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. К. Носов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

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: 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).

Ход выполнения

Для выполнения лабораторной работы я использовал ноутбук операционной системой Ubuntu. Для выполнения анализа использовал программы Valgrind и gproof.

1 Valgrind

Valgrind - инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования. В ходе выполнения лабораторной работы утилита будет использована исключительно для отладки использования памяти.

==113422== Memcheck, a memory error detector
==113422== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==113422== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==113422== Command: ./lab2
==113422==
+ a 1234
OK
+ b 214324324
OK
+ c 23532523
OK
- a
OK
- b
OK
c
OK: 23532523
a
NoSuchWord
! Save asd
OK
- c
OK
c
NoSuchWord
! Load asd
OK
c
OK: 23532523
==113422== ==113422== HEAP SUMMARY:
==113422== in use at exit: 0 bytes in 0 blocks
==113422== total heap usage: 13 allocs, 13 frees, 92,441 bytes allocated
==113422==
==113422== All heap blocks were freed – no leaks are possible

```
==113422==
==113422== For lists of detected and suppressed errors, rerun with: -s
==113422== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Как видим, Valgrind не обнаружил утечек памяти. Можно спать спокойно.

2 gprof

gprof - это инструмент для профилирования программы. С помощью него мы можем отследить сколько времени и на каком участке кода выполнялась программа, тем самым выявляя слабые участки. Возьмем достаточно большой тест и применим утилиту gprof.

% time	cumulative seconds	self seconds	calls	self Ts/calls	total Ts/calls	name
0.00	0.00	0.00	265	0.00	0.00	bool std::operator< <char, std::char_traits<char
0.00	0.00	0.00	174	0.00	0.00	std::char_traits<char>::compare(char const*, ch
0.00	0.00	0.00	174	0.00	0.00	__gnu_cxx::__enable_if<std::__is_char<char
0.00	0.00	0.00	174	0.00	0.00	bool std::operator!=<char, std::char_traits<char
0.00	0.00	0.00	71	0.00	0.00	bool std::operator><char, std::char_traits<char
0.00	0.00	0.00	54	0.00	0.00	KV:: KV()
0.00	0.00	0.00	40	0.00	0.00	bool std::operator==<char, std::char_traits<char
0.00	0.00	0.00	36	0.00	0.00	KV::KV()
0.00	0.00	0.00	29	0.00	0.00	TAVLTree::Find(std::__cxx11::basic_string<char
0.00	0.00	0.00	23	0.00	0.00	KV::operator=(KV const&)
0.00	0.00	0.00	20	0.00	0.00	TAVLTree::BalanceF(TNode*)
0.00	0.00	0.00	20	0.00	0.00	TAVLTree::Balance(TNode*)
0.00	0.00	0.00	19	0.00	0.00	TAVLTree::LeftRotate(TNode*)
0.00	0.00	0.00	18	0.00	0.00	TAVLTree::RightRotate(TNode*)
0.00	0.00	0.00	18	0.00	0.00	AVLTree::Insert(KV)
0.00	0.00	0.00	18	0.00	0.00	KV::KV(KV const&)
0.00	0.00	0.00	18	0.00	0.00	TNode::TNode()
0.00	0.00	0.00	18	0.00	0.00	TNode:: TNode()
0.00	0.00	0.00	11	0.00	0.00	TAVLTree::Remove(TNode*)
0.00	0.00	0.00	2	0.00	0.00	__static_initialization_and_destruction_0(int,
0.00	0.00	0.00	1	0.00	0.00	TAVLTree::Clear(TNode*)
0.00	0.00	0.00	1	0.00	0.00	TAVLTree::TAVLTree()
0.00	0.00	0.00	1	0.00	0.00	TAVLTree:: TAVLTree()

Итак:
Наиболее часто вызываемые функции:

* ‘__gnu_cxx::__enable_if<std::__is_char<char>::__value, bool>::__type std::operator==‘ для сравнения строк на равенство
* ‘std::operator!=‘ для сравнения строк на неравенство
* ‘std::operator>‘ для сравнения строк

* ‘std::operator<‘ для сравнения строк типа ‘std::string‘

Менее часто вызываемые функции:

* Деструктор класса ‘KV‘ (уничтожение объектов класса ‘KV‘)
* Конструктор класса ‘KV‘ (создание объектов класса ‘KV‘)
* ‘TAVLTree::Find‘ для поиска узла в двоичном дереве поиска
* ‘KV::operator=‘ для присваивания значений объектам класса ‘KV‘
* ‘TAVLTree::BalanceF‘ для нахождения баланс-фактора
* Функции для перебалансировки дерева после вставки и удаления узлов

Функции, вызываемые редко:

* Конструктор и деструктор класса ‘TNode‘ (для узлов двоичного дерева поиска)
* Функции для статической инициализации и уничтожения в C++
* Удаление всего дерева

3 Выводы

Выполняя данную лабораторную работу, я познакомился с очень полезными инструментами и приобрел большой практический опыт. В дальнейшем я буду стараться использовать Valgrind и gprof как можно чаще.

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] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008