



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по рубежному контролю №1 по курсу «Анализ алгоритмов»

Вариант Модифицировать алгоритм исправления орфографических ошибок в тексте

Студент Пискунов П.

Группа ИУ7-56Б

Преподаватель Волкова Л.Л., Строганов Д.В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Аналитическая часть	4
1.1 Многопоточность	4
1.2 Исправление орфографических ошибок в тексте	5
2 Конструкторская часть	6
2.1 Требования к программному обеспечению	6
2.2 Разработка алгоритма поиска расстояния Левенштейна	6
2.3 Разработка модифицированного алгоритма поиска расстояния Левенштейна	8
2.4 Разработка многопоточного алгоритма	9
3 Технологическая часть	10
3.1 Выбор средств реализации	10
3.2 Реализация алгоритмов	10
4 Исследовательская часть	15
4.1 Интерфейс приложения	15
4.2 Технические характеристики	15
4.3 Время выполнения реализаций алгоритмов	15
ЗАКЛЮЧЕНИЕ	18
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	20

ВВЕДЕНИЕ

С появлением новых вычислительных систем программисты стали сталкиваться с необходимостью проведения одновременной обработки данных для улучшения отзывчивости системы, ускорения выполнения вычислений и более эффективного использования вычислительных ресурсов. Развитие процессоров позволило использовать один процессор для выполнения нескольких параллельных операций, что привело к появлению термина «Многопоточность».

Целью данного рубежного контроля является исследование обычный и модифицированный алгоритм исправления орфографических ошибок в тексте (алгоритм поиска расстояния Левенштейна).

Для достижения цели, требуется выполнить следующие задачи:

- 1) проанализировать параллельный вариант алгоритма исправления орфографических ошибок в тексте;
- 2) определить средства программной реализации выбранного алгоритма;
- 3) реализовать алгоритм поиска расстояния Левенштейна и модифицированный алгоритм поиска расстояния Левенштейна;
- 4) провести сравнительный анализ по времени реализованного алгоритма;
- 5) подготовить отчет о рубежном контроле.

1 Аналитическая часть

1.1 Многопоточность

Многопоточность представляет собой способность центрального процессора (ЦП) одновременно обрабатывать несколько потоков, используя ресурсы одного процессора [1]. Поток – это последовательность инструкций, которые могут выполняться параллельно с другими потоками в рамках одного процесса, из которого они произошли.

Процесс представляет собой исполняемую программу во время ее выполнения [2]. При запуске программы создается процесс. Процесс может включать в себя один или несколько потоков. Поток является частью процесса, ответственной за выполнение конкретных задач в рамках приложения. Завершение процесса происходит, когда все его потоки завершают свою работу. В операционной системе каждый поток представляет собой задачу, которую процессор должен выполнить. Современные процессоры могут обрабатывать несколько задач на одном ядре, создавая виртуальные ядра, или иметь несколько физических ядер, и такие процессоры называются многоядерными.

При разработке программы, которая использует несколько потоков, важно учитывать, что если потоки запускаются последовательно и управление передается каждому из них поочередно, то полный потенциал многозадачности не будет реализован. Это связано с тем, что выигрыш от параллельного выполнения задач не будет полностью использован. Эффективное использование многозадачности достигается путем создания потоков для независимых по данным задач и их параллельного выполнения, что позволяет сократить общее время выполнения процесса.

При работе с потоками возникает проблема общего доступа к данным. Одним из основных ограничений является запрет на одновременную запись в одну и ту же ячейку памяти из двух или более потоков. Поэтому требуется использовать механизм синхронизации доступа к данным, который называется мьютексом. Этот мьютекс может быть захвачен одним потоком для монопольного доступа к данным или освобожден для доступа других потоков. Например, если два потока одновременно пытаются захватить мьютекс, толь-

ко одному из них это удастся, а второй поток будет ожидать, пока мьютекс освободится.

Совокупность инструкций, которые выполняются между захватом и освобождением мьютекса, называется критической секцией. Поскольку в период захвата мьютекса остальные потоки, которым требуется доступ к тем же данным для выполнения критической секции, ожидают освобождения мьютекса, необходимо стараться минимизировать объем операций в критической секции.

1.2 Исправление орфографических ошибок в тексте

Исправление орфографических ошибок в тексте представляет собой важный этап редактирования, направленный на улучшение грамматической корректности и читаемости текстовой информации [3].

Орфографические ошибки могут варьироваться от неверного написания отдельных слов до некорректного использования знаков препинания. Процесс исправления таких ошибок способствует повышению качества текста и обеспечивает более точное и понятное восприятие сообщаемой информации.

В данной лабораторной работе проводится распараллеливание алгоритма исправления орфографических ошибок в тексте. Для этого весь текст поровну распределяется между всеми потоками.

В качестве одного из аргументов каждый поток получает выделенный для него строку массива слов.

Вывод

В данном разделе была представлена информация о многопоточности и исследуемом алгоритме.

2 Конструкторская часть

В этом разделе будут представлены требования к программному обеспечению (ПО) и схема алгоритмов.

2.1 Требования к программному обеспечению

Программе передаются текст с орфографическими ошибками или без и словарь в качестве входных данных, а на выход получается текст без орфографических ошибок. Кроме того, необходимо сообщить пользователю затраченное каждым алгоритмом процессорное время.

В создаваемом приложении пользователю должен быть доступен выбор желаемого алгоритма.

2.2 Разработка алгоритма поиска расстояния Левенштейна

На рисунке 2.1 представлен алгоритм поиска расстояния Левенштейна.

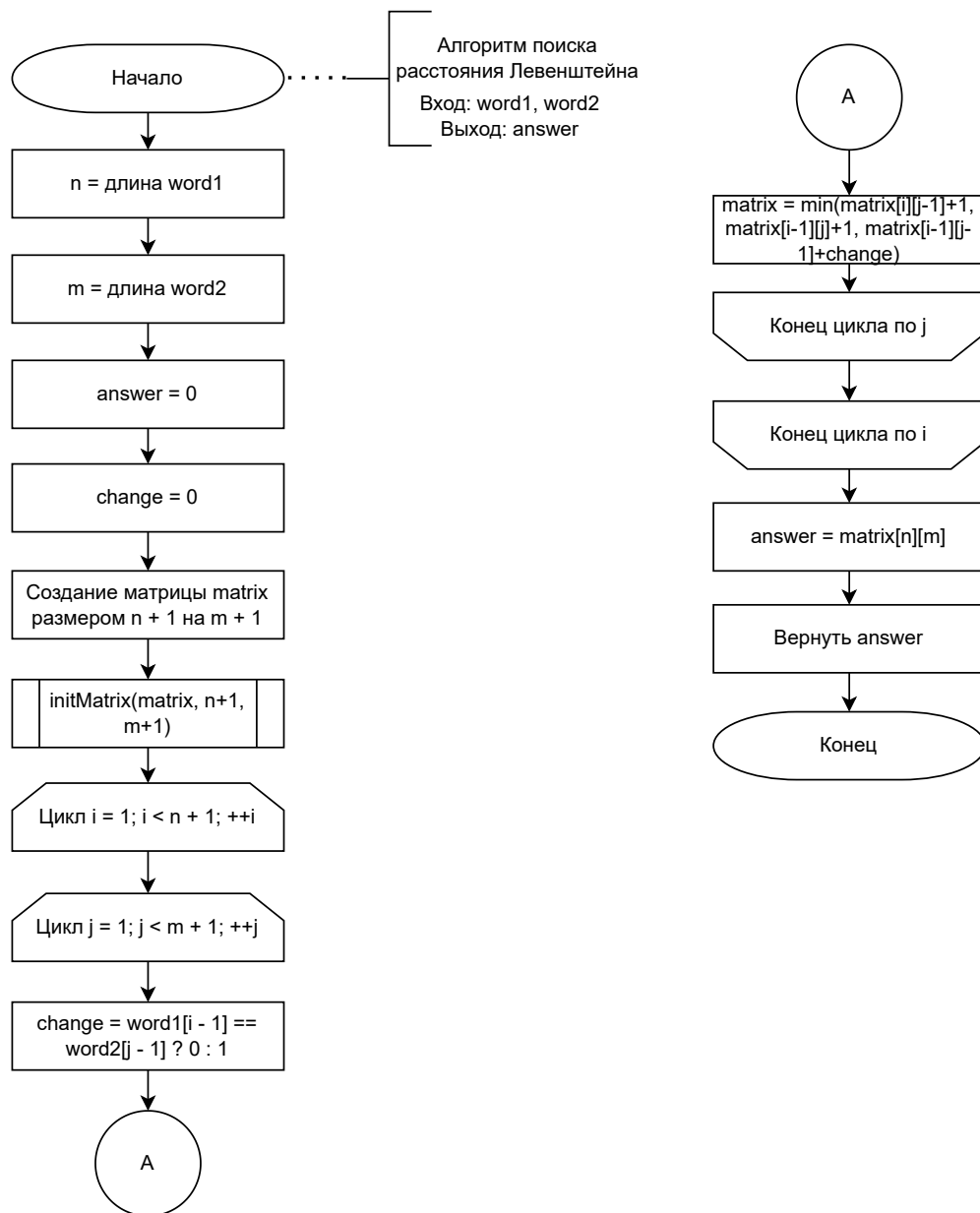


Рисунок 2.1 – Алгоритм поиска расстояния Левенштейна

2.3 Разработка модифицированного алгоритма поиска расстояния Левенштейна

На рисунке 2.2 представлен модифицированный алгоритм поиска расстояния Левенштейна.

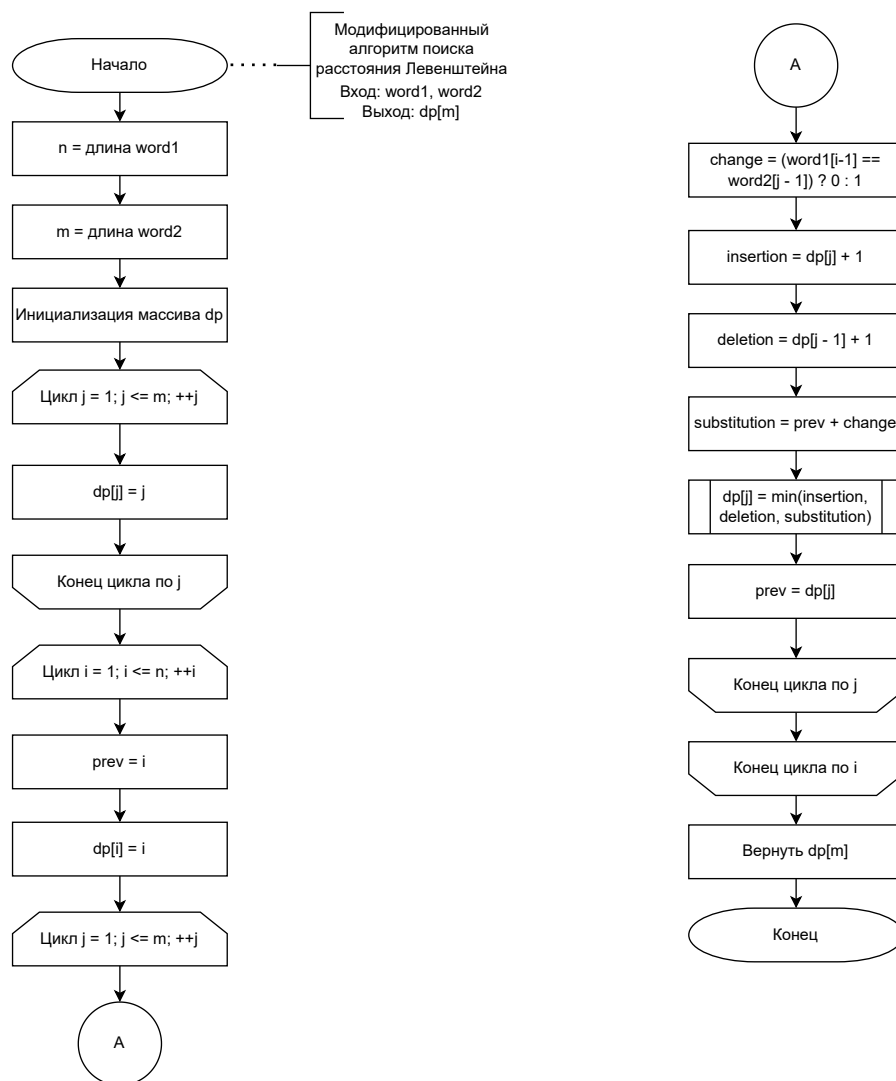


Рисунок 2.2 – Модифицированный алгоритм поиска расстояния Левенштейна

2.4 Разработка многопоточного алгоритма

На рисунке 2.3 представлен многопоточный алгоритм.

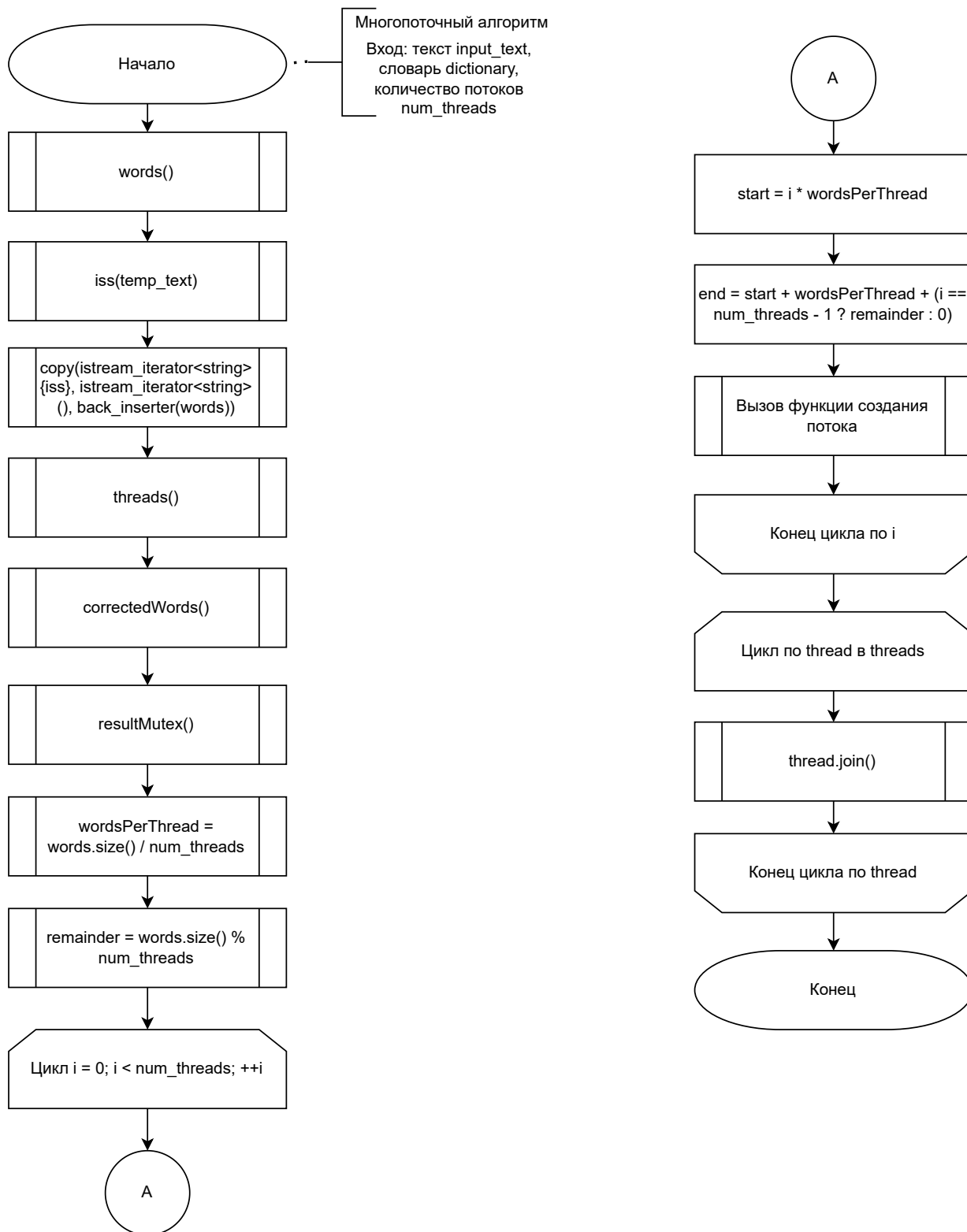


Рисунок 2.3 – Многопоточный алгоритм

3 Технологическая часть

В этом разделе предоставляются листинги реализованных алгоритмов и осуществляется выбор средств реализации.

3.1 Выбор средств реализации

Для выполнения данной лабораторной работы был выбран язык программирования C++. Время измерялось с помощью функции `clock()` из библиотеки `time.h` [4]. Для реализации синхронизации в программе были использованы мьютексы, обеспечивающие безопасный доступ к общим ресурсам [5]. Их применение позволяет избежать гонок данных и обеспечить корректное выполнение потоков.

3.2 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма поиска расстояния Левенштейна.

Листинг 3.1 – Алгоритм поиска расстояния Левенштейна

```
1  int alg_lev(const string &word1, const string &word2)
2  {
3      const size_t n = word1.length();
4      const size_t m = word2.length();
5      int change = 0;
6      vector<vector<int>> matrix(n + 1, vector<int>(m + 1, 0));
7
8      init_matrix(matrix, n + 1, m + 1);
9
10     for (size_t index_i = 1; index_i < n + 1; index_i++)
11     for (size_t index_j = 1; index_j < m + 1; index_j++)
12     {
13         change = word1[index_i - 1] == word2[index_j - 1] ?
EQUAL : NOT_EQUAL;
14         matrix[index_i][index_j] =
min(matrix[index_i][index_j - 1] + 1,
15         min(matrix[index_i - 1][index_j] + 1,
16         matrix[index_i - 1][index_j - 1] + change));
17     }
18     return matrix[n][m];
19 }
```

В листинге 3.2 представлена реализация модифицированного алгоритма поиска расстояния Левенштейна.

Листинг 3.2 – Модифицированный алгоритм поиска расстояния Левенштейна

```
1      int alg_lev_optimized(const string &word1, const string
2      &word2) {
3          const size_t n = word1.length();
4          const size_t m = word2.length();
5          vector<int> dp(m + 1, 0);
6
7          for (size_t j = 1; j <= m; ++j)
8              dp[j] = j;
9
10         for (size_t i = 1; i <= n; ++i)
11         {
12             int prev = i;
13             dp[0] = i;
14
15             for (size_t j = 1; j <= m; ++j)
16             {
17                 int change = (word1[i - 1] == word2[j - 1]) ? 0 :
18                 1;
19
20                 int insertion = dp[j] + 1;
21                 int deletion = dp[j - 1] + 1;
22                 int substitution = prev + change;
23
24                 dp[j] = min({insertion, deletion, substitution});
25                 prev = dp[j];
26             }
27         }
28         return dp[m];
29     }
```

В листинге 3.3 представлена реализация многопоточного алгоритма исправления орфографических ошибок в тексте.

Листинг 3.3 – Многопоточный алгоритм исправления орфографических ошибок в тексте

```
1  void multi_thread(const string &input_text, const
vector<string> &dictionary, int num_threads)
2  {
3      vector<string> words;
4      istringstream iss(input_text);
5      copy(istream_iterator<string>(iss),
istream_iterator<string>(), back_inserter(words));
6
7      vector<thread> threads;
8      vector<string> correctedWords;
9      mutex resultMutex;
10
11     int wordsPerThread = words.size() / num_threads;
12     int remainder = words.size() % num_threads;
13
14     for (int i = 0; i < num_threads; ++i) {
15         int start = i * wordsPerThread;
16         int end = start + wordsPerThread + (i == num_threads
- 1 ? remainder : 0);
17
18         threads.emplace_back(processWords, words.begin() +
start, words.begin() + end,
19         ref(dictionary), ref(correctedWords),
ref(resultMutex));
20     }
21
22     for (auto& thread : threads) {
23         thread.join();
24     }
25 }
```

Вывод

Были выбраны инструменты для реализации и разработаны алгоритм поиска расстояния Левенштейна и модифицированный алгоритм поиска расстояния Левенштейна. Также предоставлены листинги кода на выбранном языке программирования.

4 Исследовательская часть

4.1 Интерфейс приложения

На рисунке 4.1 представлен интерфейс приложения.

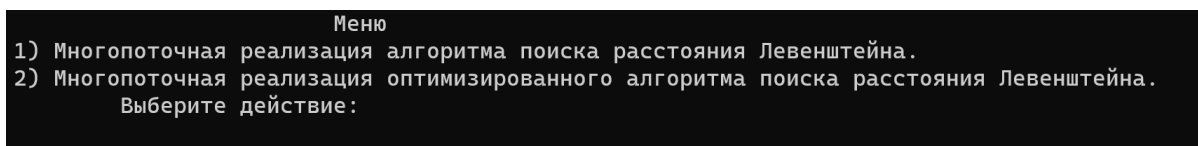


Рисунок 4.1 – Интерфейс приложения

4.2 Технические характеристики

Технические характеристики устройства:

- операционная система — Windows 11 Pro 64 – разрядная система [6];
- оперативная память — 16 Гбайт;
- процессор — 11th Gen Intel(R) Core(TM) i7-1165G7 с тактовой частотой 2.8 ГГц;
- количество ядер — 4 физических и 8 логических ядер.

4.3 Время выполнения реализаций алгоритмов

На рисунке 4.2 приведено сравнение реализации алгоритма поиска расстояния Левенштейна и модифицированного алгоритма поиска расстояния Левенштейна. Время было найдено как среднее пяти измерений.

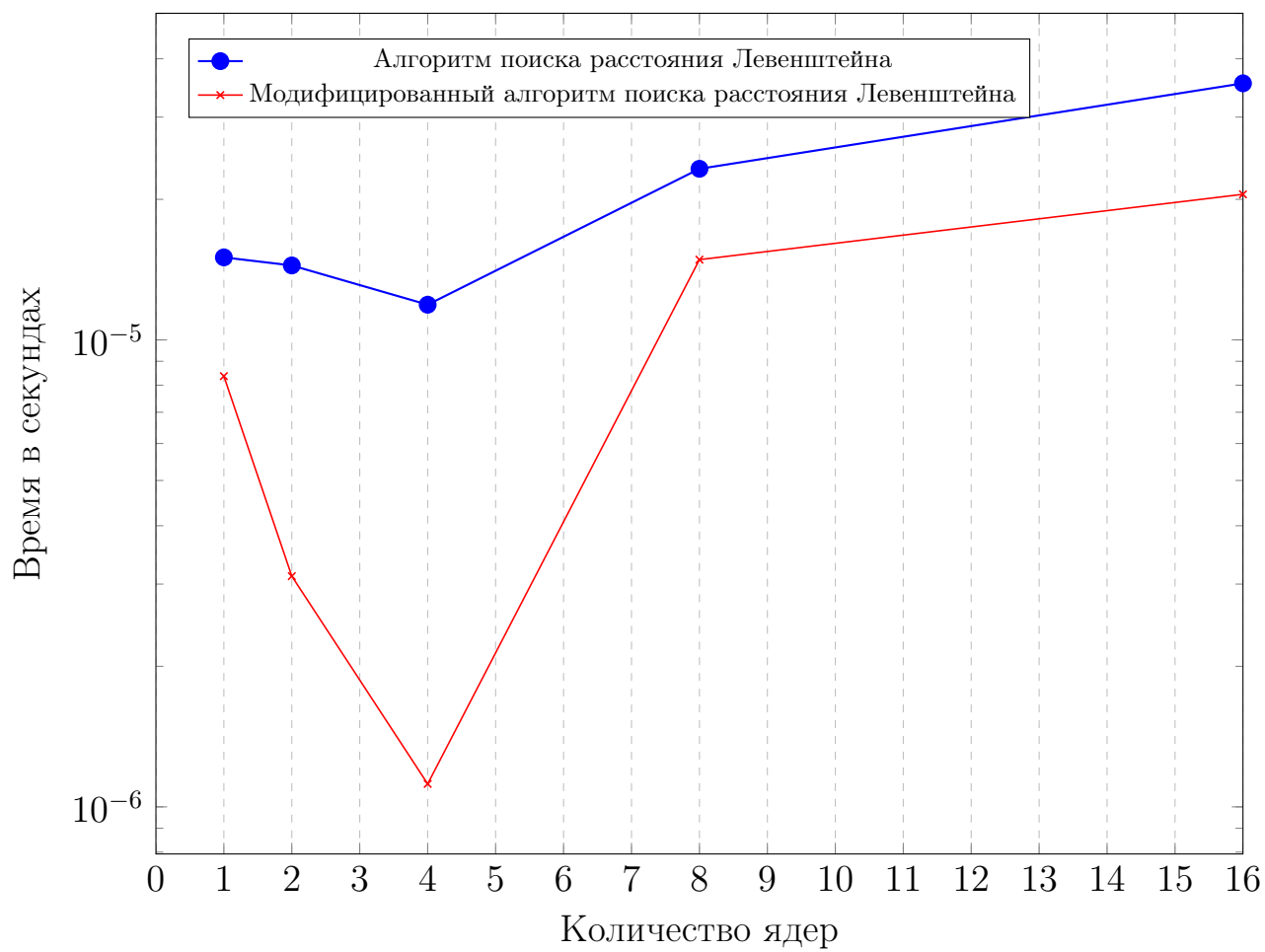


Рисунок 4.2 – Сравнение времени работы реализаций алгоритмов

Вывод

В ходе сравнения алгоритма поиска расстояния Левенштейна и модифицированного алгоритма поиска расстояния Левенштейна выявлено, что модифицированный алгоритм по времени быстрее работает. Этот успех объясняется несколькими ключевыми оптимизациями.

Прежде всего, модификация алгоритма включает в себя использование одномерного массива вместо матрицы для хранения расстояний между символами. Такой подход существенно экономит память и сокращает количество операций обращения к ней, что в итоге приводит к повышению общей производительности.

Дополнительно, внедрена дополнительная переменная, которая заменяет значения двух ячеек ($\text{arr}[i-1][j-1]$ и $\text{arr}[i][j-1]$). Это уменьшает необходимость постоянного обращения к ячейкам матрицы в рамках каждой итерации цикла, что также способствует улучшению общей производительности алгоритма.

Кроме того, применен оптимизированный подход к расчету замены. Расчет стоимости замены осуществляется прямо внутри формулы для определения минимального значения, что сокращает время, требуемое для вычисления изменения, и, следовательно, повышает эффективность алгоритма в целом.

ЗАКЛЮЧЕНИЕ

В результате исследования алгоритма поиска расстояния Левенштейна и его модификации обнаружили, что модифицированный вариант демонстрирует более высокую эффективность. Этот успех объясняется несколькими ключевыми оптимизациями.

Прежде всего, модификация алгоритма включает в себя использование одномерного массива вместо матрицы для хранения расстояний между символами. Такой подход существенно экономит память и сокращает количество операций обращения к ней, что в итоге приводит к повышению общей производительности.

Дополнительно, внедрена дополнительная переменная, которая заменяет значения двух ячеек ($arr[i-1][j-1]$ и $arr[i][j-1]$). Это уменьшает необходимость постоянного обращения к ячейкам матрицы в рамках каждой итерации цикла, что также способствует улучшению общей производительности алгоритма.

Кроме того, применен оптимизированный подход к расчету замены. Расчет стоимости замены осуществляется прямо внутри формулы для определения минимального значения, что сокращает время, требуемое для вычисления изменения, и, следовательно, повышает эффективность алгоритма в целом.

Таким образом, модификация алгоритма поиска расстояния Левенштейна представляет собой успешное слияние оптимизированных методов, что приводит к более быстрой и эффективной обработке строк.

В результате выполнения рубежного контроля выполнены следующие задачи:

- 1) проанализирован параллельный вариант алгоритма исправления орфографических ошибок в тексте;
- 2) определены средства программной реализации выбранного алгоритма;
- 3) реализованы алгоритм поиска расстояния Левенштейна и модифицированный алгоритм поиска расстояния Левенштейна;
- 4) проведен сравнительный анализ по времени реализованного алгоритма;

5) подготовлен отчет о рубежном контроле.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Stoltzfus Justin. Multithreading [Электронный ресурс]. — Режим доступа: <https://www.techopedia.com/definition/24297/multithreading-computer-architecture> (дата обращения: 08.01.2024).
- 2 Стивенс У. Ричард, Раго Стивен А. UNIX. Профессиональное программирование. 3-е издание. — СПб.: Питер, 2018. — 994 с.
- 3 Автоматическая обработка текстов на естественном языке и компьютерная лингвистика / Большакова Е.И., Клышинский Э.С., Ландэ Д.В. [и др.]. — М.: МИЭМ, 2011. — 272 с.
- 4 C library function clock() [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/c/chrono/clock_t (дата обращения: 16.01.2024).
- 5 C++ library mutex [Электронный ресурс]. — Режим доступа: <https://en.cppreference.com/w/cpp/thread/mutex> (дата обращения: 16.01.2024).
- 6 Windows 11 Pro [Электронный ресурс]. — Режим доступа: <https://www.microsoft.com/en-us/windows/business/windows-11-pro#windows11security> (дата обращения: 16.01.2024).