Лабораторная работа №3 по курсу дискретного анализа: сбалансированные деревья

Выполнил студент группы М8О-207Б-20 Мерц Савелий Павлович

Условие

Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти

Основная информация

gprof

Утилита gprof позволяет измерить время работы всех функций, методов и операторов программы, количество их вызовов и долю от общего времени работы программы в процентах.

Сначала скомпилируем программу с флагом -pg. Затем запустим программу и обнаружим что создался файл gmon.out. Конвертируем его в txt с помощью команды :

gprof /*exec_file*/ gmon.out > profile -data.txt

Получим таблицу в файле и много другой информации.

Результаты для 50000 пар

% time	self seconds	calls	name
14.29	0.02	304780	NodeIsLeaf()
10.71	0.06	4702406	vector::size()
7.14	0.01	1016811	ItemInNode()

Много времени заняли методы std::vector и прочие стандартные. Размер вектора можно хранить в ноде для ускорения программы как и данные является ли нода листом.

valgrind

Valgrind один из самых распространенных инструментов для проверки программы на утечки памяти. Для проверки выполним команду:

valgrind .//*exec_file*/

Результаты для 50000 пар

==8511== Process terminating with default action of signal 27 (SIGPROF)

==8511== at 0x4CAA12A: __open_nocancel (in /usr/lib/libc.so.6)

==8511== by 0x4CB803F: write_gmon (in /usr/lib/libc.so.6)

==8511== by 0x4CB883E: _mcleanup (in /usr/lib/libc.so.6)

```
==8511==    by 0x4BE80F6: __cxa_finalize (in /usr/lib/libc.so.6)
==8511==    by 0x10A2C7: ??? (in /home/sava/Документы/GitHub/DA/lab_2/benchmark/lab)
==8511==    by 0x4005C8D: _dl_fini (in /usr/lib/ld-linux-x86-64.so.2)
==8511==    by 0x4BE7C04: __run_exit_handlers (in /usr/lib/libc.so.6)
==8511==    by 0x4BE7D7F: exit (in /usr/lib/libc.so.6)
==8511==    by 0x4BD0316: (below main) (in /usr/lib/libc.so.6)
==8511==
==8511== HEAP SUMMARY:
==8511==     in use at exit: 9,814,963 bytes in 109,837 blocks
==8511==   total heap usage: 2,895,012 allocs, 2,785,175 frees, 271,414,297 bytes allocated
==8511==
==8511== LEAK SUMMARY:
==8511==    definitely lost: 2,259,008 bytes in 56,475 blocks
==8511==    indirectly lost: 7,275,315 bytes in 53,353 blocks
==8511==      possibly lost: 0 bytes in 0 blocks
==8511==    still reachable: 280,640 bytes in 9 blocks
==8511==         suppressed: 0 bytes in 0 blocks
==8511== Rerun with --leak-check=full to see details of leaked memory
==8511==
==8511== For lists of detected and suppressed errors, rerun with: -s
==8511== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Я не ожидал таких плохих результатов. Изучая причины утечек обнаружил, что отключение синхронизации iostreams с stdio и cin с cout порождает still reachable: 280,640 bytes in 9 blocks. Остальные утечки вызваны некорректным очищением памяти при удалении, я сначала занулял ссылки а потом их очищал дабы избежать сегфолда на некоторых тестах.

Выводы

Проделав работу познакомился с очень полезными инструментами. Valgrind я знал и раньше, но закрепил навыки работы с ним, а вот gprof стал для меня находкой. Теперь я могу прикидывать какие методы выгоднее заменить на поля в классе и наоборот для достижения большей эффективности программы. Надо будет устранить утечки, а то вся работа насмарку.