

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

Выполнил студент группы М8О-308Б-22 *Зиновьев Данил*.

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.

Дневник выполнения работы

Для выполнения лабораторной работы я воспользовался двумя утилитами: **valgrind**, с помощью которой происходила отладка программы на этапе её написания, и **gprof**, с которой я ознакомился в процессе подготовки данной лабораторной работы.

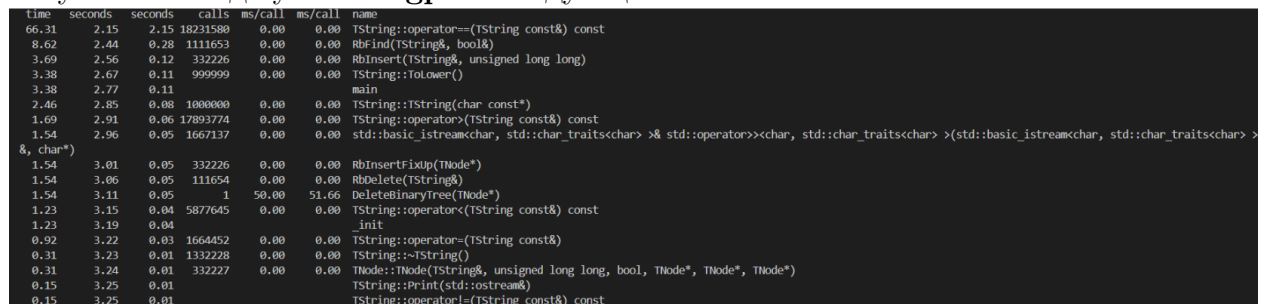
Для начала я запущу свою программу с утилитой **gprof**. Данная утилита хороша тем, что позволяет увидеть время работы всех функций, реализованных в программе, количество их вызовов и вычисляет процентное соотношение работы конкретной функции по сравнению с работой всей программы.

Скомпилируем код при помощи команды **g++ main.cpp -pg**, где **-pg** - специальный флаг, позволяющий вывести необходимые данные в файл **gmon.out**.

Предварительно сгенерируем файл **in.txt**, состоящий из 1000000 команд на поиск, вставку и удаление. После - запустим нашу программу с этим файлом: **./a.out < in.txt**

После компиляции создался файл **gmon.out**, который является выводом нашей утилиты. Посмотрим его при помощи команды **gprof a.out**.

Результат вывода утилиты **gprof** следующий:



time	seconds	seconds	calls	ms/call	ms/call	name
66.31	2.15	2.15	18231580	0.00	0.00	TString::operator==(TString const&) const
8.62	2.44	0.28	1111653	0.00	0.00	RbFind(TString&, bool&)
3.69	2.56	0.12	332226	0.00	0.00	Rbinsert(TString&, unsigned long long)
3.38	2.67	0.11	999999	0.00	0.00	TString::ToLower()
3.38	2.77	0.11				main
2.46	2.85	0.08	1000000	0.00	0.00	TString::TString(char const*)
1.69	2.91	0.06	17893774	0.00	0.00	TString::operator>(TString const&) const
1.54	2.96	0.05	1667137	0.00	0.00	std::basic_istream<char, std::char_traits<char> >& std::operator>><char, std::char_traits<char> >(std::basic_istream<char, std::char_traits<char> >&, char*)
1.54	3.01	0.05	332226	0.00	0.00	RbInsertFixUp(TNode*)
1.54	3.06	0.05	111654	0.00	0.00	RbDelete(TString&)
1.54	3.11	0.05	1	50.00	51.66	DeleteBinaryTree(TNode*)
1.23	3.15	0.04	5877645	0.00	0.00	TString::operator<(TString const&) const
1.23	3.19	0.04				_init
0.92	3.22	0.03	1664452	0.00	0.00	TString::operator=(TString const&)
0.31	3.23	0.01	1332228	0.00	0.00	TString::~~TString()
0.31	3.24	0.01	332227	0.00	0.00	TNode::TNode(TString&, unsigned long long, bool, TNode*, TNode*)
0.15	3.25	0.01				TString::Print(std::ostream&)
0.15	3.25	0.01				TString::operator!=(TString const&) const

Остальные функции показывали 0.00 в графе time.

В результате мы видим, что для теста в 1000000 строк, наибольшее время выполнения наблюдается у оператора сравнения строк, функций удаления и поиска элементов в красно-чёрном дереве. Лидерство оператора сравнения может быть связано с тем, что многие строки в тестах достигают длины в 256 символов, что довольно-таки много.

Теперь запустим программу через **valgrind** с флагом **-leak-check=full**. Поскольку в процессе отладки программы использовалась данная утилита. То в итоговом варианте

кода утечек памяти не наблюдается. Однако, далее будут описаны ошибки в работе с памятью, возникшие в процессе отладки кода. Итак, вывод результатов использования утилиты выглядит следующим образом:

```
==6874==
==6874== HEAP SUMMARY:
==6874==    in use at exit: 0 bytes in 0 blocks
==6874== total heap usage: 3,325,182 allocs, 3,325,182 frees, 77,255,120 bytes allocated
==6874==
==6874== All heap blocks were freed -- no leaks are possible
==6874==
==6874== For lists of detected and suppressed errors, rerun with: -s
==6874== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Далее будут описаны ошибки в работе с памятью, найденные с помощью **valgrind** в процессе работы программы:

1. Была обнаружена ошибка в конструкторе узла дерева *TNode*. Поскольку изначально объект класса *TString* передавался ему не по ссылке, это способствовало утечкам памяти при удалении дерева. Ошибка была устранена после того, как принимаемый тип конструктора *TNode* был изменён на *TString&*
2. Была обнаружена ошибка в функции *Load*, осуществляющей загрузку дерева из файла. Изначально внутри неё создавалась переменная типа *char**, в которую читалась строка из файла, она удалялась один раз после завершения цикла, в котором происходило заполнение дерева. Это способствовало утечкам памяти, так как, по сути, удалялась только последняя считываемая строка. Ошибка была устранена после того, как удаление содержимого данной переменной стало выполняться на каждой итерации цикла.

Общие выводы о выполнении лабораторной работы

Лабораторная работа № 3 помогла мне закрепить навыки работы с утилитой **valgrind**, а также познакомила с утилитой **gprof**, которая помогает пользователю увидеть время выполнения отдельных функций, что может быть полезным для больших программ. Помимо этого, я смог добиться полной очистки динамически выделенной памяти.