Лабораторная работа №3, по курсу дискретного анализа: Исследование качества программы:

Выполнил студент группы М8О-203Б-22 Мудров Павел Федорович.

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.

Результатом лабораторной работы является отчет, 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 недочетах. Сравнение работы исправлен- ной 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 (напри- мер, gcov).

Метод решения

Valgrind. valgrind - это многоцелевой инструмент профилирования кода и отладки памяти для Linux. Чаще всего утилита используется для обнаружения ошибок при работе с памятью.

Инструмент memcheck утилиты valgrind позволяет обнаружить следующие виды ошибок:

1. Invalid read / Invalid write - неверное использование указателей, выход за границы массива, считывание или запись в ячейку памяти, не принадлежащую программе.

2. Conditional jump or move depends on uninitialised value(s) - использование неинициализированных переменных.

3. Invalid free / delete / delete[] - ошибка в использовании функций для освобождения памяти. Например, повторное освобождение памяти или освобождение памяти, выделенной через new, при помощи free.

4. Memory leaks - выделенная память не освобождается после завершения работы программы, возникает утечка памяти. Большое количество утечек памяти может негативно сказаться на работе операционной системы.

Утилита valgrind выводит пользователю список всех найденных ошибок. К каждой ошибке прикрепляется состояние стека вызовов на момент её обнаружения. При помо- щи этой информации пользователь может определить, какая функция вызвала ту или иную ошибку. В случае обнаружения утечки памяти valgrind может вывести состояние стека вызовов на момент выделения памяти для проблемной переменной и её адрес. В некоторых случаях это помогает быстрее устранить ошибку.

Gprof. Многие программы обладают жёсткими требованиями по производительно-

сти, поэтому работу многих функций приходится оптимизировать по максимуму. Одна- ко не всегда оптимизация функции приведёт к значительному росту производительно- сти: если данная функция вызывается всего пару раз, то её оптимизация будет не очень уж и полезной. Чтобы определить слабые места в коде, необходимо воспользоваться профилировщиком. Я буду рассматривать утилиту gprof (gnu profiler). Профилировщи- ки собирают данные во время работы программы. С некоторой периодичностью профилировщик останавливает программу, записывает текущее состояние стека вызовов и возобновляет её работу. После завершения сбора данных он собирает статистику, по которой можно понять, сколько процентов от общего времени работы программы занимает время работы какой-либо функции. Функции, которые имеют наибольшую долю рабочего времени, являются первоочерёдными кандидатами на оптимизацию. Стоит заметить, что для достоверности собираемой информации программу под профилировщиком следует запускать на больших тестах, охватывающих все функции программы.

Консоль:

Вначале проведем проверка инструментом valgrind:

```
==77868==
==77868== HEAP SUMMARY:
==77868==     in use at exit: 0 bytes in 0 blocks
==77868==   total heap usage: 2,602 allocs, 2,602 frees, 214,028 bytes allocated
==77868==
==77868== All heap blocks were freed -- no leaks are possible
==77868==
==77868== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

После анализа работы valgrind можно сделать вывод, что утечки памяти не обнаружены.

Теперь попробуем прогнать программу через профилировщик. Для того чтобы gprof собрал необходимую информацию о работе программы, требуется скомпилировать её с ключом -pg и запустить её на достаточно большом тесте. Профилировщик соберет необходимую информацию в файл gmon.out. Затем нужно вызвать утилиту gprof, которая на основе собранных данных сформирует отчёт.

```
➜ lab2 gprof ./a.out gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 57.14      0.04     0.04   266545     0.00     0.00  _fini
 42.86      0.07     0.03       14     2.14     2.14  main
  0.00      0.07     0.00   179372     0.00     0.00  erase(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)
  0.00      0.07     0.00   100000     0.00     0.00  __do_global_dtors_aux
  0.00      0.07     0.00     1174     0.00     0.00  __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::operator+(long) const
  0.00      0.07     0.00      216     0.00     0.00  std::char_traits<char>::compare(char const*, char const*, unsigned long)
  0.00      0.07     0.00      153     0.00     0.00  __gnu_cxx::__ops::_Iter_equals_val<char const> __gnu_cxx::__ops::__iter_equals_val<char const>(char const&)
  0.00      0.07     0.00       64     0.00     0.00  __gnu_cxx::__ops::_Iter_equals_val<char const>::_Iter_equals_val(char const&)
  0.00      0.07     0.00       61     0.00     0.00  DecartDictionary::add(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 0.07 0.00 46 0.00 0.00 DecartDictionary::remove(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)
0.00 0.07 0.00 44 0.00 0.00 bool __gnu_cxx::__ops::_Iter_equals_val<char const>::operator()<__gnu_cxx::__normal_iterator<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> > >)
0.00 0.07 0.00 16 0.00 0.00 Node::Node(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 0.07 0.00 1 0.00 0.00 printDecartTree(Node*, int, char)

% time — the percentage of the total running time of the program used by this function.

cumulative seconds — a running sum of the number of seconds accounted for by this function and those listed above it.

self seconds — the number of seconds accounted for by this 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 ms/call — the average number of milliseconds spent in this function per call, if this function is profiled, else blank.

total ms/call — the average number of milliseconds spent in this 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 4 byte(s) for 14.29% of 0.07 seconds

index % time self children called name
[1] 100.0 0.07 0.00 266544+179388 <cycle 1 as a whole> [1]
0.04 0.00 266545+532299 _fini <cycle 1> [2]
0.03 0.00 14+5461 main <cycle 1> [4]
0.00 0.00 179372 erase(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) <cycle 1> [12]
0.00 0.00 1 printDecartTree(Node*, int, char) <cycle 1> [22]

532299 _fini <cycle 1> [2]
1 printDecartTree(Node*, int, char) <cycle 1> [22]
0.00 0.00 1/266544 merge(Node*, Node*) [7]
0.01 0.00 33323/266544 to_lower(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [6]
0.03 0.00 99761/266544 split(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, Node*&, Node*&) [5]
0.04 0.00 133459/266544 search(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [3]
[2] 57.1 0.04 0.00 266545+532299 _fini <cycle 1> [2]
0.00 0.00 216/1174 __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::operator+(long) const [14]
33019 erase(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) <cycle 1> [12]
14 main <cycle 1> [4]
532299 _fini <cycle 1> [2]

<spontaneous>
[3] 50.1 0.00 0.04 search(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [3]
0.04 0.00 133459/266544 _fini <cycle 1> [2]
0.00 0.00 33458/100000 __do_global_dtors_aux [13]

5461 main <cycle 1> [4]
14 _fini <cycle 1> [2]
[4] 42.9 0.03 0.00 14+5461 main <cycle 1> [4]

146353 erase(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) <cycle 1> [12]
5461 main <cycle 1> [4]

<spontaneous>
[5] 37.4 0.00 0.03 split(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, Node*&, Node*&) [5]
0.03 0.00 99761/266544 _fini <cycle 1> [2]
0.00 0.00 66542/100000 __do_global_dtors_aux [13]

<spontaneous>
[6] 12.5 0.00 0.01 to_lower(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [6]
0.01 0.00 33323/266544 _fini <cycle 1> [2]

<spontaneous>
[7] 0.0 0.00 0.00 merge(Node*, Node*) [7]
0.00 0.00 1/266544 _fini <cycle 1> [2]

33019 _fini <cycle 1> [2]
146353 main <cycle 1> [4]
[12] 0.0 0.00 0.00 179372 erase(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) <cycle 1> [12]
1 printDecartTree(Node*, int, char) <cycle 1> [22]

0.00 0.00 33458/100000 search(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [3]
0.00 0.00 66542/100000 split(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, Node*&, Node*&) [5]
[13] 0.0 0.00 0.00 100000 __do_global_dtors_aux [13]

0.00 0.00 12/1174 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&) [35]
0.00 0.00 16/1174 DecartDictionary::find(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [26]
0.00 0.00 216/1174 _fini <cycle 1> [2]
0.00 0.00 930/1174 __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::operator++() [33]

[14] 0.0 0.00 0.00 1174 __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::operator+(long) const [14]
0.00 0.00 46/61 DecartDictionary::add(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&) [18]

0.00 0.00 216/216 std::iterator_traits<__gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > >::difference_type std::distance<__gnu_cxx::__normal_iterator<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> > >, __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >) [43]
[15] 0.0 0.00 0.00 216 std::char_traits<char>::compare(char const*, char const*, unsigned long) [15]

0.00 0.00 153/153 __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::__normal_iterator(char* const&) [32]
[16] 0.0 0.00 0.00 153 __gnu_cxx::__ops::_Iter_equals_val<char const> __gnu_cxx::__ops::__iter_equals_val<char const>(char const&) [16]
0.00 0.00 64/64 __gnu_cxx::__ops::_Iter_equals_val<char const>::_Iter_equals_val(char const&) [17]

0.00 0.00 64/64 __gnu_cxx::__ops::_Iter_equals_val<char const> __gnu_cxx::__ops::__iter_equals_val<char const>(char const&) [16]
[17] 0.0 0.00 0.00 64 __gnu_cxx::__ops::_Iter_equals_val<char const>::_Iter_equals_val(char const&) [17]

0.00 0.00 3/61 DecartDictionary::find(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [26]
0.00 0.00 12/61 DecartDictionary::DecartDictionary() [29]
0.00 0.00 46/61 __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::operator+(long) const [14]
[18] 0.0 0.00 0.00 61 DecartDictionary::add(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&) [18]

0.00 0.00 1/46 DecartDictionary::find(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [26]
0.00 0.00 12/46 __gnu_cxx::__normal_iterator<char*,

std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::base() const [36]

0.00 0.00 33/46 std::iterator_traits<__gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > >::difference_type std::__distance<__gnu_cxx::__normal_iterator<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> > >, __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >, std::random_access_iterator_tag) [40]

[19] 0.0 0.00 0.00 46 DecartDictionary::remove(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [19]

0.00 0.00 44/44 std::iterator_traits<__gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > >::difference_type std::__distance<__gnu_cxx::__normal_iterator<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> > >, __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >, std::random_access_iterator_tag) [40]

[20] 0.0 0.00 0.00 44 bool __gnu_cxx::__ops::_Iter_equals_val<char const>::operator()<__gnu_cxx::__normal_iterator<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> > >) [20]

0.00 0.00 16/16 std::iterator_traits<__gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > >::difference_type std::distance<__gnu_cxx::__normal_iterator<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> > >, __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >) [43]

[21] 0.0 0.00 0.00 16 Node::Node(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&) [21]

---

1 erase(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) <cycle 1> [12]

[22] 0.0 0.00 0.00 1 printDecartTree(Node*, int, char) <cycle 1> [22]

1 _fini <cycle 1> [2]

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.

notice and this notice are preserved.

Index by function name

[22] printDecartTree(Node*, int, char) [17] __gnu_cxx::__ops::_Iter_equals_val<char const>::_Iter_equals_val(char const&) [13] __do_global_dtors_aux
[12] erase(Node*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [20] bool __gnu_cxx::__ops::_Iter_equals_val<char const>::operator()<__gnu_cxx::__normal_iterator<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> > >) [2] _fini
[18] DecartDictionary::add(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&) [16] __gnu_cxx::__ops::_Iter_equals_val<char const> __gnu_cxx::__ops::__iter_equals_val<char const>(char const&) [4] main
[19] DecartDictionary::remove(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) [14] __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::operator+(long) const [1] <cycle 1>
[21] Node::Node(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&) [15] std::char_traits<char>::compare(char const*, char const*, unsigned long)
➜ lab2

Анализ профиля времени выполнения с использованием gprof показывает, какие функции в программе занимают наибольшее время выполнения. Вот что мы видим в плоском профиле (flat profile):

Профиль времени выполнения

- **_fini**: 57.14% времени (0.04 секунды)
  - Функция `_fini` занимает больше половины общего времени выполнения программы. Эта функция обычно вызывается при завершении программы для выполнения финализации, такой как очистка ресурсов.
- **main**: 42.86% времени (0.03 секунды)
  - Функция `main` — это точка входа в программу, и она занимает оставшуюся часть значительного времени выполнения.
- **erase**: 0.00% времени
  - Функция `erase` вызвана 179372 раза, но не занимает заметного времени в профиле. Это

может быть связано с тем, что вызовы этой функции очень быстрые и занимают пренебрежимо малое время по сравнению с остальными функциями.

- **__do_global_dtors_aux**: 0.00% времени
  - Функция `__do_global_dtors_aux` вызвана 100000 раз. Эта функция отвечает за вызов деструкторов для глобальных объектов при завершении программы.
- **__gnu_cxx::__normal_iterator::operator+**: 0.00% времени
  - Этот оператор используется для итерации по контейнерам и не занимает заметного времени.
- **std::char_traits<char>::compare**: 0.00% времени
  - Функция сравнения строк, вызвана 216 раз, и тоже не занимает значительного времени.
- **__gnu_cxx::__ops::_Iter_equals_val**: 0.00% времени
  - Несколько вариаций этой функции (конструктор и оператор) также не занимают значительного времени.
- **DecartDictionary::add**: 0.00% времени
  - Функция добавления в словарь вызвана 61 раз и не занимает заметного времени.