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: А. В. Постнов
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №3

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.

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).

1 Описание

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

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

2 Консоль

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
56.68      1.74     1.74 278281190     0.01     0.01  TAVlTree::Height(TAVlTree::TNode
7.65      1.98     0.23  2000000     0.12     0.20  TAVlTree::Find(TAVlTree::TNode*,st
4.23      2.10     0.13 40954849     0.00     0.00  __gnu_cxx::__enable_if<std::__is_c
std::operator==<char>(std::__cxx11::basic_string<char,std::char_traits<char>,std::allo
4.07      2.23     0.12   608506     0.21     0.21  TAVlTree::FindMin(TAVlTree::TNode*
3.26      2.33     0.10 41281786     0.00     0.05  TAVlTree::Balance(TAVlTree::TNode*
2.93      2.42     0.09   999883     0.09     1.25  TAVlTree::Erase(TAVlTree::TNode*,s
2.28      2.49     0.07 85032909     0.00     0.01  TAVlTree::BFactor(TAVlTree::TNode*
2.28      2.56     0.07 54107686     0.00     0.02  TAVlTree::FixHeight(TAVlTree::TNod
2.28      2.63     0.07 54107686     0.00     0.00  int const& std::max<int>(int
const&,int const&)
2.28      2.70     0.07   999883     0.07     1.16  TAVlTree::Insert(TAVlTree::TNode*
long long)
2.28      2.77     0.07                             _init
1.30      2.81     0.04  2000000     0.02     0.23  TAVlTree::Exist(std::__cxx11::basi
1.30      2.85     0.04                             main
0.98      2.88     0.03 40949980     0.00     0.00  bool std::operator<<char,std::char
0.98      2.91     0.03 39927072     0.00     0.00  __gnu_cxx::__normal_iterator<char*
const
0.81      2.94     0.03 31209100     0.00     0.00  std::char_traits<char>::compare(ch
const*,char const*,unsigned long)
0.65      2.96     0.02 39927072     0.00     0.00  __gnu_cxx::__normal_iterator<char*
0.65      2.98     0.02  3200538     0.01     0.04  TAVlTree::RotateRight(TAVlTree::TN
0.49      2.99     0.01  3000000     0.01     0.01  bool std::operator==<char,std::cha
const*)
0.49      3.00     0.01  1000000     0.01     0.01  std::pair<unsigned long
long,bool>::pair<int,bool,true>(int&&,bool&&)
0.33      3.02     0.01 50519325     0.00     0.00  bool std::operator><char,std::char
0.33      3.02     0.01 43927072     0.00     0.00  __gnu_cxx::__normal_iterator<char*
const
0.33      3.04     0.01  3212412     0.00     0.03  TAVlTree::RotateLeft(TAVlTree::TNo
0.33      3.04     0.01  1000000     0.01     0.04  __gnu_cxx::__normal_iterator<char*
char)#2>(__gnu_cxx::__normal_iterator<char*,std::__cxx11::basic_string<char,std::char_
```

```
char)#2)
0.33      3.06     0.01  1000000     0.01     0.04  __gnu_cxx::__normal_iterator<char*
char)#1>(__gnu_cxx::__normal_iterator<char*,std::__cxx11::basic_string<char,std::char_
char)#1)
0.33      3.06     0.01   999883     0.01     0.01  TAVlTree::TNode::TNode(std::__cxx1
long long)
0.16      3.07     0.01                             std::operator|(std::_Ios_Openmode
0.00      3.07     0.00 31209100     0.00     0.00  std::__is_constant_evaluated()
0.00      3.07     0.00 21963536     0.00     0.00  bool __gnu_cxx::operator!=<char*,s
0.00      3.07     0.00  9981768     0.00     0.00  main::lambda(unsigned char)#2::ope
char) const
0.00      3.07     0.00  9981768     0.00     0.00  main::lambda(unsigned char)#1::ope
char) const
0.00      3.07     0.00  2000000     0.00     0.00  bool&& std::forward<bool>(std::rem
0.00      3.07     0.00  1000000     0.00     0.00  std::pair<unsigned long
long,bool>::pair<unsigned long long&,bool,true>(unsigned long long&,bool&&)
0.00      3.07     0.00  1000000     0.00     0.00  unsigned long long& std::forward<u
long long&>(std::remove_reference<unsigned long long&>::type&)
0.00      3.07     0.00  1000000     0.00     0.00  int&& std::forward<int>(std::remov
0.00      3.07     0.00   999883     0.00     1.25  TAVlTree::Erase(std::__cxx11::basi
0.00      3.07     0.00   999883     0.00     0.00  TAVlTree::TNode::~TNode()
0.00      3.07     0.00   999883     0.00     1.16  TAVlTree::Insert(std::__cxx11::bas
long long)
0.00      3.07     0.00   608506     0.00     0.06  TAVlTree::RemoveMin(TAVlTree::TNod
0.00      3.07     0.00        1     0.00     0.00  __static_initialization_and_destru
0.00      3.07     0.00        1     0.00     0.00  TAVlTree::Destroy(TAVlTree::TNode*
0.00      3.07     0.00        1     0.00     0.00  TAVlTree::Destroy()
0.00      3.07     0.00        1     0.00     0.00  TAVlTree::TAVlTree()
0.00      3.07     0.00        1     0.00     0.00  TAVlTree::~TAVlTree()

[alex@fedora lab3]$ valgrind --leak-check=full --leak-resolution=med ./lab2
<test.txt >out.txt
==23306== Memcheck,a memory error detector
==23306== Copyright (C) 2002-2022,and GNU GPL'd,by Julian Seward et al.
==23306== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==23306== Command: ./lab2
==23306==
==23306==
==23306== Process terminating with default action of signal 27 (SIGPROF)
==23306==    at 0x4D02003: __open_nocancel (open64_nocancel.c:39)
==23306==    by 0x4D10A6F: write_gmon (gmon.c:370)
```

```
==23306==    by 0x4D1123E: _mcleanup (gmon.c:444)
==23306==    by 0x4C410B4: __run_exit_handlers (exit.c:113)
==23306==    by 0x4C4122F: exit (exit.c:143)
==23306==    by 0x4C29516: (below main) (libc_start_call_main.h:74)
==23306==
==23306== HEAP SUMMARY:
==23306==     in use at exit: 107,336 bytes in 4 blocks
==23306==   total heap usage: 1,004 allocs,1,000 frees,171,336 bytes allocated
==23306==
==23306== LEAK SUMMARY:
==23306==    definitely lost: 0 bytes in 0 blocks
==23306==    indirectly lost: 0 bytes in 0 blocks
==23306==      possibly lost: 0 bytes in 0 blocks
==23306==    still reachable: 107,336 bytes in 4 blocks
==23306==         suppressed: 0 bytes in 0 blocks
==23306== Reachable blocks (those to which a pointer was found) are not shown.
==23306== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==23306==
==23306== For lists of detected and suppressed errors,rerun with: -s
==23306== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3 Анализ вывода утилит

Из **gprof** можно сделать вывод, что наиболее частый используемый метод - Height - получение высоты ноды(он и занимает наибольшее количество времени). из интересного, также достаточно много времени тратится на сравнение строк, оно и понятно, так как сравнение идет посимвольно.

Из вывода утилиты **valgrind**, можно сделать вывод, что программа работает корректно, но были найдены still rechable bytes, но в ходе поиска информации выяснилось, что они не создают проблем, которые могут вызвать настоящие утечки памяти.

4 Выводы

В ходе выполнения лабораторной работы по курсу «Дискретный анали», я изучил утилиты gprof и valgrind, нашел их для себя полезными, с помощью них проверил программу на корректность.

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] *АВЛ-дерево — Вики-конспекты ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево` (дата обращения: 23.04.2023).