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: Е. А. Суханов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №3

Задача: Провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочётов, требуется их исправить.

Для исследования моей программы я буду использовать valgrind и gprof.

С помощью valgrind можно найти утечки памяти, кроме этого, данная утилита имеет дополнительные модули, которые позволяют исследовать частоту вызовов функций, стек вызовов, работу кеша. Но использовать её я буду именно для исследования работы памяти.

Утилита gprof поставляется вместе с gnu компилятором. Она позволяет производить исследование скорости выполнения определенных функций. Кроме этого она позволяет построить граф вызовов функций. Данную утилиту я буду использовать для исследования времени выполнения определенных функций

1 Дневник выполнения работы

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

```
$ make clean && make
rm -f *.o solution banchmark
g++ -c -Wall -pedantic -std=c++14 -O0 -g main.cpp -o main.o
g++ -O0 -g main.o -o solution
```

Запускаем программу valgrind с флагом –leak-check=full, который включает поиск утечек памяти с подробным описанием:

```
$ valgrind --leak-check=full ./solution <../generated_tests/randomtest1000.txt
>/dev/null
==20992== Memcheck,a memory error detector
==20992== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==20992== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==20992== Command: ./solution
==20992==
==20992==
==20992== HEAP SUMMARY:
==20992==     in use at exit: 122,880 bytes in 6 blocks
==20992==   total heap usage: 1,321 allocs,1,315 frees,624,888 bytes allocated
==20992==
==20992== LEAK SUMMARY:
==20992==    definitely lost: 0 bytes in 0 blocks
==20992==    indirectly lost: 0 bytes in 0 blocks
==20992==      possibly lost: 0 bytes in 0 blocks
==20992==    still reachable: 122,880 bytes in 6 blocks
==20992==         suppressed: 0 bytes in 0 blocks
==20992== Reachable blocks (those to which a pointer was found) are not shown.
==20992== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==20992==
==20992== For lists of detected and suppressed errors,rerun with: -s
==20992== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Как видно, в данном случае утечек памяти не было. Definitely lost означает, что память была выделена, но указателя на нее нет. Indireclty lost означает, что на выделенную область существует указатель, но он потерялся. Possibly lost показывает,

что найден указатель, указывающий на часть области памяти, но valgrind не уверен в том, что указатель на начало области памяти до сих пор существует. Still reachable означает, что память была выделена, указатель на нее остался, но она не была освобождена. Для более подробного рассмотрения ошибок при работе с памятью нужно использовать флаг –show-leak-kinds=all:

```
$ valgrind  --leak-check=full --show-leak-kinds=all ./solution
<../generated_tests/randomtest1000.txt  >/dev/null
==21356== Memcheck,a memory error detector
==21356== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==21356== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==21356== Command: ./solution
==21356==
==21356==
==21356== HEAP SUMMARY:
==21356==     in use at exit: 122,880 bytes in 6 blocks
==21356==   total heap usage: 1,321 allocs,1,315 frees,624,888 bytes allocated
==21356==
==21356== 8,192 bytes in 1 blocks are still reachable in loss record 1 of 6
==21356==    at 0x483C583: operator new[](unsigned long)
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==21356==    by 0x4969F63: std::basic_filebuf<char,std::char_traits<char>>::
_M_allocate_internal_buffer()
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4967D49: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4919B47: std::ios_base::sync_with_stdio(bool)
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x10A5DD: main (main.cpp:75)
==21356==
==21356== 8,192 bytes in 1 blocks are still reachable in loss record 2 of 6
==21356==    at 0x483C583: operator new[](unsigned long)
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==21356==    by 0x4969F63: std::basic_filebuf<char,std::char_traits<char>>::
_M_allocate_internal_buffer()
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4967D49: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4919B68: std::ios_base::sync_with_stdio(bool)
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x10A5DD: main (main.cpp:75)
==21356==
==21356== 8,192 bytes in 1 blocks are still reachable in loss record 3 of 6
```

```
==21356==    at 0x483C583: operator new[](unsigned long)
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==21356==    by 0x4969F63: std::basic_filebuf<char,std::char_traits<char>>::
_M_allocate_internal_buffer()
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4967D49: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4919B88: std::ios_base::sync_with_stdio(bool)
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x10A5DD: main (main.cpp:75)
==21356==
==21356== 32,768 bytes in 1 blocks are still reachable in loss record 4 of
6
==21356==    at 0x483C583: operator new[](unsigned long)
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==21356==    by 0x496BD76: std::basic_filebuf<wchar_t,std::char_traits<wchar_t>>::
_M_allocate_internal_buffer()
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4967F39: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4919BFD: std::ios_base::sync_with_stdio(bool)
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x10A5DD: main (main.cpp:75)
==21356==
==21356== 32,768 bytes in 1 blocks are still reachable in loss record 5 of
6
==21356==    at 0x483C583: operator new[](unsigned long)
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==21356==    by 0x496BD76: std::basic_filebuf<wchar_t,std::char_traits<wchar_t>>::
_M_allocate_internal_buffer()
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4967F39: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4919C17: std::ios_base::sync_with_stdio(bool)
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x10A5DD: main (main.cpp:75)
==21356==
==21356== 32,768 bytes in 1 blocks are still reachable in loss record 6 of
6
==21356==    at 0x483C583: operator new[](unsigned long)
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==21356==    by 0x496BD76: std::basic_filebuf<wchar_t,std::char_traits<wchar_t>>::
_M_allocate_internal_buffer()
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
```

```
==21356==    by 0x4967F39: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x4919C30: std::ios_base::sync_with_stdio(bool)
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==21356==    by 0x10A5DD: main (main.cpp:75)
==21356==
==21356== LEAK SUMMARY:
==21356==    definitely lost: 0 bytes in 0 blocks
==21356==    indirectly lost: 0 bytes in 0 blocks
==21356==      possibly lost: 0 bytes in 0 blocks
==21356==    still reachable: 122,880 bytes in 6 blocks
==21356==         suppressed: 0 bytes in 0 blocks
==21356==
==21356== For lists of detected and suppressed errors,rerun with: -s
==21356== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Как мы видим, все отсавшиеся блоки были выделены на строке 75 в файле main.cpp в функции sync_with_stdio. Эта функция отключает синхронизацию потоков C++ и Си. Что позволяет ускорить ввод-вывод. Если её убрать, то ошибок работы с памятью не будет.

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

```
$ make clean && make
rm -f *.o solution banchmark
g++ -c -Wall -pedantic -std=c++14 -O0 -g -pg main.cpp -o main.o
g++ -O0 -g -pg main.o -o solution
```

Исследовать результаты выполнения можно с помощью программы gprof. Есть два варианта вывода: плоский профиль - выводит краткую информацию о времени, затраченном на выполнение определенных функций, и граф вызовов функций с более подробной информацией:

```
$ gprof ./solution ./gmon.out -p
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  ns/call  ns/call  name
```

```
24.62      0.44     0.44 39482317    11.16    11.16  TData::operator==(char
const*) const
20.70      0.81     0.37   508269   729.08   729.08  TData::TData()
18.46      1.14     0.33 40994240     8.06     8.06  TData::operator<(TData
const&) const
17.90      1.46     0.32  3000000   106.83   413.97  TRedBlackTree<TData,unsigned
long>::SearchTNode(TData const&,TRedBlackTree<TData,unsigned long>::TNode**)
6.71       1.58     0.12 39482317     3.04    14.97  TData::operator!=(TData
const&) const
3.36       1.64     0.06  1000000    60.09   854.65  TRedBlackTree<TData,unsigned
long>::Insert(TData const&,unsigned long const&)
3.36       1.70     0.06                             main
1.68       1.73     0.03 39482317     0.76    11.92  TData::operator==(TData
const&) const
1.68       1.76     0.03  1000000    30.05   444.02  TRedBlackTree<TData,unsigned
long>::Remove(TData const&)
0.56       1.77     0.01  3000000     3.34     3.34  std::setw(int)
0.56       1.78     0.01  1000000    10.02   423.98  TRedBlackTree<TData,unsigned
long>::Search(TData const&)
0.56       1.79     0.01   508267    19.70    19.70  TRedBlackTree<TData,unsigned
long>::InsertFixup(TRedBlackTree<TData,unsigned long>::TNode*)
0.00       1.79     0.00  3000000     0.00     3.34  operator>>(std::istream&,TData&)
0.00       1.79     0.00   508267     0.00   729.08  TRedBlackTree<TData,unsigned
long>::TNode::TNode()
0.00       1.79     0.00   500036     0.00     0.00  TRedBlackTree<TData,unsigned
long>::TNode::~TNode()
0.00       1.79     0.00   144024     0.00     0.00  TRedBlackTree<TData,unsigned
long>::LeftRotate(TRedBlackTree<TData,unsigned long>::TNode*)
0.00       1.79     0.00   135722     0.00     0.00
std::remove_reference<TData&>::type&& std::move<TData&>(TData&)
0.00       1.79     0.00   135722     0.00     0.00  std::remove_reference<unsigned
long&>::type&& std::move<unsigned long&>(unsigned long&)
0.00       1.79     0.00   132563     0.00     0.00  TRedBlackTree<TData,unsigned
long>::RightRotate(TRedBlackTree<TData,unsigned long>::TNode*)
0.00       1.79     0.00   132069     0.00     0.00
TRedBlackTree<TData,unsigned long>::RemoveFixup(TRedBlackTree<TData,unsigned
long>::TNode*,TRedBlackTree<TData,unsigned long>::TNode*)
0.00       1.79     0.00        1     0.00     0.00  _GLOBAL__sub_I_main
0.00       1.79     0.00        1     0.00     0.00
__static_initialization_and_destruction_0(int,int)
0.00       1.79     0.00        1     0.00     0.00  TRedBlackTree<TData,unsigned
```

```
long>::TRedBlackTree()
0.00      1.79     0.00        1     0.00     0.00  TRedBlackTree<TData,unsigned
long>::~TRedBlackTree()

%         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) 2012-2020 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.
```

Как видно из таблицы, функция сравнения строк является самой длительной по суммарному времени выполнения. Однако единичный вызов занимает не так много времени. Далее идет конструктор TData, который, хоть и вызывался в 100 раз меньше,

чем сравнение, но работает намного медленнее. Оказывается, эта функция работает медленно из-за цикла, который инициализирует буффер строки, хотя вполне достаточно задать только первый элемент. После чего данная функция выполняется на несколько порядков быстрее:

```
0.05     10.49     0.01   508269     0.01     0.01  TData::TData()
```

Однако, после проверки valgrind выясняется, что, при сохранени, в файл записываются неинциализированные байты. Я приведу ниже только несколько строк, так как само сообщение достаточно длинное:

```
$ valgrind  ./solution <../generated_tests/randomtest1000.txt  >/dev/null
==22555== Memcheck,a memory error detector
==22555== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==22555== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==22555== Command: ./solution
==22555==
==22555== Syscall param writev(vector[...]) points to uninitialised byte(s)
==22555==    at 0x4B715E7: writev (writev.c:26)
==22555==    by 0x492A80C: std::__basic_file<char>::xsputn_2(char const*,long,char
const*,long) (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==22555==    by 0x4968C05: std::basic_filebuf<char,std::char_traits<char>>::
xsputn(char const*,long) (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==22555==    by 0x4991586: std::ostream::write(char const*,long) (
in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==22555==    by 0x10B197: TRedBlackTree<TData,unsigned long>::
Save(char const*,TRedBlackTree<TData,unsigned long>const&) (rbtree.hpp:154)
==22555==    by 0x10A7AB: main (main.cpp:107)
...
...
...
==22555== HEAP SUMMARY:
==22555==     in use at exit: 122,880 bytes in 6 blocks
==22555==   total heap usage: 1,321 allocs,1,315 frees,624,888 bytes allocated
==22555==
==22555== LEAK SUMMARY:
==22555==    definitely lost: 0 bytes in 0 blocks
==22555==    indirectly lost: 0 bytes in 0 blocks
==22555==      possibly lost: 0 bytes in 0 blocks
==22555==    still reachable: 122,880 bytes in 6 blocks
==22555==         suppressed: 0 bytes in 0 blocks
==22555== Rerun with --leak-check=full to see details of leaked memory
```

```
==22555==
==22555== Use --track-origins=yes to see where uninitialised values come from
==22555== For lists of detected and suppressed errors,rerun with: -s
==22555== ERROR SUMMARY: 21 errors from 2 contexts (suppressed: 0 from 0)
```

Программа при этом остается корректной, но алгоритм сохранения и загрузки работает не самым лучшим образом. Да и чекер такое решение не примет. Я думаю, что правильным решением этой проблемы является сохранение только значащей информации. Т.е. Сохраняться будет сама строка, значение, а так же байт, который определяет имеются ли у данной вершины дети. К тому же такой способ ускорит сохранение и загрузку и уменьшит размер сохраненного файла. Однако сохранение строки неопределённой длины немного усложнит алгоритм загрузки и сохранения.

2 Выводы

В данной лабораторной работе я познакомился с некоторыми инструментами профилирования программы.

Valgrind позволяет отслеживать утечки памяти, промахи кеша, стек вызовов и некоторые другие вещи. gprof позволяет измерить приблизительное время выполнения каждой функции программы. Это позволяет находить медленные участки кода и оптимизировать в первую очередь именно их.

Однако, стоит отметить, что использование этих программ замедляет работу целевой программы.

Список литературы

[1] *Профилятор gprof*
URL: `https://www.opennet.ru/docs/RUS/gprof/` (дата обращения 21.01.2021).

[2] *Что такое valgrind и зачем он нужен*
URL: `http://alexott.net/ru/linux/valgrind/Valgrind.html` (дата обращения 21.01.2021).