Лабораторная работа №3 по курсу дискретного анализа: исследование качества программ.

Выполнил студент группы М80-208Б-20 Зубко Дмитрий Валерьевич.

Условие

Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочетов, требуется их исправить.

Дневник выполнения работы

В данной лабораторной работе я воспользовался утилитами: **gprof** и **valgrind**.

1) **Gprof –** позволяет получить данные профилирования программы. То есть, эта утилита позволяет измерить время работы всех функций, методов и операторов программы, количество их вызовов, долю от общего времени работы программы.
   Запуск:
   1) Генерация тестов: **python main.py 1000000**
   2) Компиляция программы: **g++ -pg main.cpp bits.cpp node.cpp patricia.cpp -o main**
   3) Запуск программы: **./main < test.txt**
   4) Просмотр файла gmon.out, который является выводом нашей утилиты: **gprof main**

Результат вывода утилиты **gprof** следующий:

```
g++ -pg main.cpp bits.cpp node.cpp patricia.cpp -o main
dmitriy@dmitriy:/mnt/d/DA/lab2_v2/benchmark/da_lab_3_tests$ ./main < tests.txt
dmitriy@dmitriy:/mnt/d/DA/lab2_v2/benchmark/da_lab_3_tests$ gprof main
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 67.84      4.23     4.23 341927059     0.01     0.01  Node::GetIndex() const
  7.23      4.68     0.45 167564031     0.00     0.00  BitNumberIndexIsOne(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, int)
  5.31      5.01     0.33  3000000     0.11     0.39  ToLower(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&)
  4.98      5.32     0.31 387332565     0.00     0.00  bool __gnu_cxx::operator!=<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >
(__gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > const&, __gnu_cxx::__normal_iterator<char*, std::__
cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > const&)
  3.70      5.55     0.23 384332565     0.00     0.00  __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
 >::operator++()
  2.57      5.71     0.16 774665130     0.00     0.00  __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
 >::base() const
  2.25      5.85     0.14 384332565     0.00     0.00  __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
 >::operator*() const
  1.29      5.93     0.08  1999999     0.04     0.94  Patricia::FindElement(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)
  0.96      5.99     0.06  1000000     0.06     2.58  Patricia::AddElement(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, unsigne
d long long const&)
  0.64      6.03     0.04   707790     0.06     0.99  Patricia::ThirdCaseDeletion(Node*, Node*, Node*, Node*, Node*)
  0.64      6.07     0.04   707790     0.06     0.06  Node::GetValue() const
  0.48      6.10     0.03 53969957     0.00     0.00  Node::GetLeftNode()
  0.48      6.13     0.03 52485479     0.00     0.00  Node::GetRightNode()
  0.48      6.16     0.03 41645874     0.00     0.00  Node::GetKey[abi:cxx11]() const
  0.48      6.19     0.03   992491     0.03     0.22  FirstDifferenceInBit(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std::__
cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)
  0.32      6.21     0.02  3999998     0.01     0.01  __gnu_cxx::__enable_if<std::__is_char<char>::__value, bool>::__type std::operator==<char>(std::__cxx11::basic_string
<char, std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)
  0.16      6.22     0.01  1000000     0.01     1.52  Patricia::DeleteElement(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)
  0.16      6.23     0.01                             main
  0.00      6.23     0.00  3999999     0.00     0.00  Patricia::Empty()
  0.00      6.23     0.00  2339354     0.00     0.00  Node::SetRightNode(Node*)
  0.00      6.23     0.00  2338402     0.00     0.00  Node::SetLeftNode(Node*)
  0.00      6.23     0.00  1011531     0.00     0.00  std::char_traits<char>::compare(char const*, char const*, unsigned long)
  0.00      6.23     0.00   999999     0.00     0.01  bool std::operator!=<char, std::char_traits<char>, std::allocator<char> >(std::__cxx11::basic_string<char, std::char
_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)
  0.00      6.23     0.00   992492     0.00     0.00  Node::Node(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, unsigned long lon
```

Все остальные функции по данным gprof работали 0% времени, поэтому на скриншоте они отсутствуют. Больше всего времени заняла функция получения индекса, так как она вызывается чаще всего. Дальше идут функции для работы со строками: BitNumberIndexIsOne, ToLower.

2) **Valgrind** – мощное средство для поиска ошибок работы с памятью. Для проверки программы на ошибки с памятью нужно ввести в терминал:
   1) Генерация тестов: **python main.py 1000000**
   2) Компиляция программы: **g++ main.cpp bits.cpp node.cpp patricia.cpp -o main**
   3) Valgrind ./main < tests.txt

```
dmitriy@dmitriy:/mnt/d/DA/lab2_v2/benchmark/da_lab_3_tests$ make
g++ main.cpp bits.cpp node.cpp patricia.cpp -o main
dmitriy@dmitriy:/mnt/d/DA/lab2_v2/benchmark/da_lab_3_tests$ valgrind ./main < tests.txt
==51== Memcheck, a memory error detector
==51== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==51== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==51== Command: ./main
==51==
==51==
==51== HEAP SUMMARY:
==51==     in use at exit: 0 bytes in 0 blocks
==51==   total heap usage: 357,218 allocs, 357,218 frees, 45,438,849 bytes allocated
==51==
==51== All heap blocks were freed -- no leaks are possible
==51==
==51== For lists of detected and suppressed errors, rerun with: -s
==51== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
dmitriy@dmitriy:/mnt/d/DA/lab2_v2/benchmark/da_lab_3_tests$
```

Из вывода утилиты видно, что утечек памяти нет.

Вывод о найденных недочетах

Valgrind показал still reachable при запуске программы в первый раз. Чтобы это исправить я убрал из программы “строки-ускорители” std::ios::sync_with_stdio(false); std::cin.tie(nullptr); std::cout.tie(nullptr).

Общие выводы о выполнении лабораторной работы

Данная лабораторная работа познакомила меня с утилитой gprof, помогла мне закрепить навыки работы с утилитой для нахождения утечек памяти valgrind. Я проанализировал время выполнения функций программы. Нашёл утечки памяти и устранил их. В дальнейшем при написании программ я буду стараться использовать эти утилиты чаще.