

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: М. В. Леухин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-20
Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №3

Задача: Необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочётов, требуется их исправить.

Для выполнения задачи использовать средства gprof и valgrind.

1 Описание

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. Общих выводов о выполнении лабораторной работы, полученном опыте.

2 Дневник выполнения работы

Основные этапы создания программы:

1. Реализация необходимых алгоритмов.
2. Выявление логических ошибок в коде программы.
3. Выявление утечек памяти.
4. Выявление неэффективно работающих участков кода.

3 Используемые инструменты

1 Valgrind

Как сказано в [1]: «Набор инструментов Valgrind предоставляет ряд инструментов отладки и профилирования, которые помогут вам сделать ваши программы быстрее и корректнее. Самый популярный из этих инструментов называется Memcheck. Он может обнаруживать многие ошибки, связанные с памятью, которые часто встречаются в программах на C и C++ и которые могут привести к сбоям и непредсказуемому поведению.»

Для использования утилиты необходимо компилировать программу с ключом -g, так как Valgrind использует для работы отладочную информацию.

```
==13907== Memcheck, a memory error detector
==13907== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13907== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
```

```

==13907== Command: ./lab3
==13907==
==13907==
==13907== HEAP SUMMARY:
==13907==      in use at exit: 122,880 bytes in 6 blocks
==13907==    total heap usage: 23,008 allocs,23,002 frees,1,851,656 bytes allocated
==13907==
==13907== LEAK SUMMARY:
==13907==    definitely lost: 0 bytes in 0 blocks
==13907==    indirectly lost: 0 bytes in 0 blocks
==13907==    possibly lost: 0 bytes in 0 blocks
==13907==    still reachable: 122,880 bytes in 6 blocks
==13907==           suppressed: 0 bytes in 0 blocks
==13907== Rerun with --leak-check=full to see details of leaked memory
==13907==
==13907== For lists of detected and suppressed errors, rerun with: -s
==13907== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

В результате использования утилиты Valgrind утечки памяти не выявлены — программа работает с памятью корректно.

2 Gprof

Gprof — инструмент профилирования, благодаря которому можно узнать, на что именно тратится больше всего времени при выполнении программы и какие её части стоит оптимизировать. Будем использовать gprof в связке с утилитой gprof2dot, которая по результаты работы gprof строит удобный для анализа граф.

- Вызов конструктора строки — 11,76%.
- Оператор сравнения строк — 5,88%.
- Получение длины строки — 5,88%.

Помимо строк, 9,46% уходит на работу с pair.

Также стоит упомянуть, что и функции, вызываемые в процессе ребалансировки, не бесплатные: на вызов функции `TNode::Rebalance` приходится 9,63% времени из 32,14% времени работы функции `TNode::Insert`. Аналогично на `TNode::RemovalRebalance` приходится 4,92% из 12,45%.

4 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я познакомился с различными инструментами отладки и профилирования программы, благодаря которым можно выявлять неявные недостатки, связанные с работой с памятью или плохооптимизированным кодом. В ходе анализа производительности программы я наглядно увидел, что строки в C++ это хоть и удобно, но «дорого».

Список литературы

[1] *Valgrind Documentation*

URL: <https://valgrind.org/docs/> (дата обращения: 23.05.2022).