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

Выполнил студент группы 08-208 МАИ *Попов Николай*.

Условие

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.

Метод решения

Изучение утилит valgrind и gprof для исследования качества программ и использование их для оптимизации программы.

- Valgrind — инструментальное ПО, предназначенное в основном для контроля использования памяти и обнаружения её утечек. С помощью этой утилиты можно обнаружить попытки использования (обращения) к неинициализированной памяти, работу с памятью после её освобождения и некоторые другие.
- 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

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования. Первоначальная версия программы содержала ошибку:

```
==8940== Memcheck, a memory error detector
==8940== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==8940== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright info
==8940== Command: ./a.out
==8940==
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
```

```
NoSuchWord
==8940==
==8940== HEAP SUMMARY:
==8940==     in use at exit: 123,177 bytes in 8 blocks
==8940==   total heap usage: 13 allocs, 5 frees, 197,716 bytes allocated
==8940==
==8940== LEAK SUMMARY:
==8940==    definitely lost: 297 bytes in 2 blocks
==8940==    indirectly lost: 0 bytes in 0 blocks
==8940==      possibly lost: 0 bytes in 0 blocks
==8940==    still reachable: 122,880 bytes in 6 blocks
==8940==         suppressed: 0 bytes in 0 blocks
==8940== Rerun with --leak-check=full to see details of leaked memory
==8940==
==8940== For lists of detected and suppressed errors, rerun with: -s
==8940== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

При помощи valgrind с флагом –leak-check=full можно понять, где была создана память, которая не освобождалась. В функции вставки создаётся новый объект с помощью new, но он не освобождается при удалении вершины. Для этого добавим деконструктор структуре node, который освободит память, созданную при вставке.

```
==9215== Memcheck, a memory error detector
==9215== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==9215== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright info
==9215== Command: ./a.out
==9215==
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
==9215==
==9215== HEAP SUMMARY:
==9215==     in use at exit: 297 bytes in 2 blocks
==9215==   total heap usage: 9 allocs, 7 frees, 79,956 bytes allocated
==9215==
==9215== 40 bytes in 1 blocks are definitely lost in loss record 1 of 2
==9215==    at 0x4841FB5: operator new(unsigned long) (vg_replace_malloc.c:472)
```

```
==9215==    by 0x40125F: newNode(char*, unsigned long) (in /home/helio/Desktop/DA/lab3/s
==9215==    by 0x401863: main (in /home/helio/Desktop/DA/lab3/src/a.out)
==9215==
==9215== 257 bytes in 1 blocks are definitely lost in loss record 2 of 2
==9215==    at 0x48432F3: operator new[](unsigned long) (vg_replace_malloc.c:714)
==9215==    by 0x40126D: newNode(char*, unsigned long) (in /home/helio/Desktop/DA/lab3/s
==9215==    by 0x401863: main (in /home/helio/Desktop/DA/lab3/src/a.out)
==9215==
==9215== LEAK SUMMARY:
==9215==    definitely lost: 297 bytes in 2 blocks
==9215==    indirectly lost: 0 bytes in 0 blocks
==9215==      possibly lost: 0 bytes in 0 blocks
==9215==    still reachable: 0 bytes in 0 blocks
==9215==         suppressed: 0 bytes in 0 blocks
==9215==
==9215== For lists of detected and suppressed errors, rerun with: -s
==9215== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Запустим программу со внесёнными изменениями, чтобы убедиться, что других ошибок нет

```
==9467== Memcheck, a memory error detector
==9467== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==9467== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright info
==9467== Command: ./a.out
==9467==
OK
Exist
OK
NoSuchWord
OK: 18446744073709551615
NoSuchWord
OK: 1
OK
NoSuchWord
NoSuchWord
==9467==
==9467== HEAP SUMMARY:
==9467==     in use at exit: 0 bytes in 0 blocks
==9467==   total heap usage: 9 allocs, 9 frees, 79,956 bytes allocated
==9467==
```

```
==9467== All heap blocks were freed -- no leaks are possible
==9467==
==9467== For lists of detected and suppressed errors, rerun with: -s
==9467== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

gprof

Используя утилиту gprof, можем отследить, где и сколько времени проводила программа, тем самым выявляя слабые участки. Возьмем достаточно большой тест (1000000 строк) и вызовем gprof

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]     96.5    0.02    1.07                 main [1]
                0.89    0.00 1000002/1000002     search(node*, char*) [2]
                0.09    0.01  288003/288003      insert(node*&, node*) [3]
                0.03    0.00 1000002/1000002     toLower(char*) [5]
                0.03    0.00       1/1           destroy(node*) [6]
                0.01    0.00 1667354/1667354     std::basic_istream<char, std::char_trai
                0.01    0.00   45213/45213       remove(node*&, char*) [9]
                0.00    0.00  288003/288003      node::node(char*, unsigned long) [16]
-----------------------------------------------
                             21678849             search(node*, char*) [2]
                0.89    0.00 1000002/1000002     main [1]
[2]     78.8    0.89    0.00 1000002+21678849 search(node*, char*) [2]
                             21678849             search(node*, char*) [2]
-----------------------------------------------
                             5713815             insert(node*&, node*) [3]
                0.09    0.01  288003/288003      main [1]
[3]      8.8    0.09    0.01  288003+5713815 insert(node*&, node*) [3]
                0.01    0.00  191513/191513      split(node*, node*&, node*&, char*) [8]
                             5713815             insert(node*&, node*) [3]
-----------------------------------------------
                0.03    0.00 1000002/1000002     main [1]
[5]      2.7    0.03    0.00 1000002         toLower(char*) [5]
-----------------------------------------------
                              485580             destroy(node*) [6]
                0.03    0.00       1/1           main [1]
[6]      2.7    0.03    0.00       1+485580  destroy(node*) [6]
                0.00    0.00  242790/288003      node::~node() [17]
                             485580             destroy(node*) [6]
-----------------------------------------------
```

```
                                574748             split(node*, node*&, node*&, char*) [8]
                0.01    0.00  191513/191513      insert(node*&, node*) [3]
[8]      0.9    0.01    0.00  191513+574748  split(node*, node*&, node*&, char*) [8]
                                574748             split(node*, node*&, node*&, char*) [8]
-----------------------------------------------
                                933474             remove(node*&, char*) [9]
                0.01    0.00   45213/45213       main [1]
[9]      0.9    0.01    0.00   45213+933474  remove(node*&, char*) [9]
                0.00    0.00   45213/45213       merge(node*, node*) [18]
                0.00    0.00   45213/288003      node::~node() [17]
                                933474             remove(node*&, char*) [9]
-----------------------------------------------
                0.00    0.00  288003/288003      main [1]
[16]     0.0    0.00    0.00  288003         node::node(char*, unsigned long) [16]
-----------------------------------------------
                0.00    0.00   45213/288003      remove(node*&, char*) [9]
                0.00    0.00  242790/288003      destroy(node*) [6]
[17]     0.0    0.00    0.00  288003         node::~node() [17]
-----------------------------------------------
                                44567              merge(node*, node*) [18]
                0.00    0.00   45213/45213       remove(node*&, char*) [9]
[18]     0.0    0.00    0.00   45213+44567   merge(node*, node*) [18]
                                44567              merge(node*, node*) [18]
-----------------------------------------------
```

- `main` функция была вызвана 96,5% времени выполнения программы и сама занимает 0,02% времени. Она вызывает:
  - `search`
  - `insert`
  - `toLower`
  - `destroy`
  - `std::operator»`
  - `remove`
- `search` функция была вызвана 78,8% времени и занимает 0,89% времени. Она вызывает саму себя и `search`.
- `insert` функция была вызвана 8,8% времени и занимает 0,09% времени. Она вызывает `split`.
- `toLower` функция занимает 2,7% времени и была вызвана 1000002 раза.

- `destroy` функция занимает 2,7% времени и была вызвана 1 раз.
- `std::operator»` функция занимает 0,9% времени и была вызвана 1667354 раза.
- `split` функция занимает 0,9% времени и была вызвана 191513 раз.
- `remove` функция занимает 0,9% времени и была вызвана 45213 раза.

Выводы

При выполнении лабораторной работы я познакомился с профилированием, крайне необходимым для качественной разработки, изучила возможные методы работы с ним, применив их на практике. Ранее я не использовала утилиты Valgrind для контроля утечек памяти, а также gprof, которая выводит число вызовов функций при работе программы, определяет время работы каждой функции как обособленно, так и в сравнении с общим временем работы программы, что позволяет найти наиболее часто используемую функцию и в первую очередь оптимизировать именно её.