Лабораторная работа № 3 по курсу дискретного анализа: Сбаласированные деревья

Выполнил студент группы М8О-201Б-21 МАИ *Рудаков Никита*.

Условие

Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти.

gprof

Основная информация

Утилита gprof позволяет измерить время работы всех функций, методов и операторов программы, количество их вызовов и долю от общего времени работы программы в процентах.

Команды для работы с утилитой

Сначала скомпилируем исходную программу с ключом $-pg$:

```
g++ main.cpp -pg -o lab
```

Затем запустим программу, передав ей на ввод файл *test.txt*, в котором содержится по 10000 команд на вставку, поиск и удаление:

```
./main < test.txt > out.txt
```

Выполнив эту команду, заметим, что кроме файла *out.txt*, в котором содержатся результаты выполнения команд из *test.txt*, появился файл *gmon.out*, в котором содержится вся информация, предоставляемая утилитой gprof. Чтобы получить текстовый файл, выполним следующую команду:

```
gprof main gmon.out > profile-data.txt
```

Таким образом, выполнив 3 простые команды, получили текстовый файл с подробной информацией о времени работы и вызовах всех функций и операторов, которые использовались в программе.

Результат работы утилиты

Ниже приведена таблица, в которую перенесены данные из файла $profile - data.txt$, полученного с помощью утилиты gprof.

time %	seconds	calls	name
10.00	0.10	20000	RBTree::searchTree()
10.00	0.10	20000	RBTree::searchNodeHelper()
6.32	0.06	13601	RBTree::transplant()
5.00	0.05	10000	RBTree::deleteNode()
5.00	0.05	10000	RBTree::deleteNodeHelper()
4.86	0.05	9982	RBTNode::RBTNode()
4.86	0.05	9981	RBTree::insertNode()
4.85	0.05	9978	RBTree::fixedInsert()
3.95	0.04	7363	RBTree::fixedDelete()
2.67	0.03	5236	RBTree::searchMinimum()
2.53	0.03	5189	RBTree::leftRotate()
2.14	0.02	4575	RBTree::rightRotate()

Остальные функции, по данным результатам измерений утилиты gprof, работали примерно 0 секунд, но были функции, которые работали значительное количество времени, но я не стал их добавлять в таблицу, поскольку эти функции не являются мною написанными. Из полученных данных следует, что большая часть времени работы программы тратится на поиск элемента в дереве.

Time Limit

Во время выполнения работы, я пару раз получал ошибку TL. Первый раз я смог ее исправить, разобрался в своем коде и переделал логику поиска, она была неверная. Второй раз я уже не смог разобраться, где была ошибка. При помощи grprof, я получил следующее:

time %	seconds	calls	name
50.00	0.01	30000	to_lower()
25.00	0.01	3858783	__gnu_cxx::__normal_iterator<...>::operator*() const
25.00	0.02	1	std::_Vector_base<...>::_Vector_impl::_Vector_impl()

Вероятнее всего, где-то в main было многократное преобразование в нижний регистр и программа зациклилась, возможно, она не смогла преобразовать какую-то строку. Пришлось переделывать реализацию.

valgrind

Valgrind является самым распространённым инструментом для отслеживания утечек памяти и других ошибок, связанных с памятью. Чтобы проверить программу *main* на проблемы с памятью, выполним следующую команду:

```
valgrind ./main < test.txt > out.txt
```

В результате выполнения этой команды получаем следующее сообщение:

```
    ==48652== Memcheck, a memory error detector
==48652== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==48652== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==48652== Command: ./main
==48652==
==48652==
==48652== HEAP SUMMARY:
==48652==     in use at exit: 122,952 bytes in 7 blocks
==48652==   total heap usage: 57,804 allocs, 57,797 frees, 6,326,850 bytes al
==48652==
==48652== LEAK SUMMARY:
==48652==    definitely lost: 0 bytes in 0 blocks
==48652==    indirectly lost: 0 bytes in 0 blocks
==48652==      possibly lost: 0 bytes in 0 blocks
==48652==    still reachable: 122,952 bytes in 7 blocks
==48652==         suppressed: 0 bytes in 0 blocks
==48652== Rerun with --leak-check=full to see details of leaked memory
==48652==
==48652== For lists of detected and suppressed errors, rerun with: -s
==48652== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

С помощью Valgrind обнаружили несколько незначительных ошибок и неосвобождённую память после выполнения программы.

Выводы

Проделав лабораторную работу, я познакомился с полезной утилитой gprof, необходимой для измерения времени работы программы и отдельных её частей, определил функции, в которых программа циклилась, закрепил навыки работы с утилитой valgrind. В своей программе не обнаружил неосвобожденную память.