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: Е.А. Кондратьев

Преподаватель: А. А. Кухтичев

Группа: М8О-206Б

Дата:

Оценка:

Подпись:

Москва, 2020

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 недочетов, требуется их исправить.

Минимальный набор используемых средств должен содержать утилиту gprof и библиотеку dmalloc, однако их можно заменять на любые другие аналогичные или более известные утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov)

Вариант дерева: AVL-дерево.

1 Дневник отладки

1. Напишем файл benchmark.cpp в котором замерим время работы std::map и AVL-дерева при помощи библиотеки <chrono> (пункт: скорость выполнения).

2. При помощи valgrind оценим утечки и расход памяти для std::map и собственного AVL-дерева (пункт: потребление оперативной памяти).

3. В собственной реализации были найдены некоторые незначительные утечки памяти: still reachable: 122,880 bytes in 6 blocks, они возникают из-за использования оптимизаций, таких как std::ios::sync_with_stdio(false) и std::cin.tie(nullptr).

2 Скорость выполнения

Тест представляет из себя следующее: создаем объекты std::map и наш avl. Вставляем в оба объекта по 1 млн. элементов с ключом=значению в диапазоне от 0 до 999999. Измеряем время работы для std::map и avl. Далее 1 млн. раз ищем элемент с ключом=значению=999999 и замеряем время для std::map и avl. Последний тест - 1 млн. раз удаляем значение(от 999999 до 0) из std::map и avl, замеряем время.

```
du$ ./benchmark
Insert map time: 6.99499 seconds
Insert avl time: 12.1846 seconds
Find map time: 7.5032 seconds
Find avl time: 3.30145 seconds
Delete map time: 10.0912 seconds
Delete avl time: 9.73254 seconds
```

Как видно, что удаление в avl работает совсем чуть-чуть быстрее чем в std::map, вставка в avl работает значительно более медленно, чем в std::map, поиск в avl работает значительно быстрее чем в std::map

3 Потребление оперативной памяти

Тест представляет из себя следующее: создаем объекты std::map и наш avl. Вставляем в оба объекта по 1 млн. элементов с ключом=значению в диапазоне от 0 до 999999, 1 млн. раз ищем элемент с ключом=значению=999999, 1 млн. раз удаляем значение(от 999999 до 0) из std::map и avl, запускаем с использованием valgrind сначала для avl потом для std::map и смотрим на использование и утечки памяти. Файл memory_test.txt представляет из себя 3 млн. команд, 1 млн. вида «+ i i», 1 млн. вида «i» и 1 млн. вида «- i». Исполняемый файл «map» получается при компиляции файла map.cpp в котором выполняется по 1 млн. операций вставки, поиска и удаления в std::map.

```
du@Du:~/solution/solution$ valgrind ./solution <memory_test.txt
 ==2127== Memcheck,a memory error detector
==2127== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.

==2127== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2127== Command: ./solution
==2127==
==2127== HEAP SUMMARY:
==2127==         in use at exit: 122,880 bytes in 6 blocks
==2127==         total heap usage: 56,894,144 allocs,56,894,138 frees,422,797,359 bytes
allocated
==2127==
==2127== LEAK SUMMARY:
==2127==         definitely lost: 0 bytes in 0 blocks
==2127==         indirectly lost: 0 bytes in 0 blocks
==2127==         possibly lost: 0 bytes in 0 blocks
==2127==         still reachable: 122,880 bytes in 6 blocks
==2127==         suppressed: 0 bytes in 0 blocks
==2127==
==2127==
==2127== Rerun with --leak-check=full to see details of leaked memory
==2127==
==2127== For counts of detected and suppressed errors,rerun with: -v
==2127== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

du@D:~/solution/solution$ valgrind ./map
==2219== Memcheck,a memory error detector
==2219== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==2219== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2219== Command: ./map
==2219==
==2219==
==2219== HEAP SUMMARY:
==2219==         in use at exit: 0 bytes in 0 blocks
==2219==         total heap usage: 1,000,001 allocs,1,000,001 frees,72,072,704 bytes
allocated
==2219==
==2219== All heap blocks were freed --no leaks are possible
==2219==
==2219== For counts of detected and suppressed errors,rerun with: -v
==2219== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Как видно из тестов, в собственной реализации AVL-дерева есть незначительные утечки памяти, значительно больше количество аллокаций, очищений памяти, а объем аллоцированной памяти примерно в 6 раз больше чем в std::map. Как можно видеть число аллокаций и очисток в std::map совпадает, т.к. нет ни одной утечки памяти, в отличие от собственного AVL-дерева, где число аллокаций больше чем число очищений, разница равняется 6 блокам, которые отображаются в still reachable.

5 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я познакомился с очень полезными инструментами. Valgrind позволяет оценивать работу с памятью в программе (искать утечки памяти, смотреть использование памяти).