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

Выполнил студент группы М8О-212Б-22 МАИ *Мамонтов Егор*.

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.

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, gcov).

Метод решения

В рамках выполнения лабораторной работы я буду использовать следующие утилиты:

- Анализ времени работы: gprof
- Анализ потребления памяти: valgrind

gprof

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

Для работы с утилитой gprof было необходимо скомпилировать программу с ключом `-pg` После запуска полученного исполняемого файла появился файл gmon.out, в котом содержалась информация предоставленная для анализа программы. Далее этот файл был обработан gprof для получения текстового файла с подробной информацией о времени работы и вызовах всех функций и операторов, которые использовались в программе.

% time	self seconds	calls	name
22.25	0.02	200220	`Node::search(...)`
22.25	0.02	200114	`Node::insert(...)`
22.25	0.02	199665	`Node::remove(...)`
11.12	0.01	5167452	`__gnu_cxx:: ... std::operator==<char>(...)`
11.12	0.01	200220	`AVL_tree::search(...)`
11.12	0.01	104557	`Node::rebalance(...)`

Время работы остальных функций по данным gprof заняло меньше 0.01 секунды, поэтому они не были внесены в таблицу.

Как можно заметить из таблицы, больше всего времени заняли функции поиска ключей в дереве, удаление и вставка узла в дерево. Функции `Node::search`, `Node::insert` и `Node::remove` выполняют рекурсивные операции в дереве. Рекурсивные вызовы могут быть затратными по памяти и времени, особенно если дерево имеет большую глубину или содержит много элементов.

В два раза меньше времени занимают функции ребалансировки, сравнения строк и поиска, но уже в классе AVL_tree.

По поводу функции `Node::rebalance`. В AVLдереве необходимо поддерживать баланс высоты поддеревьев. Операции вставки и удаления могут приводить к необходимости балансировки дерева, что также может занимать значительное количество времени.

По поводу функции `AVL_tree::search`. Она занимает в 2 раза меньше времени по сравнению с `Node::search`, т.к функция `Node::search` может вызывать саму себя целых 2 раза.

По поводу функции `__gnu_cxx::__enable_if<...>:: ... std::operator==<char>()`. Это встроенная функция сравнения строк, поэтому она занимает много времени.

Valgrind

Valgrind является утилитой для поиска ошибок в работе с памятью в программе, таких как утечки памяти и выход за границу массива.

В результате исследования программы `Valgrind` была выявлена утечка памяти в функции вставки: `by 0x10D53A: AVL_tree::insert(...)`. Для устранения ошибки нужно было очищать дерево в конце. Я создал функцию и добавил строку `tree.clear_tree()` в конце функции `main()` и всё починилось.

Все утечки были устранены. Результат работы `Valgrind`:

```
==10208== Memcheck, a memory error detector
==10208== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10208== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10208== Command: ./main
==10208==
==10208==
==10208== HEAP SUMMARY:
==10208== in use at exit: 122,880 bytes in 6 blocks
==10208== total heap usage: 140,277 allocs, 140,271 frees, 8,330,878 bytes allocated
==10208==
==10208== LEAK SUMMARY:
==10208== definitely lost: 0 bytes in 0 blocks
==10208== indirectly lost: 0 bytes in 0 blocks
==10208== possibly lost: 0 bytes in 0 blocks
==10208== still reachable: 122,880 bytes in 6 blocks
==10208== suppressed: 0 bytes in 0 blocks
==10208== Rerun with -leak-check=full to see details of leaked memory
==10208==
==10208== For lists of detected and suppressed errors, rerun with: -s
==10208== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Сообщение `still reachable: 122,880 bytes in 6 blocks` происходит из-за использования `ios::sync_with_stdio()`. Я его использовал для ускорения работы программы и это не является утечкой памяти.

Выводы

В ходе выполнения практической работы я овладел методами выявления узких мест и проблем в управлении памятью. Использование утилиты gprof для анализа времени выполнения программы помогает выявить функции, затрачивающие больше всего времени, что, в свою очередь, способствует оптимизации кода с целью улучшения производительности. Применение valgrind для анализа работы с памятью позволяет выявить утечки памяти, неправильное использование указателей и другие проблемы, которые могут привести к непредсказуемому поведению программы.

Исследование качества программ является необходимым для разработки надежного, эффективного и безопасного программного обеспечения. Это способствует улучшению пользовательского опыта, сокращению числа ошибок и сбоев, повышению производительности и снижению затрат на обслуживание.