Лабораторная работа № 3 по курсу дискретного анализа: Сбаласированные деревья

Выполнил студент группы М8О-208Б-20 МАИ *Попов Матвей*.

Условие

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

gprof

Основная информация

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

Команды для работы с утилитой

Сначала скомпилируем исходную программу с ключом $-pg$:

```
g++ lab.cpp -pg -o lab
```

Затем запустим программу, передав ей на ввод файл *test.txt*, в котором содержится по 5000 команд на вставку, поиск и удаление:

```
./lab <test.txt >out.txt
```

Выполнив эту команду, заметим, что кроме файла *out.txt*, в котором содержатся результаты выполнения команд из *test.txt*, появился файл *gmon.out*, в котором содержится вся информация, предоставляемая утилитой gprof. Чтобы получить текстовый файл, выполним следующую команду:

```
gprof lab gmon.out > profile-data.txt
```

Таким образом, выполнив 3 простые команды, получили текстовый файл с подробной информацией о времени работы и вызовах всех функций и операторов, которые использовались в программе.

Результат работы утилиты

Ниже приведена таблица, в которую перенесены данные из файла $profile - data.txt$, полученного с помощью утилиты gprof.

% time	self seconds	calls	name
93.33	2.09	3664224	IPair::TPair::operator=
1.34	0.03	33982	Clear(char*)
1.34	0.03	25233	IPair::TPair::TPair(IPair::TPair const&)
0.89	0.02	330720	IPair::operator<
0.89	0.02	20000	ToLower(char*, IPair::TPair&)
0.45	0.01	54943	IBTree::BinarySearch
0.45	0.01	42109	IPair::operator==

Все остальные функции, по данным результатам измерений утилиты gprof, работали примерно 0 секунд, поэтому в таблицу внесены не были. Из полученных данных следует, что большая часть времени работы программы тратится на операцию копирования пары «ключ-значение», это может быть связано с использованием объектов класса *vector* в узлах дерева.

valgrind

Valgrind является самым распространённым инструментом для отслеживания утечек памяти и других ошибок, связанных с памятью. Чтобы проверить программу *lab* на проблемы с памятью, выполним следующую команду:

```
valgrind ./lab <test.txt >out.txt
```

В результате выполнения этой команды получаем следующее сообщение:

```
==8733== Process terminating with default action of signal 2 (SIGINT)
==8733==    at 0x4B66075: write (write.c:26)
==8733==    by 0x492777D: std::__basic_file<char>
::xsputn(char const*, long) (in /usr/lib
/x86_64-linux-gnu/libstdc++.so.6.0.28)
==8733==    by 0x4966EA0: std::basic_filebuf<char,
std::char_traits<char> >::_M_convert_to_external
(char*, long) (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==8733==    by 0x49672FB: std::basic_filebuf<char,
 std::char_traits<char> >::overflow(int) (in /usr
 /lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==8733==    by 0x496505C: std::basic_filebuf<char,
 std::char_traits<char> >::sync() (in /usr/lib
 /x86_64-linux-gnu/libstdc++.so.6.0.28)
==8733==    by 0x498D7A2: std::ostream::flush()
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==8733==    by 0x10CF92: main (in /mnt/c/Home/Prog/DA/lab2/lab)
==8733==
==8733== HEAP SUMMARY:
==8733==     in use at exit: 266,288 bytes in 9 blocks
==8733==   total heap usage: 27 allocs, 18 frees,
```

```
482,064 bytes allocated
==8733==
==8733== LEAK SUMMARY:
==8733==    definitely lost: 0 bytes in 0 blocks
==8733==    indirectly lost: 0 bytes in 0 blocks
==8733==      possibly lost: 0 bytes in 0 blocks
==8733==    still reachable: 266,288 bytes in 9 blocks
==8733==         suppressed: 0 bytes in 0 blocks
==8733== Rerun with --leak-check=full to see details of leaked memory
==8733==
==8733== For lists of detected and suppressed errors, rerun with: -s
==8733== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

С помощью Valgrind обнаружили несколько незначительных ошибок и неосвобождённую память после выполнения программы.

Выводы

Проделав лабораторную работу, я познакомился с полезной утилитой gprof, необходимой для измерения времени работы программы и отдельных её частей, закрепил навыки работы с утилитой valgrind, а также обнаружил неосвобождённую память в своей программе.