# 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: П. Ф. Гришин
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата:
Оценка:
Подпись:


**Москва, 2023**

# Лабораторная работа №3

# 1 Описание

Результатом лабораторной работы является отчёт, состоящий из:

1. Дневника выполнения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы.
2. Выводов о найденных недочётах.
3. Общих выводов о выполнении лабораторной работы, полученном опыте.

# 2 Дневник выпонления работы

Основные этапы создания программы:

1. Реализация необходимых алгоритмов.
2. Выявление логических ошибок в коде программы.
3. Выявление утечек памяти.
4. Выявление неэффективно работающих участков кода.

# 3 Используемые инструменты

## 1 Valgrind

Как сказано в [1]: «Набор инструментов Valgrind предоставляет ряд инструментов отладки и профилирования, которые помогут вам сделать ваши программы быстрее и корректнее. Самый популярный из этих инструментов называется Memcheck. Он может обнаруживать многие ошибки, связанные с памятью, которые часто встречаются в программах на C и C++ и которые могут привести к сбоям и непредсказуемому поведению.»

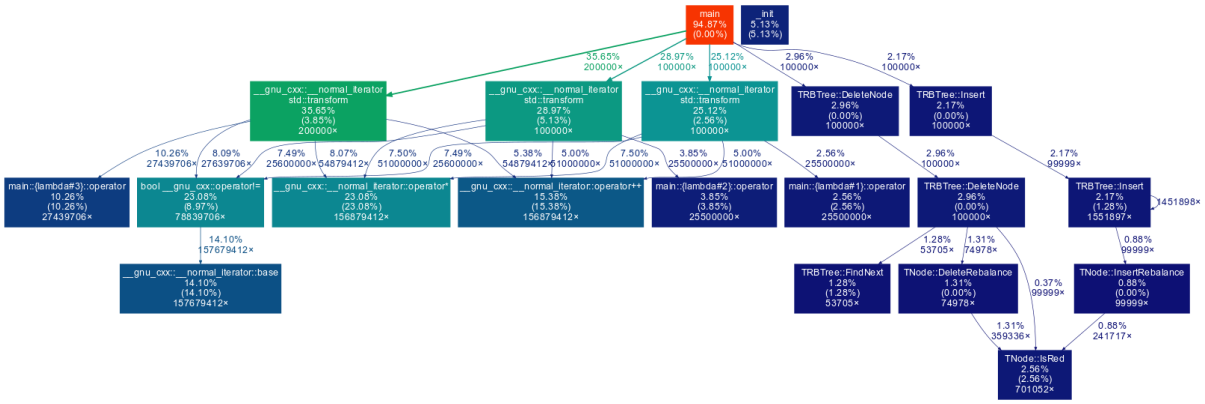
Для использования утилиты необходимо компилировать программу с ключом -g, так как Valgrind использует для работы отладочную информацию.

```
==36792== Memcheck,a memory error detector
==36792== Copyright (C) 2002-2022,and GNU GPL'd,by Julian Seward et al.
==36792== Using Valgrind-3.20.0-5147d671e4-20221024 and LibVEX; rerun with
-h for copyright info
==36792== Command: ./a.out
==36792==
==36792==
==36792== HEAP SUMMARY:
==36792==     in use at exit: 195,592 bytes in 8 blocks
==36792==   total heap usage: 8 allocs,0 frees,195,592 bytes allocated
==36792==
==36792== Searching for pointers to 8 not-freed blocks
==36792== Checked 149,192 bytes
==36792==
==36792== LEAK SUMMARY:
==36792==    definitely lost: 0 bytes in 0 blocks
==36792==    indirectly lost: 0 bytes in 0 blocks
==36792==      possibly lost: 0 bytes in 0 blocks
==36792==    still reachable: 195,592 bytes in 8 blocks
==36792==         suppressed: 0 bytes in 0 blocks
==36792== Rerun with --leak-check=full to see details of leaked memory
==36792==
==36792== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

В результате использования утилиты Valgrind утечки памяти не выявлены — программа работает с памятью корректно.

## 2 Gprof

Gprof — инструмент профилирования, благодаря которому можно узнать, на что именно тратится большего всего времени при выполнении программы и какие её части стоит оптимизировать. Будем использовать gprof в связке с утилитой gprof2dot, которая по результаты работы gprof строит удобный для анализа граф.




Как видно из графа, основное время выполнения программы приходится на __gnu_cxx. Возможно это связано с тем, что компаратор долго сравнивает строки из-за чего происходит такое большое количество обращений к функциям библиотеки STL и работа со строками дорошо обходится программе. Если же брать только реализацию дерева, то большая часть приходится на удаление TRBTREE::DeleteNode(2,96), если сравнивать с функцией вставки, то помимо функции TNode::IsRed, она обращается к функции TRBTree::FindNext. Хотя нагрузка со стороны удаления вершины и вставки довольно не велика.

# 4 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я познакомился с различными инструментами отладки и профилирования программы, благодаря которым можно выявлять неявные недостатки, связанные с работой с памятью или плохооптимизированным кодом. В ходе анализа производительности программы я наглядно увидел, что строки в C++ это хоть и удобно, но «дорого».

# Список литературы

[1] Valgrind Documentation.
*URL: https://valgrind.org/docs/ (дата обращения: 15.05.2023).*