Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: А. М. Голев
Преподаватель: С. А. Михайлова
Группа: М8О-201Б

Дата: 29.04.25
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №3

Задача: 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).

В моём случае необходимо совершить профилирование кода красно-черного дерева.

1 Описание

Красно-черное дерево представляет собой бинарное дерево поиска с одним дополнительным битом цвета в каждом узле. Цвет узла может быть либо красным (RED), либо черным (BLACK). В соответствии с накладываемыми на узлы дерева ограничениями ни один простой путь от корня в красно-черном дереве не отличается от другого по длине более чем в два раза, так что красно-черные деревья являются приближенно сбалансированными. [1]

В данной лабораторной работе было проведено исследование производительности реализации словаря на основе структуры данных красно-черное дерево. Основные задачи:

1. Анализ скорости выполнения операций
2. Исследование потребления памяти
3. Выявление и исправление узких мест в производительности

2 Дневник выполнения работы

1 Исходная реализация

Первоначальная версия программы использовала следующие подходы:

- Реализация красно-черного дерева с базовыми операциями (вставка, удаление, поиск)
- Преобразование строк к нижнему регистру в дополнительном буфере
- Использование стандартных инструментов для работы со строками

Были проведены тесты с помощью Valgrind и gprof.

2 Результаты профилирования исходной версии

Анализ gprof показал следующие проблемные места:

- 80.65% времени в `readKey`
- 13.98% в `searchRBTree`

Выполнение остальных функций занимает меньше 5-и процентов времени работы программы.
Результаты проверки на утечки памяти:

```
==13682== Memcheck,a memory error detector
==13682== Copyright (C) 2002-2022,and GNU GPL'd,by Julian Seward et al.
==13682== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==13682== Command: ./app.out
==13682==
==13682==
==13682== HEAP SUMMARY:
==13682==     in use at exit: 115,911 bytes in 894 blocks
==13682==   total heap usage: 3,584 allocs,2,690 frees,374,752 bytes allocated
==13682==
==13682== 115,911 bytes in 894 blocks are definitely lost in loss record 1
of 1
==13682==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-l
==13682==    by 0x10A7DE: UI (in /home/smoking_elk/discran-labs/lab3/app.out)
==13682==    by 0x10A66F: main (in /home/smoking_elk/discran-labs/lab3/app.out)
```

```
==13682==
==13682== LEAK SUMMARY:
==13682==    definitely lost: 115,911 bytes in 894 blocks
==13682==    indirectly lost: 0 bytes in 0 blocks
==13682==      possibly lost: 0 bytes in 0 blocks
==13682==    still reachable: 0 bytes in 0 blocks
==13682==         suppressed: 0 bytes in 0 blocks
==13682==
==13682== For lists of detected and suppressed errors,rerun with: -s
==13682== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

3 Выявленные проблемы

Основные узкие места:

1. Чтение ключа занимает много времени
2. Поиск, вероятно, неэффективный

Также выявлена утечка памяти в функции UI.

4 Исправленная реализация

Попытки ускорения поиска при помощи хэширования и считывания ключа при помощи арифметики указателей, более эффективного приведения к нижнему регистру и изменения строки inplace не привели к ускорению программы, что свидетельствует о том, что текущая реализация является оптимальной.

В ходе анализа утечки памяти в функции UI было обнаружено, что проблема содержится не в теле этой функции, а в функции `removeRBTree`. Перед удалением узла не освобождалась память, выделяемая под ключ. Этот недочет был исправлен.

5 Результаты после оптимизации

Время выполнения работы программы не изменилось.

Утечка памяти была успешна устранена, о чем свидетельствует вывод gprof:

```
==13788== Memcheck,a memory error detector
==13788== Copyright (C) 2002-2022,and GNU GPL'd,by Julian Seward et al.
==13788== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==13788== Command: ./app.out
```

```
==13788==
==13788==
==13788== HEAP SUMMARY:
==13788==     in use at exit: 0 bytes in 0 blocks
==13788==   total heap usage: 3,584 allocs,3,584 frees,374,768 bytes allocated
==13788==
==13788== All heap blocks were freed --no leaks are possible
==13788==
==13788== For lists of detected and suppressed errors,rerun with: -s
==13788== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3 Анализ покрытия кода с помощью gcov

Для оценки полноты тестирования был использован инструмент gcov, который показал следующие результаты покрытия исходного кода. Основной файл реализации (main.c) оказался покрыт на 62.61% (345 строк)

Непокрытые участки в основном связаны с операциями сохранения дерева в файл и загрузке из него, которые не тестировались.

4 Выводы

1. Новая и старая версии программ не отличаются по скорости выполнения или даже ухудшает его.

2. Использование Valgrind позволило эффективно выявить и устранить места утечек памяти.

3. Использование gprof позволило оценить соотношения затрат на работы различных функций в программе. В частности, что ввод и начальная обработка ключа занимают большую часть времени.

4. Низкое покрытие основного кода обусловлено отсутствием тестов на функции сохранения словаря в файл и загрузке из него. Для улучшения метрик можно проработать сценарии, вызывающие эти функции.

5 Заключение

В ходе лабораторной работы были успешно исследованы характеристики производительности реализации словаря на красно-черном дереве. Узких мест в программе выявлено не было, однако была обнаружена и устранена не очевидная утечка памяти. Полученный опыт демонстрирует:

1. Важность повышенной внимательности при работе с динамической памятью в языке C

2. Эффективность инструментов стандартной библиотеки

3. Сам формат входных данных может оказывать весомый вклад в скорость обработки. В данном случае, большуя часть времени происходит считывание ключей и приведение их к нижнему регистру.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Красно-черные деревья — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Красно-чёрное_дерево` (дата обращения: 27.04.2025).

[3] *Документация на Valgrind*
URL: `https://valgrind.org/docs/manual/quick-start.html` (дата обращения: 28.04.2025).