Лабораторная работа №3 по курсу дискрeтного анализа:

исследование качества программ.

Выполнил студент группы М80-208Б-20 Морозов Артем Борисович.

Условие

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.

# Дневник выполнения работы

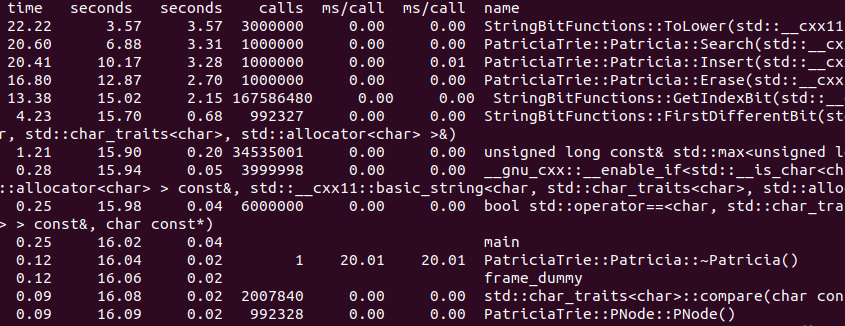
Для выполнения лабораторной работы я воспользовался двумя утилитами: **gprof** и **valgrind**.

Для начала я запущу свою программу с утилитой **gprof**. Утилита gprof хороша тем, что позволяет увидеть время работы всех функций, реализованных в программе, количество их вызовов и вычисляет процентное соотношение работы конкретной функции по сравнению с работой всей программы.

Скомпилируем при помощи команды **g++ task2.cpp -pg -o main**, где -pg – специальный флаг, позволяющий вывести необходимые данные в файл gmon.out.

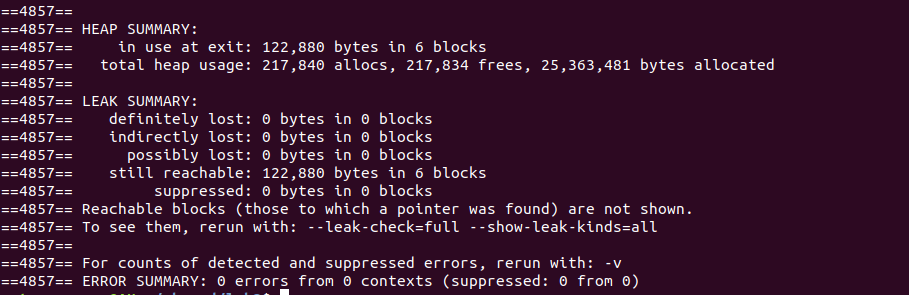
Предварительно сгенерируем файл **gprof\_test.txt**, состоящий из 1000000 команд на поиск, вставку и удаление. После – запустим нашу программу с этим файлом: **./main < gprof\_test.txt**  
После компиляции создался файл gmon.out, который является выводом нашей утилиты. Просмотрим его при помощи следующей команды: **gprof main.**

Результат вывода утилиты **gprof** следующий:

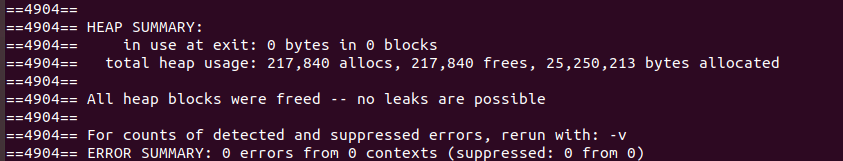


Остальные функции показывали 0.00 в графе time.

Как мы можем увидеть, на тесте в 1000000 строк для каждого метода, наибольшее время выполнения у всех трех основных операций с деревом: вставкой, поиском и удалением, а также у всех основных действий со строками: ToLower, GetIndexBit и FirstDifferentBit. Вероятно, это связано с тем, что на многие строки, подающиеся в дерево, могут быть и длиной в 256 символов, что довольно-таки много.

Давайте теперь запустим нашу программу через **valgrind** – одной из самых лучших утилит для поиска всевозможных утечек памяти в программе. Проверять будем со специальным флагом –leak-check=full:  


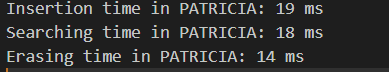
Во многих источниках пишут, что, если в definitely lost 0 bytes и ERROR SUMMARY: 0 errors, то утечек памяти нет. Параметр still reachable, по мнению многих, означает, что указатель не утерян и может быть освобожден, но на данный момент не освобожден. Если верить подобным людям, то, по идее, уже на данном этапе утечек в программе нет, однако давайте попробуем убрать строку still reachable и сделать количество frees равным количеству allocs.

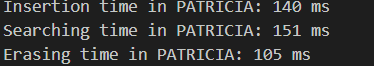
Немного поизучав различные источники, я пришел к выводу, что still reachable могут вызывать так называемые “строчки-ускорители”: **ios:: sync\_with\_stdio(false), cin.tie(0), cout.tie(0).** Закомментировав их, я полностью избавился от каких-либо дополнительных сообщений valgrind:  


Вывод о найденных недочетах

После использования двух утилит недочетов не было обнаружено. Единственное, что могло бы условно считаться недочетом – это вышеупомянутые блоки still reachable, однако это было исправлено при помощи комментирования “строк-ускорителей”. Ответ, почему эти строки влияют на неосвобождение некоторых блоков памяти, дает нам стандарт языка С++ - он просто говорит, что когда программа выходит из потока, сами объекты - и, следовательно, используемые ими буферы - не будут уничтожены.

# Сравнение работы исправленной программы с предыдущей версией

Так как в исправленной программе были закомментированы, а не исправлены, всего лишь 3 строчки исходной программы, связанные с ее ускорением, то очевидно, что старая версия будет работать гораздо быстрее новой. Однако давайте в этом убедимся точно на каком-нибудь тесте – скажем, в 5000 строк на удаление, вставку и добавление.  
  
Результат работы со “строками-ускорителями”:  


Результат работы без “строк-ускорителей”:

# Общие выводы о выполнении лабораторной работы

Лабораторная работа №3 по дискретному анализу помогла мне закрепить навыки работы с утилитой для нахождения утечек памяти valgrind, а также познакомила меня с очень интересной утилитой gprof, которая помогает пользователю увидеть время выполнения отдельных функций, что очень полезно для огромных программ. Помимо этого, я смог добиться полного очищения выделенной памяти.