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: Н. П. Ежов
Преподаватель: Н. С. Капралов
Группа: М8О-204Б
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 недочётов, 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 структуры данных: `Красно-чёрное дерево.`

1 Дневник отладки

1. Напишем файл benchmark.cpp, в котором замерим время работы std::map и КЧ-Дерева при помощи библиотеки <chrono> (пункт: скорость выполнения).
2. При помощи valgrind оценим утечки и расход памяти для std::map и собственного КЧ-Дерева (пункт: потребление оперативной памяти).
3. В собственной реализации не было найдено никаких утечек памяти: All heap blocks were freed – no leaks are possible
4. При помощи gprof выяснили, что наибольшее время занимают операции вставки и удаления в дерево.

2 Скорость выполнения

Тест представляет из себя следующее: при помощи генератора тестов был создан файл с 10^7 строками, каждая из которых является командой к дереву (добавление, удаление или поиск). Измеряем время на выполнение операций для std::map и КЧ-Дерева.

```
(base) nikita@nikita-desktop:~/CLionProjects/Diskran/lab2$ ./benchmark.o <tests/07.t
==============START============
INSERT std::map time: 5286 ms
INSERT rb tree time: 38894 ms
==============================
DELETE std::map time: 4438 ms
DELETE rb tree time: 19721 ms
==============================
SEARCH std::map time: 1825 ms
SEARCH rb tree time: 15717 ms
==============END==============
```

Как видно, все операции из std::map выполняются быстрее, чем для собственной реализации КЧ-Дерева. Скорее всего, это связано с тем, что работу стандартной библиотеки C++ постоянно оптимизируют, из-за чего собственные реализации функций выполняются относительно медленнее.

3 gprof

Скомпилируем программу с флагом -pg. Запустим нашу программу на тесте с 10^7 входных строк. После этого в текущей директории создался файл gmoun.out, чтобы посмотреть результат работы профилирования выполним команду gprof solution gmon.out.

```
%   cumulative   self              self     total
time   seconds   seconds    calls  ns/call  ns/call  name
68.05      1.74     1.74
NMyStd::TRBTree::Insert(NMyStd::TItem const&,NMyStd::TRBNode*)
24.64      2.37     0.63
NMyStd::TRBTree::Remove(char const*)
3.13      2.45     0.08  1998364    40.08    60.12
NMyStd::TRBTree::Remove(NMyStd::TRBNode*)
1.96      2.50     0.05
NMyStd::TRBTree::Insert(NMyStd::TItem const&)
1.56      2.54     0.04  1161306    34.49    34.49
NMyStd::TRBTree::RemoveFixUp(NMyStd::TRBNode*,NMyStd::TRBNode*)
0.78      2.56     0.02   853961    23.45    23.45
NMyStd::TRBTree::InsertFixUp(NMyStd::TRBNode*)
0.00      2.56     0.00        1     0.00     0.00
_GLOBAL__sub_I__ZN6NMyStd7TRBTree6SearchEPcRNS_5ITemE
```

По данной таблице можно понять, что больше всего времени уходит на вставку и удаление из дерева.
Дальше выводится граф вызовов (тут я вывел его в укороченном виде), в каждой строке графа указываются функция, в предыдущих строках выводиятся функции вызываемые данной функцией. Таким образом можно определить, где происходят вызовы наиболее долгих функций, ведь сама функция может выполняться быстро, но функции вызываемые ей могут тратить много времени.

```
index % time    self  children    called     name
<spontaneous>
[1]     68.8    1.74    0.02
NMyStd::TRBTree::Insert(NMyStd::TItem const&,NMyStd::TRBNode*) [1]
0.02    0.00  853961/853961
NMyStd::TRBTree::InsertFixUp(NMyStd::TRBNode*) [6]
-----------------------------------------------
<spontaneous>
[2]     29.3    0.63    0.12
NMyStd::TRBTree::Remove(char const*) [2]
```

```
0.08    0.04 1998364/1998364
NMyStd::TRBTree::Remove(NMyStd::TRBNode*) [3]
-----------------------------------------------
0.08    0.04 1998364/1998364
NMyStd::TRBTree::Remove(char const*) [2]
[3]      4.7    0.08    0.04 1998364
NMyStd::TRBTree::Remove(NMyStd::TRBNode*) [3]
0.04    0.00 1161306/1161306
NMyStd::TRBTree::RemoveFixUp(NMyStd::TRBNode*,NMyStd::TRBNode*) [5]
-----------------------------------------------
<spontaneous>
[4]      2.0    0.05    0.00
NMyStd::TRBTree::Insert(NMyStd::TItem const&) [4]
-----------------------------------------------
0.04    0.00 1161306/1161306
NMyStd::TRBTree::Remove(NMyStd::TRBNode*) [3]
[5]      1.6    0.04    0.00 1161306
NMyStd::TRBTree::RemoveFixUp(NMyStd::TRBNode*,NMyStd::TRBNode*) [5]
-----------------------------------------------
0.02    0.00  853961/853961
NMyStd::TRBTree::Insert(NMyStd::TItem const&,NMyStd::TRBNode*) [1]
[6]      0.8    0.02    0.00  853961
NMyStd::TRBTree::InsertFixUp(NMyStd::TRBNode*) [6]
```

4 Потребление оперативной памяти

Тест представляет из себя следующее: запускаем исполняемый файл solution на тесте 10^7 с использованием valgrind

```
(base) nikita@nikita-desktop:~/CLionProjects/Diskran/lab2$ valgrind --tool=memcheck
./solution <tests/07.t >res.txt
==5308== Memcheck,a memory error detector
==5308== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==5308== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==5308== Command: ./solution
==5308==
==5308==
==5308== HEAP SUMMARY:
==5308==     in use at exit: 0 bytes in 0 blocks
==5308==   total heap usage: 2,008,766 allocs,2,008,766 frees,610,761,250 bytes
```

```
allocated
==5308==
==5308== All heap blocks were freed --no leaks are possible
==5308==
==5308== For lists of detected and suppressed errors,rerun with: -s
==5308== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Как видно из теста, в собственной реализации КЧ-Дерева нет никаких утечек памяти.

5 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я познакомился с очень полезными инструментами.
1. Valgrind позволяет оценивать работу с памятью в программе (искать утечки памяти, смотреть использование памяти, а некоторые инструменты valgrind подрузамевают профилирование кода)
2. Gprof позволяет оценить производительность работы программы
При помощи данных инструментов удобно исправлять и улучшать свой код.