Лабораторная работа № 3 по курсу дискретного анализа: Исследование качества программ

Выполнил студент группы 8О-207Б-20 МАИ *Чекменёв Вячеслав Алексеевич.*

Условие

Общая постановка задачи:

1. Исследовать программу на наличие утечек памяти с помощью утилиты valgrind.
2. Исследовать скорость выполнения программы с помощью утилиты gprof, выявить недочеты производительности.

Дневник выполнения работы

0.1 Valgrind – memory

Проанализируем нашу программу на наличие утечек памяти с помощью valgrind.
Запустим программу:

```
valgrind -tool=memcheck -leak-check=full -show-leak-kinds=all -leak-resolution=med
./main < test.txt
```

Возьмем тест на 10000 строк с функциями Save, Load, AddNode, DeleteNode, ReturnNode
Проанализируем полученный отчет:

```
==15100== LEAK SUMMARY:
==15100==    definitely lost: 0 bytes in 0 blocks
==15100==    indirectly lost: 0 bytes in 0 blocks
==15100==      possibly lost: 0 bytes in 0 blocks
==15100==    still reachable: 122,880 bytes in 6 blocks
==15100==         suppressed: 0 bytes in 0 blocks
==15100==
==15100== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Категория "still reachable"в отчете об утечке Valgrind говорит о том, что эти блоки не были освобождены, но они могли бы быть освобождены, потому что программа все еще отслеживала указатели на эти блоки памяти.

Эту ошибку можно исправить убрав три строчки кода:

```
std::ios::sync_with_stdio(false);
std::cin.tie(0);
std::cout.tie(0);
```

Теперь посмотрим, что нам выдает Valgrind:

```
==3703== HEAP SUMMARY:
==3703==     total heap usage: 23,235,515 allocs, 23,235,515 frees, 774,349,1
==3703== LEAK SUMMARY:
==3703==    definitely lost: 0 bytes in 0 blocks
==3703==    indirectly lost: 0 bytes in 0 blocks
==3703==      possibly lost: 0 bytes in 0 blocks
==3703==    still reachable: 0 bytes in 0 blocks
==3703==         suppressed: 0 bytes in 0 blocks
==3703==
==3703== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

0.1.1 gprof – time

Исследуем нашу программу с помощью gprof.
Для использования профилятора следует добавить флаг -pg в команды компиляции. После этого можно запускать программу обычным образом. Возьмем тест на 25000 строк с функциями Save, Load, AddNode, DeleteNode, ReturnNode
После завершения нашей программы будет сгенерирован файл отчета **gmon.out**, запустим утилиту gprof для получения анализа:

```
gprof main.o > gprof.txt
```

Проанализируем полученный отчет:

seconds-self	calls	name
2.83	156591265	CharToBinary
0.91	158215535	RetrieveBit
0.30	4459264	ReturnNodePrivate
0.00	6419	ReturnNode
0.21	4446338	AddNodePrivate
0.09	4446558	AddNode
0.00	6287	DeleteNode
0.04	2993	Load
0.01	3056	Save
0.03	3055	FillNodesVectorPrivate
0.00	3056	FillNodesVector
0.04	2993	ClearNode

Как мы видим, много времени тратится на функцию ReturnNodePrivate, это из-за того, что здесь при каждом рекурсивном вызове функции ReturnNodePrivate проверяется header на NULL, этого можно делать один раз.

```
if (header == NULL) {
    return NULL;
}
```

Также много времени тратится на функции CharToBinary, RetrieveBit, в них можно было использовать побитовые операции вместо циклов, однако я этого не сделал

Найденные недочеты

Проблема	Инструмент
Утечка памяти "still reachable"	valgrind

Выводы

После использования таких инструментов как gprof и valgrind стало ясно что они действительно способны указать на проблемные места в программе. С помощью gprof было найдено вычислено время выполнения каждой функции в программе. Инструмент valgrind помог найти утечки памяти в программе.