

Лабораторная работа № 3 по курсу Дискретный Анализ. Исследование качества программ

Выполнил студент группы М8О-207Б-21 МАИ Друхольский Александр.

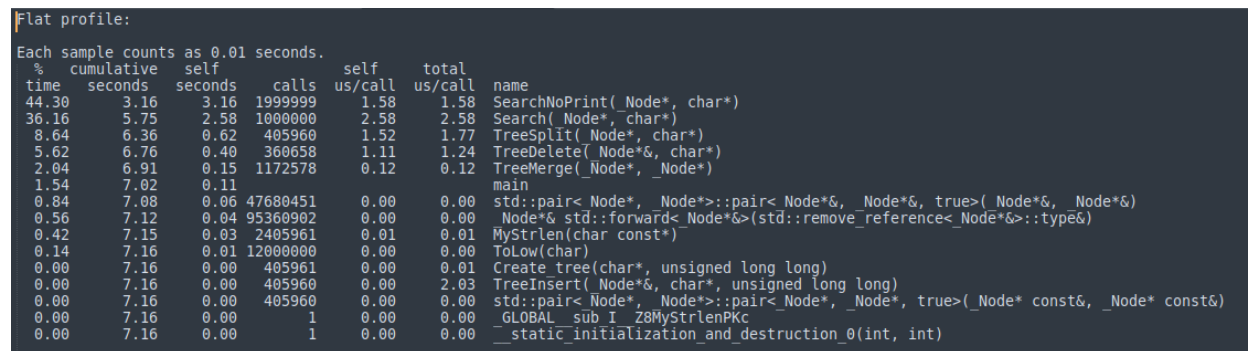
Условие

Для реализации словаря из прошлой лабораторной работы необходимо провести исследование времени выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочётов необходимо исправить их

Выполнение работы

Для исследования используем утилиты *valgrind* и *gprof*.

gprof вычисляет время работы всех функций, количество их вызова, а также показывает в процентах, сколько времени заняло выполнение конкретной функции от общего времени. Скомпилируем *main.cpp* с флагом *-pg*, чтобы получить файл *gmon.out*. Используем для теста файл с запросами вставки, поиска, удаления 1000000 элементов (на каждую операцию), длина строк - 4.



```
Flat profile:
Each sample counts as 0.01 seconds.
%   cumulative   self           self         total
time  seconds    seconds   calls   us/call   us/call   name
44.30    3.16      3.16  1999999    1.58    1.58  SearchNoPrint(Node*, char*)
36.16    5.75      2.58  1000000    2.58    2.58  Search(Node*, char*)
8.64     6.36      0.62   405960    1.52    1.77  TreeSplit(Node*, char*)
5.62     6.76      0.40   360658    1.11    1.24  TreeDelete(Node*&, char*)
2.04     6.91      0.15   1172578    0.12    0.12  TreeMerge(Node*, _Node*)
1.54     7.02      0.11                main
0.84     7.08      0.06  47680451    0.00    0.00  std::pair<Node*, Node*>::pair<Node*&, Node*&, true>(Node*&, Node*&)
0.56     7.12      0.04  95360902    0.00    0.00  Node*& std::forward<Node*&>(std::remove_reference<Node*&>::type&)
0.42     7.15      0.03  2405961    0.01    0.01  MyStrlen(char const*)
0.14     7.16      0.01  12000000    0.00    0.00  ToLow(char)
0.00     7.16      0.00   405961    0.00    0.01  Create tree(char*, unsigned long long)
0.00     7.16      0.00   405960    0.00    2.03  TreeInsert(Node*&, char*, unsigned long long)
0.00     7.16      0.00   405960    0.00    0.00  std::pair<Node*, Node*>::pair<Node*, Node*, true>(Node* const&, Node* const&)
0.00     7.16      0.00      1      0.00    0.00  _GLOBAL_sub_I_Z8MyStrlenPKc
0.00     7.16      0.00      1      0.00    0.00  _static_initialization_and_destruction_0(int, int)
```

Рис. 1: Результат работы gprof

Утилита *gprof* показала, что большую часть времени занимает выполнение операции *SearchNoPrint()* - мы её вызываем при попытке выполнения любой операции, так как нам нужно проверять наличие элемента в дереве (количество вызовов 1999999, а не 2000000, так как для создания дерева мы поиск не выполняем). Следующее по занимаемому времени - *Search()* - выполняет запрос поиска. Операции вставки, удаления, разбиения и склеивания дерева занимают меньше времени, так как из-за небольшой длины строки многие строки в запросах повторяются, а значит после поиска в дереве мы не будем выполнять операцию.

Утилита *valgrind* позволяет проследить утечки памяти и RE. Скомпилируем файл с флагом *-g*, чтобы отобразить строки с ошибками. Затем запускаем утилиту.

```

==32080== HEAP SUMMARY:
==32080==      in use at exit: 16 bytes in 8 blocks
==32080==    total heap usage: 19 allocs, 11 frees, 75,088 bytes allocated
==32080==
==32080== LEAK SUMMARY:
==32080==    definitely lost: 16 bytes in 8 blocks
==32080==    indirectly lost: 0 bytes in 0 blocks
==32080==    possibly lost: 0 bytes in 0 blocks
==32080==    still reachable: 0 bytes in 0 blocks
==32080==    suppressed: 0 bytes in 0 blocks
==32080== Rerun with --leak-check=full to see details of leaked memory
==32080==
==32080== For lists of detected and suppressed errors, rerun with: -s
==32080== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Рис. 2: Результат работы valgrind

Как видим, присутствуют утечки памяти. Скорее всего это из-за выделения памяти под строки, ведь когда я чищу ноду, я не делаю `free(tree->key)`, хотя память была выделена. Попробуем исправить это.

```

==32764==
==32764== HEAP SUMMARY:
==32764==      in use at exit: 0 bytes in 0 blocks
==32764==    total heap usage: 19 allocs, 19 frees, 75,088 bytes allocated
==32764==
==32764== All heap blocks were freed -- no leaks are possible
==32764==
==32764== For lists of detected and suppressed errors, rerun with: -s
==32764== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Рис. 3: Результат работы valgrind с исправлением

Выводы

В ходе работы мной были изучены инструменты для анализа и оптимизации работы программного кода.

С помощью утилиты *gprof* можно увидеть наименее оптимизированные участки кода. Мы видим, что именно занимает большую часть время и, опираясь на это, можем изменить логику алгоритма, чтобы лишний раз не тратить время на некоторых функциях.

Утилита *valgrind* показывает утечки памяти. В моём случае, у меня были явные проблемы с этим, так как я забыл чистить память, выделенную под строки. На больших данных до исправления утечки были весьма большими

Данные утилиты полезны при отладке кода и для его оптимизации. Благодаря ним можно отследить слабые места своего программного кода и исправить их.