxx11::bas
[22]      0.0   0.00    0.00    2240         Node::Node() [22]
-----------------------------------------------
                0.00    0.00       1/2240        RBTree::~RBTree() [34]
                0.00    0.00    2239/2240        RBTree::deleteValue(std::__cxx11::bas
[23]      0.0   0.00    0.00    2240         Node::~Node() [23]
-----------------------------------------------
                0.00    0.00    2236/2236        RBTree::deleteValue(std::__cxx11::bas
[24]      0.0   0.00    0.00    2236         RBTree::fixDeleteRBTree(Node*) [24]
                0.00    0.00     357/1067        RBTree::rotateLeft(Node*) [27]
                0.00    0.00       3/3           RBTree::rotateRight(Node*) [29]
-----------------------------------------------
                0.00    0.00    2233/2233        RBTree::insertValue(std::__cxx11::bas
[25]      0.0   0.00    0.00    2233         RBTree::fixInsertRBTree(Node*) [25]
                0.00    0.00     710/1067        RBTree::rotateLeft(Node*) [27]
-----------------------------------------------
                0.00    0.00    1118/1118        RBTree::deleteValue(std::__cxx11::bas
[26]      0.0   0.00    0.00    1118         RBTree::minRightNode(Node*) [26]
-----------------------------------------------
                0.00    0.00     357/1067        RBTree::fixDeleteRBTree(Node*) [24]
                0.00    0.00     710/1067        RBTree::fixInsertRBTree(Node*) [25]
[27]      0.0   0.00    0.00    1067         RBTree::rotateLeft(Node*) [27]
-----------------------------------------------
                0.00    0.00       4/4           main [1]
[28]      0.0   0.00    0.00       4         RBTree::get(std::__cxx11::basic_string<c
-----------------------------------------------
                0.00    0.00       3/3           RBTree::fixDeleteRBTree(Node*) [24]
[29]      0.0   0.00    0.00       3         RBTree::rotateRight(Node*) [29]
-----------------------------------------------
                0.00    0.00       3/3           main [1]
[30]      0.0   0.00    0.00       3         RBTree::saveFile(std::basic_ofstream<char
-----------------------------------------------
                0.00    0.00       1/2           RBTree::~RBTree() [34]
                0.00    0.00       1/2           RBTree::loadFile(std::basic_ifstream<
[31]      0.0   0.00    0.00       2         RBTree::erace(Node*) [31]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[32]      0.0   0.00    0.00       1         RBTree::loadFile(std::basic_ifstream<char
                0.00    0.00       1/2           RBTree::erace(Node*) [31]
```

```
-----------------------------------------------
                0.00    0.00       1/1          main [1]
[33]     0.0    0.00    0.00       1        RBTree::RBTree() [33]
                0.00    0.00       1/2240       Node::Node() [22]
-----------------------------------------------
                0.00    0.00       1/1          main [1]
[34]     0.0    0.00    0.00       1        RBTree::~RBTree() [34]
                0.00    0.00       1/2          RBTree::erace(Node*) [31]
                0.00    0.00       1/2240       Node::~Node() [23]
-----------------------------------------------

 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) 2012-2024 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

  [22] Node::Node()            [28] RBTree::get(std::__cxx11::basic_string<char, std::
  [23] Node::~Node()           [31] RBTree::erace(Node*)   [11] std::char_traits<char>
  [27] RBTree::rotateLeft(Node*) [14] RBTree::toLower(std::__cxx11::basic_string<char
  [21] RBTree::deleteValue(std::__cxx11::basic_string<char, std::char_traits<char>, st
   [2] RBTree::insertValue(std::__cxx11::basic_string<char, std::char_traits<char>, st
  [29] RBTree::rotateRight(Node*) [33] RBTree::RBTree()   [18] std::__cxx11::basic_str
  [26] RBTree::minRightNode(Node*) [34] RBTree::~RBTree() [15] __gnu_cxx::__normal_ite
  [10] RBTree::compareString(std::__cxx11::basic_string<char, std::char_traits<char>,
  [24] RBTree::fixDeleteRBTree(Node*) [9] bool __gnu_cxx::operator!=<char*, std::__cxx
  [25] RBTree::fixInsertRBTree(Node*) [6] __gnu_cxx::__normal_iterator<char*, std::__
```

Как видно из результатов, 100% времени тратиться на функцию insertValue, скорее всего это вызвано тем, что в ней создаються новые узлы, это вызывают аллокацию памяти, что и занимает столько времени. К сожалению я не знаю как ускорить этот процесс, возможно стоит заранее выделять для программы участок в памяти, при условии, что заранее известно сколько узлов будет в дереве всего.

2 Выводы

В ходе выполнения данной лабораторной работы я ознакомился с профилированием, которое является важным элементом качественной разработки. Я изучил различные методы профилирования и применил их на практике. До этого я уже использовал утилиту Valgrind для контроля утечек памяти, а вот про gprof узнал впервые. Мне показалась эта утилита очень интересной, особенно в коммерческой разработке, где, казалось бы, незначительный выигрыш в производительности программы может значительно уменьшить затраты компании на обеспечение своего приложения.

Список литературы

[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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008