Лабораторная работа № 3 по курсу дискретного анализа: сбалансированные деревья

Выполнил студент группы М8О-208Б-21 МАИ *Белоносов Кирилл.*

Условие

Общая постановка задачи:
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.
Результатом лабораторной работы является отчёт, состоящий из:

1. Дневника выполнения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы.
2. Выводов о найденных недочётах.
3. Сравнение работы исправленной программы с предыдущей версией.
4. 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, gcov).

Метод решения

Для исследования нашей программы используем 2 утилиты: **gprof** и **valgrind**:

1. **valgrind** - инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования
2. **gprof** - инструмент для профилирования, рассчитывает количество времени, которое проходит в процессе исполнения каждой из процедур или функций. Также для более удобного представления результата профилирования будем использовать утилиту gprof2dot. Она строит граф вызовов на основе результата работы gprof.

Исследование программы

Valgrind:

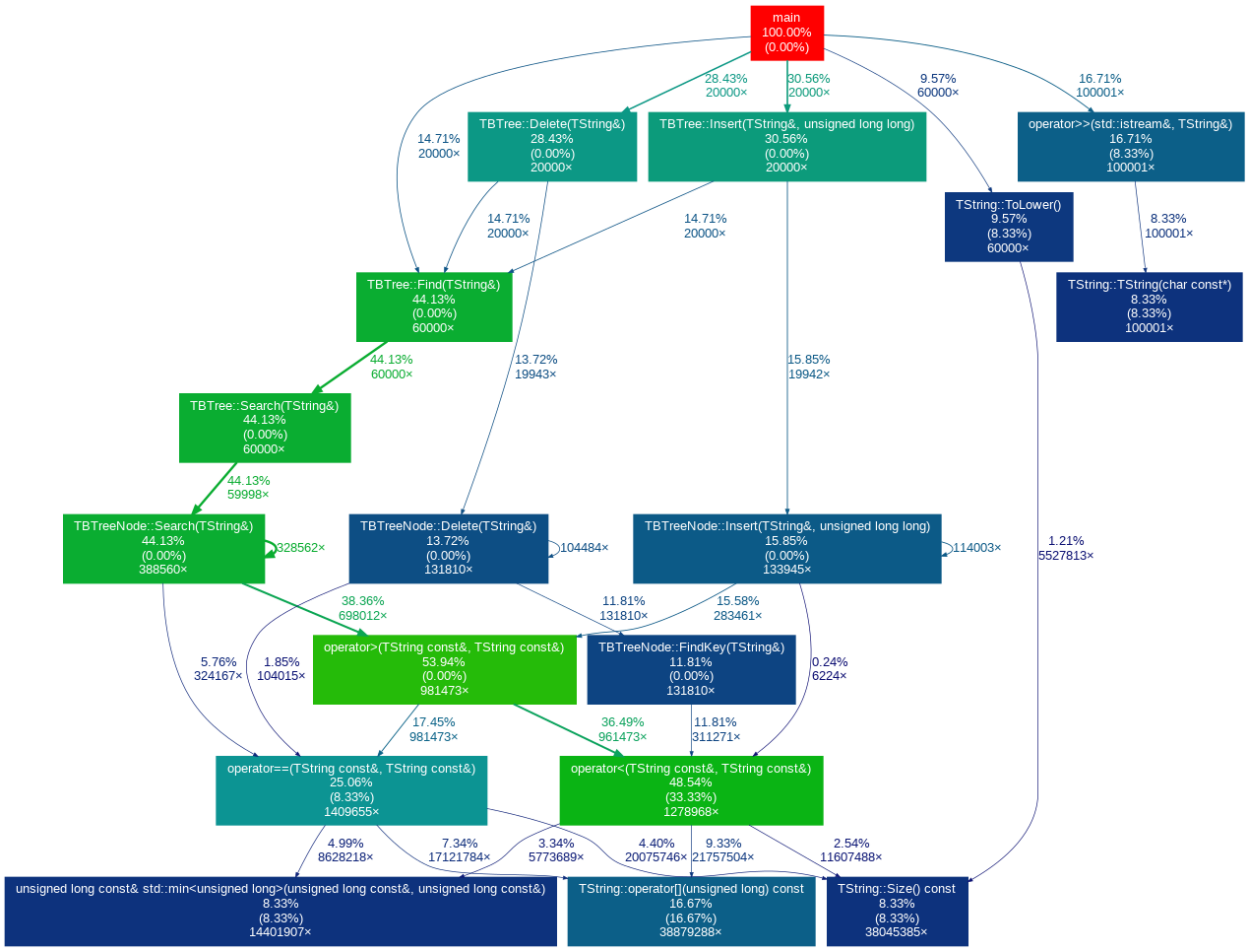
Для начала исследуем нашу программу на утечки памяти с помощью valgrind. Запустим сложный тест на несколько тысяч операций вставки и удаления.

```
==854==
==854== HEAP SUMMARY:
==854==     in use at exit: 122,880 bytes in 6 blocks
==854==   total heap usage: 52,320 allocs, 52,314 frees, 3,832,518
   bytes allocated
==854==
==854== LEAK SUMMARY:
==854==    definitely lost: 0 bytes in 0 blocks
==854==    indirectly lost: 0 bytes in 0 blocks
==854==      possibly lost: 0 bytes in 0 blocks
==854==    still reachable: 122,880 bytes in 6 blocks
==854==         suppressed: 0 bytes in 0 blocks
==854== Reachable blocks (those to which a pointer was found) are
   not shown.
==854== To see them, rerun with: --leak-check=full
   --show-leak-kinds=all
==854==
==854== For lists of detected and suppressed errors, rerun with: -s
==854== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from
   0)
```

В результате мы видим, что программа не имеет утечек памяти. Блок Still reachable обычно означает, что valgrind нашел указатель на начало не освобожденного блока памяти, что во многих случаях связано с выделением глобальных переменных и т.п. вещей и утечкой не является.

Gprof:

Сначала с помощью команды *g++ -pg main.cpp -o main* скомпилируем программу, для того, чтобы она поддерживала профилятор. Потом запустим ее, для создания файла gmon.out. После запустим сам профилятор *gprof ./main gmon.out | gprof2dot | dot -Tpng -o out.png*. В итоге мы получим граф вызовов.



Оценивая полученные результаты, можно сказать, что большая часть времени работы программы выпадает на операции сравнения строк, причем 38% падает на операцию поиска и удаление, а остальное на вставку. Также интересно заметить, время вставки примерно равно времени удаления. Не удивительно, так как обе эти операции иногда требуют перестройки структуры дерева. Также 44% занимает поиск в B-дереве, что является логичным, так как он требуется как при вставке и удалении, так и при запросе данных по ключу. Отдельно хочется отметить, что операции на строках также занимают довольно внушительное время, в частности функция *ToLower*.

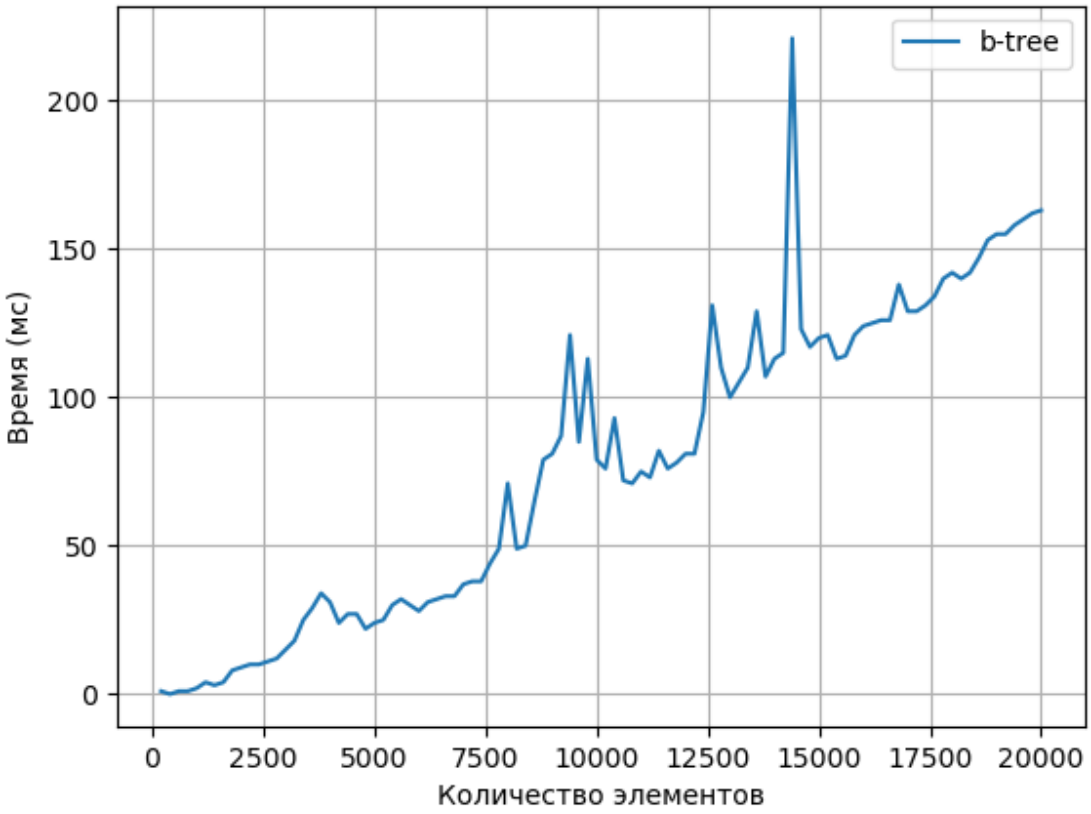
Дневник отладки

1. Проверить программу на возможные утечки памяти с помощью утилиты valgrind
2. Иследовать граф вызовов с помощью gprof, вывести его с помощью gprof2dot

Тест производительности

Тестирование программы производилось на 100 сгенерированных тестах с шагом в 200 элементов. Значения числа в диапазоне $[0; 18446744073709551615]$, а строка имела длинну $[1; 256]$.

Вставка:



Возможно, что появление "пиков"связано с более длинными строками, что приводит как большему числу сравнений и большему количеству вызова функций ToLower и конструктора имплементированной строки

Выводы

При выполнении данной лабораторной работы я познакомился с новым для себя инструментом - gprof. С помощью него я смог увидеть какие функции и методы занимают большую часть времени работы программы. Также я освежил знания в утилите valgrind успешно проверив мою программу на утечки памяти. Нельзя не отметит, что такое изучение работы программы позволяет не только исправить ошибки, но полностью пересмотреть её архитектуру, что позволит уменьшить время работы программы.

Приобритенные знания помогут не только при дальнейшем выполнии работ по дисциплине "Дискретной анализ но и в последующей работе.