



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 лабораторной работе №3 по курсу "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Зайцева А.А.

Группа ИУ7-52Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Сортировка вставками	4
1.2 Сортировка пузырьком	4
1.3 Шейкерная сортировка	5
2 Конструкторская часть	6
3 Технологическая часть	9
3.1 Требования к ПО	9
3.2 Выбор средств реализации	9
3.3 Листинги кода	9
3.4 Тестирование	11
4 Исследовательская часть	12
4.1 Пример работы	12
4.2 Технические характеристики	12
4.3 Время выполнения алгоритмов	12
4.4 Пиковая требуемая алгоритмами память	14
Заключение	16
Список использованной литературы	17

Введение

Целью данной лабораторной работы является получение навыков оценки трудоемкости алгоритмов на примере трех алгоритмов сортировок.

Под сортировкой понимается упорядочивание элементов по какому-либо признаку. Сортировка относится к важнейшему классу алгоритмов обработки данных и осуществляется большим количеством способов [?].

Алгоритмы сортировки имеют большое практическое применение и часто встречаю там, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные заранее упорядочить. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях [?].

В настоящее время, в связи с постоянно растущими объемами данных, вопрос эффективности сортировки данных не теряет свою актуальность.

В рамках выполнения работы необходимо решить следующие задачи:

- 1) изучить три алгоритма сортировок: сортировку вставками, сортировку пузырьком и шейкерную сортировку;
- 2) разработать и реализовать изученные алгоритмы;
- 3) провести сравнительный анализ трудоёмкости реализаций алгоритмов на основе теоретических расчетов;
- 4) провести сравнительный анализ процессорного времени выполнения реализаций алгоритмов на основе экспериментальных данных.

1 Аналитическая часть

В данном разделе будут рассмотрены основные идеи трех алгоритмов сортировок - сортировки вставками, сортировки пузырьком и шейкерной сортировки.

1.1 Сортировка вставками

Сортировка вставками — достаточно простой алгоритм. Сортируемый массив можно разделить на две части — отсортированную и неотсортированную. В начале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. Начиная со второго элемента массива и заканчивая последним, алгоритм вставляет очередной элемент из неотсортированной части массива в нужную позицию в отсортированной части.

Таким образом, за один шаг сортировки отсортированная часть массива увеличивается на один элемент, а неотсортированная часть массива уменьшается на один элемент. Данный процесс вставки продолжается до тех пор, пока все элементы исходного списка не окажутся в расширяющейся отсортированной части списка [?].

1.2 Сортировка пузырьком

Сортировка пузырьком — один из самых известных алгоритмов сортировки. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно, и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются $N-1$ раз. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма) [?].

1.3 Шейкерная сортировка

Алгоритм шейкерной сортировки является модификацией пузырьковой сортировки по направлению движения. Отличия от нее заключаются в том, что при прохождении части массива, происходит проверка, были ли перестановки. Если их не было, значит эта часть массива уже упорядочена, и она исключается из дальнейшей обработки. Кроме того, при прохождении массива от начала к концу минимальные элементы перемещаются в самое начало, а максимальный элемент сдвигается к концу массива [?].

Вывод

В данном разделе были рассмотрены идеи, лежащие в основе рассматриваемых алгоритмов сортировок - сортировки вставками, сортировки пузырьком и шейкерной сортировки.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы описанных в предыдущем разделе алгоритмов сортировок.

На рисунке 2.1 приведена схема алгоритма сортировки вставками.

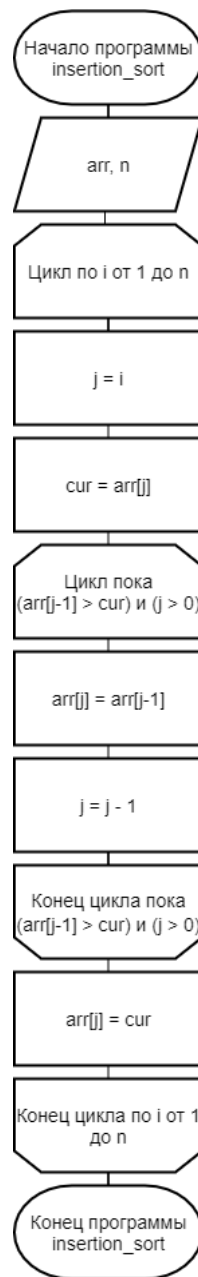


Рисунок 2.1 – Схема алгоритма сортировки вставками

На рисунке 2.2 приведена схема алгоритма сортировки пузырьком.

На рисунке 2.3 приведена схема алгоритма шейкерной сортировки.

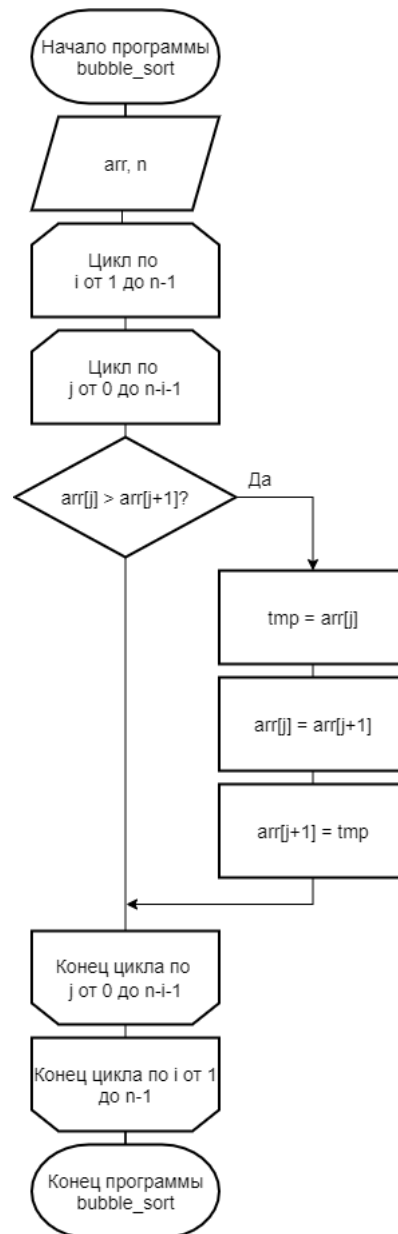


Рисунок 2.2 – Схема алгоритма сортировки пузырьком

Вывод

Были разработаны схемы трех алгоритмов сортировки.

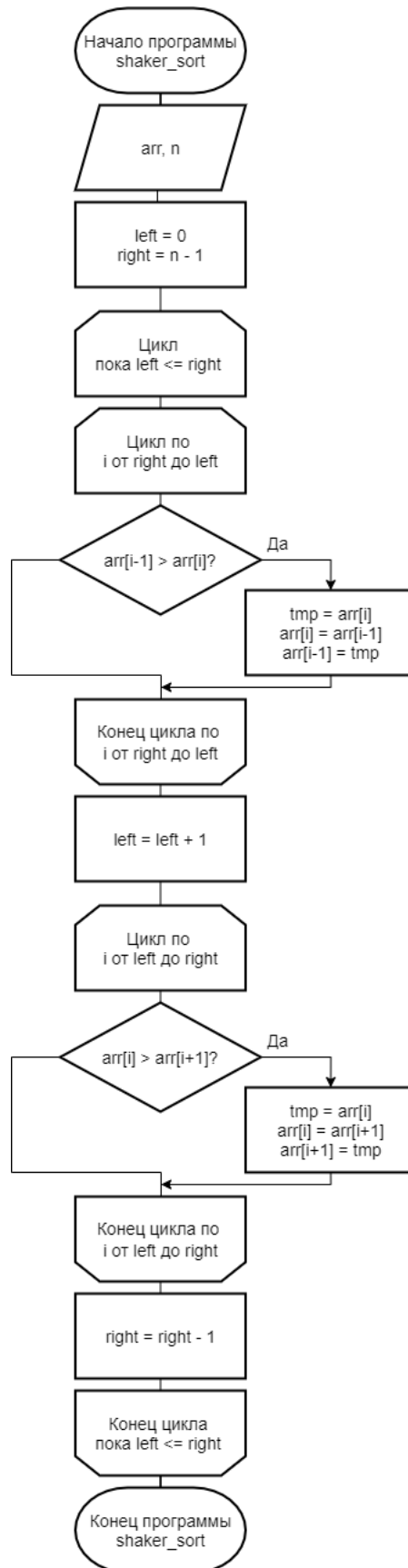


Рисунок 2.3 – Схема алгоритма шейкерной сортировки

3 Технологическая часть

В данном разделе производится выбор средств реализации, а также приводятся требования к программному обеспечению (ПО), листинги реализованных алгоритмов и тесты для программы.

3.1 Требования к ПО

На вход программе подается массив целых чисел, а на выходе должен быть получен массив, отсортированный с помощью каждого реализованного алгоритма сортировки: вставками, пузырьком и шейкерной. Также необходимо вывести затраченное каждым алгоритмом процессорное время.

3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Python [3]. Он позволяет быстро реализовывать различные алгоритмы без выделения большого времени на проектирование структуры программы и выбор типов данных.

Кроме того, в Python есть библиотека `time`, которая предоставляет функцию `process_time` для замера процессорного времени [4].

В качестве среды разработки выбран PyCharm. Он является кросс-платформенным, а также предоставляет удобный и функциональный отладчик и средства для рефакторинга кода, что позволяет быстро находить и исправлять ошибки [5].

3.3 Листинги кода

В листингах 3.1 - ?? представлены реализации рассматриваемых алгоритмов.

Листинг 3.1 – Реализация алгоритма сортировки вставками

```

1  def insertion_sort(arr, n):
2      for i in range(1, n):
3          j = i
4          cur = arr[j]
5          while arr[j - 1] > cur and j > 0:
6              arr[j] = arr[j - 1]
7              j -= 1
8          arr[j] = cur
9      return arr

```

Листинг 3.2 – Реализация алгоритма сортировки пузырьком

```

1  def bubble_sort(arr, n):
2      for i in range(n - 1):
3          for j in range(n - i - 1):
4              if arr[j] > arr[j + 1]:
5                  arr[j], arr[j + 1] = arr[j + 1], arr[j]
6      return arr

```

Листинг 3.3 – Реализация алгоритма шейкерной сортировки

```

1  def shaker_sort(arr, n):
2      left = 0
3      right = n - 1
4      while left <= right:
5          for i in range(right, left, -1):
6              if arr[i - 1] > arr[i]:
7                  arr[i - 1], arr[i] = arr[i], arr[i - 1]
8          left += 1
9          for i in range(left, right, 1):
10             if arr[i] > arr[i + 1]:
11                 arr[i + 1], arr[i] = arr[i], arr[i + 1]
12             right -= 1
13
14      return arr
15

```

3.4 Тестирование

В таблице 3.1 приведены функциональные тесты для алгоритмов сортировок. Все тесты пройдены успешно каждой реализацией.

Таблица 3.1 – Тесты

		Ожидаемый результат	
	-4		
Массив	i	РЛ	РДЛ
		0	0
abc	abc	0	0
ab	a	1	1
a	ab	1	1
see	sea	1	1
1234	1324	2	1
hello	ehlla	3	2
cat	por	3	3
кот	скат	2	2

Вывод

Был произведен выбор средств реализации, реализованы и протестированы алгоритмы поиска расстояний: Левенштейна - итерационный и рекурсивный (с кешем и без), Дамерау-Левенштейна - рекурсивный без кеша

4 Исследовательская часть

4.1 Пример работы

На рисунке 4.1 приведен пример работы программы.

□

Рисунок 4.1 – Пример работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Windows 10;
- оперативная память: 16 Гб;
- процессор: Intel® Core™ i5-8259U.

Во время тестирования ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и системой тестирования.

4.3 Время выполнения алгоритмов

Все реализации алгоритмов сравнивались на случайно сгенерированных строках длиной от 0 до 10 с шагом 1, а те, что используют матрицы расстояний, еще и на строках длиной от 20 до 100 с шагом 10. Так как замеры времени имеют некоторую погрешность, они для каждой длины строк и каждой реализации алгоритма производились 10 раз, а затем вычислялось среднее время работы реализации со строкой.

На рисунке 4.2 приведены результаты сравнения времени работы всех реализаций. Как видно на графике, реализации, использующие матрицы

рассотятий, работают значительно быстрее рекурсивных реализаций без кеширования. Это обусловлено отсутствием в первых вызова функций для вычисления значений, которые уже были подсчитаны ранее.

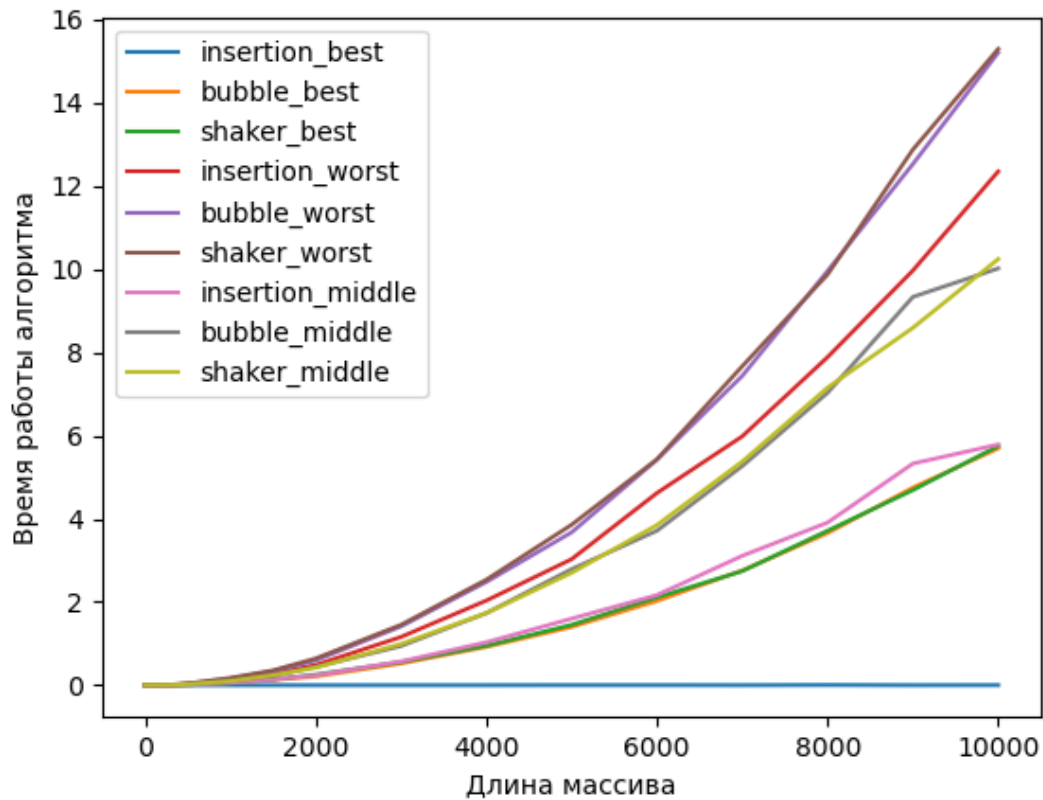


Рисунок 4.2 – Сравнение времени работы реализаций алгоритмов

Чтобы получить полную картину, сравним отдельно реализации, использующие матрицы рассотятий, и те, которые их не используют.

На рисунке 4.3 приведено сравнение времени выполнения реализаций итерационного и рекурсивного (с кешированием) алгоритмов поиска расстояния Левенштейна. Как видно, время их работы растет соизмеримо, что обусловлено их схожестью в отсутствии вызова функций для вычисления значений, которые уже были подсчитаны ранее. Однако вторая реализация все же работает дольше, так как в ней тратится время на рекурсивный вызов самой себя.

Рисунок 4.3 – Сравнение времени работы реализаций рекурсивных алгоритмов, использующих матрицы растояний

На рисунке 4.4 приведено сравнение времени выполнения реализаций

рекурсивных алгоритмов (без кеширования) поиска расстояний Левенштейна и Дамерау-Левенштейна. Эти реализации работали примерно одинаковое количество времени, однако для Дамерау-Левенштейна - немного дольше, так как ему требуется время на подсчет штрафа для дополнительной операции - обмена местами соседних символов.

Рисунок 4.4 – Сравнение времени работы реализаций рекурсивных алгоритмов, не использующих кеш

4.4 Пиковая требуемая алгоритмами память

Реализации алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна имеют одинаковый подход к использованию памяти. Следовательно, достаточно сравнить, как с увеличением длин строк изменяется пиковая затрачиваемая память в рекурсивных и итерационных реализациях.

При использовании рекурсивной реализации без кеширования для каждого рекурсивного вызова функции будет необходимо выделять память под локальные переменные, возвращаемое функцией значение, адрес возврата. Максимальная глубина стека вызовов на 2 больше суммы длин исходных строк. Следовательно, пиковая затрачиваемая память растет пропорционально сумме длин строк.

При использовании итерационной реализации память в большей степени затрачивается на матрицу расстояний размером $\text{len}(S_1) + 1$ на $\text{len}(S_2) + 1$. Следовательно, пиковая затрачиваемая память растет пропорционально произведению длин строк, что больше, чем при рекурсивной реализации.

Однако итерационную реализацию можно оптимизировать по памяти, храня лишь последние две строки матрицы расстояний, и тогда пиковая затрачиваемая память будет расти пропорционально изменению длины одной из строк, что меньше, чем при рекурсивной реализации.

Рекурсивные функции с кешированием по этому показателю - худший вариант, так как они потребуют память как под матрицу расстояний (хотя бы в одном экземпляре), так и под стек вызовов.

При небольших длинах строк затрачиваемая память сильно зависит от самой реализации каждого алгоритма (количество локальных переменных, подходы к хранению строк и матриц)

Вывод

Реализации алгоритмов нахождения расстояния Дамерау — Левенштейна по времени выполнения сопоставимы с реализациями алгоритмом нахождения расстояния Левенштейна, хотя и немного уступают вторым в связи с дополнительной проверкой, позволяющей находить ошибки пользователя, связанные с неверным порядком букв. Однако эта операция зачастую позволяет найти более короткое расстояние между строками.

Рекурсивные реализации алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна, не использующие кеширование, работают на порядок дольше итеративных реализаций. При применении кеширования они требуют меньше времени, однако все равно уступают по производительности итеративным алгоритмам, особенно при большой длине строк.

Но по расходу памяти итеративные реализации проигрывают рекурсивным: максимальный размер используемой памяти в них пропорционален произведению длин строк, в то время как в рекурсивных — сумме длин строк.

Если же применить к итеративным реализациям оптимизацию по памяти, то они будут выигрывать как по пиковой затрачиваемой памяти, так и по времени выполнения.

Заключение

В результате выполнения лабораторной работы при исследовании алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна были применены и отработаны навыки динамического программирования.

В ходе выполнения лабораторной работы были выполнены следующие задачи:

- 1) изучены расстояния Левенштейна и Дамерау-Левенштейна;
- 2) разработаны и проанализованы алгоритмы поиска этих расстояний;
- 3) проведен сравнительный анализ процессорного времени выполнения реализаций этих алгоритмов, а также затрачиваемой ими пиковой памяти;
- 4) был подготовлен отчет по лабораторной работе.

Литература

- [1] В. М. Черненко Ю. Е. Гапанюк. МЕТОДИКА ИДЕНТИФИКАЦИИ ПАССАЖИРА ПО УСТАНОВОЧНЫМ ДАННЫМ // Вестник МГТУ им. Н.Э. Баумана. Сер. “Приборостроение”. 2012. Т. 39. С. 31–35.
- [2] Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. – М.: Доклады АН СССР, 1965. Т. 163. С. 845–848.
- [3] Лутц Марк. Изучаем Python, том 1, 5-е изд. Пер. с англ. — СПб.: ООО “Диалектика”, 2019. Т. 832.
- [4] time — Time access and conversions [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html> (дата обращения: 05.09.2021).
- [5] Python и Pycharm [Электронный ресурс]. Режим доступа: <https://py-charm.blogspot.com/2017/09/pycharm.html> (дата обращения: 05.09.2021).