Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: Н. А. Ширяев
Группа: М8О-208Б-22

Дата:
Оценка:
Подпись:

Москва, 2024

Задание

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

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, geov).

Valgrind

```
testtest: /Desktop/DA/lab3valgrind --leak-check=full./test == 9547 ==
    Memcheck, amemoryerrordetector == 9547 ==
    Copyright(C)2002 - 2017, andGNUGPL'd, byJulianSewardetal. == 9547 ==
    UsingValgrind - 3.18.1andLibVEX; rerunwith -hforcopyrightinfo == 9547 == Command :
    ./test == 9547 == i = 0; 0.912435ms == 9547 ==== 9547 == HEAPSUMMARY :==
    9547 == inuseatexit : 0bytesin0blocks == 9547 == totalheapusage :
    200,002allocs, 200,002frees, 34,073,728bytesallocated == 9547 ==== 9547 ==
    Allheapblockswerefreed - -noleaksarepossible == 9547 ==== 9547 ==
    Forlistsofdetectedandsuppressederrors, rerunwith : -s == 9547 == ERRORSUMMARY :
    0errorsfrom0contexts(suppressed : 0from0)
```

Gprof

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 64.29      0.09     0.09                             main
 14.29      0.11     0.02   100000     0.00     0.00  insert(node*&,
    node*)
 14.29      0.13     0.02                             _init
  7.14      0.14     0.01        1    10.00    10.00  frame_dummy
```

```
 %         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.
```



```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 7.14% of 0.14 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     85.7    0.09    0.03                 main [1]
                0.02    0.00  100000/100000      insert(node*&, node*)
                   [2]
                0.01    0.00       1/1           frame_dummy [4]
-----------------------------------------------
                0.02    0.00  100000/100000      main [1]
[2]     14.3    0.02    0.00  100000         insert(node*&, node*) [2]
```

```
-----------------------------------------------
                                                <spontaneous>
[3]     14.3    0.02    0.00                  _init [3]
-----------------------------------------------
                                99999           frame_dummy [4]
                0.01    0.00       1/1          main [1]
[4]      7.1    0.01    0.00       1+99999   frame_dummy [4]
                                99999           frame_dummy [4]
-----------------------------------------------
```

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.
```

```
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.
```

```
Index by function name

   [2] insert(node*&, node*)   [4] frame_dummy
   [3] _init                   [1] main
```

Выводы

В рамках лабораторной работы было рассмотрено профилирование, важное для повышения качества разработки, а также изучены различные методы работы с ним. Использовались утилиты Valgrind для выявления утечек памяти и gprof, которая анализирует количество вызовов функций программы и их время выполнения. Это позволяет сравнивать работу каждой функции с общим временем работы программы и выявлять наиболее часто используемые функции для приоритетной оптимизации.