Частное учреждение профессионального образования «Высшая школа предпринимательства» (ЧУПО «ВШП»)

ОТЧЕТ ПО ПРАКТИКЕ

По основной образовательной программе среднего профессионального образования по специальности “Информационные системы и программирование”

Вид практики: учебная

Установленный по КУГ срок прохождения практики: с 02.12.2025 по 19.12.2025

Место прохождения практики:

_____

_____

_____

Выполнил студент 3-го курса: Голубев А. С.

Руководитель от образовательной организации: _____

Руководитель от предприятия: _____

Оценка: _____

2025 г.

Дата сдачи отчета __.__.20__г.

Введение

В период с 2 декабря по 19 декабря 2025 года была пройдена учебная практика, направленная на изучение способов организации данных и возможностей их применения в реальной профессиональной деятельности. В ходе практики рассматривались основные

методы хранения, обработки и поиска данных, анализировались их преимущества и недостатки, а также эффективность применения в программных и прикладных системах. Полученные знания позволили углубить теоретическую базу и закрепить её на практике, что способствует развитию профессиональных компетенций и подготовке к решению реальных задач в сфере информационных технологий. Все практические задания были выполнены на Python.

Цель практики: формирование практических навыков работы с различными структурами данных, а также понимание принципов их выбора и использования в зависимости от поставленных задач.

Практические задания включали пять разделов:

- Массивы и связные списки
- Стек и очередь
- Map, HashMap, хэш-функции и коллизии
- Деревья и графы
- Кучи и приоритетные очереди

Массивы и связные списки

Задание 1. Статический массив

Смысл этого задания был в закреплении полученных знаний о функциях. В задании были реализованы функции:

Push back, front - Функции добавления нового элемента. Back - новый элемент становится последним, Front - новый элемент занимает первое место.

Insert - более точечное действие. Ты выбираешь конкретное место в линии и вставляешь элемент именно туда, сдвигая остальных.

Remove - это удаление. Находим элемент и убираем его из структуры, после чего линия снова смыкается без пробелов.

Find - поиск. Функция отыщет нужный элемент в массиве, после чего к нему можно будет применить, например функцию удаления.

Задание 2. Динамический массив

В этом задании представлена работа с динамическими массивами.

В ходе практики были изучены особенности работы со статическими и динамическими массивами, а также проведено сравнение времени вставки элементов.

Для сравнения рассматривалась вставка 100 000 элементов. В случае статического массива память под все элементы выделялась заранее, поэтому каждая операция вставки выполнялась за постоянное время, а вот динамический массив начинал работу с небольшого размера и автоматически расширялся по мере заполнения.

Сравнение массивов:

- Динамический: 0.0107 сек
- Статический: 0.0078 сек

Динамический массив оказался чуть медленнее, но при этом значительно удобнее в использовании, так как не требует заранее фиксированного размера и более гибок в реальных задачах.

Таким образом, статические массивы выигрывают по скорости при заранее известном объёме данных, а динамические массивы являются более универсальным решением и чаще применяются на практике, несмотря на небольшие накладные расходы по времени.

Задание 3. Односвязный список

В ходе практики была изучена структура данных “односвязный список“ и реализованы основные операции.

В рамках практики была реализована вставка элемента в начало и в конец односвязного списка: вставка в начало выполняется за постоянное время, так как требует лишь изменения ссылки на первый элемент, вставка в конец также может выполняться эффективно при хранении указателя на последний элемент, однако без него требует обхода всего списка. Были реализованы операции поиска и удаления элемента по значению. Поиск в односвязном списке осуществляется последовательным обходом элементов и имеет линейную сложность. Удаление по значению также требует предварительного поиска нужного элемента и изменения ссылок, что в среднем занимает линейное время. Отдельное внимание было уделено операции разворота списка in-place. Разворот выполнялся без использования дополнительной памяти путём последовательного изменения ссылок между узлами.

При сравнении с массивом было установлено, что вставка и удаление элементов в середину массива требуют сдвига элементов и имеют линейную сложность. А вот односвязный список позволяет выполнять вставку и удаление за постоянное время при наличии ссылки на нужный элемент.

Таким образом, односвязный список эффективен при частых операциях вставки и удаления, особенно в начале структуры, тогда как массивы более удобны и быстры при необходимости произвольного доступа к элементам.

Задание 4. Двусвязный список

В ходе практики был изучен двусвязный список как структура данных, в которой каждый узел хранит ссылки как на следующий, так и на предыдущий элемент. Такая организация позволяет эффективно перемещаться по списку в обоих направлениях и упрощает операции удаления и вставки.

Была реализована операция вставки нового узла после произвольного узла. Данная операция выполняется за постоянное время, так как требует лишь корректного изменения ссылок соседних элементов без необходимости обхода всего списка.

Также было реализовано удаление узла без предварительного поиска.

Дополнительно был реализован итератор для двусвязного списка, позволяющий последовательно обходить элементы структуры ((__iter__) он обеспечивает удобный и безопасный доступ к данным, скрывая детали внутренней реализации списка и упрощая использование структуры в прикладных задачах).

Стек и очередь

Задание 5. Стек

В ходе практики была изучена структура данных «стек» и реализованы два варианта её хранения: на основе массива и на основе связанного списка. Целью работы являлось понимание принципа работы стека и его применения для решения практических задач.

Стек на массиве был реализован с использованием индексной переменной, указывающей на вершину стека. Основные операции — добавление элемента (push) и удаление элемента (pop) — выполняются за постоянное время при условии, что размер массива не превышен. Недостатком данного подхода является ограниченный размер стека или необходимость его динамического расширения.

Стек на связанном списке был реализован с использованием односвязного списка, где вершина стека соответствует голове списка. Операции push и pop также выполняются за константное время, при этом отсутствует ограничение на размер стека, так как память выделяется динамически.

Задание 6. Очередь

В ходе практики была изучена структура данных «очередь» и реализованы два способа её организации: очередь на циклическом массиве и очередь на основе двух стеков. Целью работы являлось понимание принципов работы очереди и способов её эффективной реализации.

Очередь на циклическом массиве была реализована с использованием фиксированного массива и двух индексов - начала и конца очереди. За счёт циклического перехода индексов достигается эффективное использование памяти без необходимости сдвига элементов. Операции добавления (enqueue) и удаления (dequeue) выполняются за постоянное время.

Очередь на двух стеках была реализована с использованием двух структур типа «стек». Один стек использовался для добавления элементов, второй — для извлечения. При необходимости извлечения элементы из первого стека перекладываются во второй, что обеспечивает корректный порядок обработки. В среднем операции добавления и удаления выполняются за амортизированное константное время.

Задание 7. Калькулятор

В ходе практики была реализована программа-калькулятор, которая считывает математическое выражение в инфиксной форме, преобразует его в обратную польскую нотацию (ОПН) и вычисляет результат. Если бы не ОПН, это задание стало бы самым легким, на мой счет.

После преобразования в ОПН выражение становится удобным для последовательного вычисления без необходимости учитывать приоритеты. Вычисление производилось с помощью стека для хранения промежуточных результатов: операнды помещались в стек, а при встрече оператора выполнялась соответствующая операция над верхними элементами стека.

Реализация показала эффективность использования стека для обработки выражений и демонстрирует универсальность этой структуры данных для задач, связанных с последовательной обработкой данных с учётом порядка и приоритета.

Map, HashMap, хэш-функции и коллизии

Задача 8. Своя хэщ-таблица

В ходе практики была реализована хеш-таблица для хранения строк с поддержкой основных операций: добавления (put), получения (get) и удаления (remove) элементов по ключу. Для разрешения коллизий использовались два метода: цепочки (chaining) и открытая адресация.

Хеш-функция преобразует строку в индекс массива, что позволяет обеспечивать быстрый доступ к элементам. В методе цепочек каждый индекс массива хранит список элементов с одинаковым хешем, что позволяет эффективно обрабатывать коллизии. При открытой адресации элементы размещаются в следующей свободной ячейке массива при возникновении коллизии, обеспечивая компактное хранение данных.

В рамках практики была проведена серия операций добавления, получения и удаления элементов, после чего визуализировалось состояние хеш-таблицы. Визуализация позволила наглядно увидеть распределение элементов по ячейкам, влияние коллизий и эффективность различных методов разрешения коллизий. Реализация показала, что хеш-таблица обеспечивает быстрый доступ к данным при большом объёме строк, а выбор метода разрешения коллизий влияет на производительность и использование памяти. Цепочки обеспечивают простоту и гибкость, открытая адресация - компактность и экономию памяти при умеренном количестве коллизий.

Задание 9. Частотный словарь

В ходе практики была реализована программа для подсчёта частоты встречаемости слов в тексте с использованием хеш-таблицы. Основными задачами были построение хэша частотности слов, вывод 10 самых частых слов и анализ влияния качества хеш-функции на производительность.

Для каждой уникальной строки (слова) хеш-таблица хранила количество её вхождений. После обработки текста был произведён вывод 10 слов с наибольшей частотой встречаемости, что позволило наглядно увидеть наиболее значимые элементы текста.

В рамках эксперимента была проведена проверка времени построения частотного словаря при использовании плохой хеш-функции (например, всегда возвращающей одно значение) и хорошей хеш-функции. Результаты показали, что плохая хеш-функция значительно замедляет процесс из-за большого количества коллизий и необходимости последовательного обхода цепочек или поиска свободной ячейки, тогда как хорошая хеш-функция обеспечивает равномерное распределение элементов и минимизирует время обработки.

В результате этого задания было установлено, что эффективность работы хеш-таблицы напрямую зависит от качества хеш-функции, а хеширование является удобным инструментом для анализа частоты слов в тексте и быстрого доступа к данным.

Задание 10. Trie + HashMap

В ходе практики была реализована программа для подсчёта частоты встречаемости слов в тексте с использованием хеш-таблицы, и эта задача оказалась самой тяжелой из за обьема работы. Целью этой задачи стало изучение способов хранения и поиска текстовой информации, реализование структуры данных Trie для хранения слов с возможностью поиска по префиксу и предложением автодополнения, учитывая частоту использования слов.

В классе Trie реализованы методы:

insert(word) – добавление слова и обновление его частоты.

search(word) – проверка наличия слова в структуре.

autocomplete(prefix) – поиск всех слов с данным префиксом и сортировка их по частоте с использованием HashMap.

Для демонстрации использованы тестовые данные: добавление нескольких слов с разной частотой и проверка выдачи подсказок.

Проверено, что при вводе префикса метод возвращает корректные варианты в порядке убывания частоты.

Деревья и графы

Задание 11. Бинарное дерево поиска

Для выполнения этой задачи был создан класс TrieNode с полями для детей (children), флага конца слова (is_end) и счётчика частоты (frequency). В классе Trie реализованы методы:

• insert(word) – добавление слова и обновление его частоты.

• search(word) – проверка наличия слова в структуре.

• autocomplete(prefix) – поиск всех слов с данным префиксом и сортировка их по частоте с использованием HashMap.

Для демонстрации использованы тестовые данные: добавление нескольких слов с разной частотой и проверка выдачи подсказок. Проверено, что при вводе префикса метод возвращает корректные варианты в порядке убывания частоты. Эта задача показала, что комбинация Trie и HashMap позволяет эффективно хранить слова, учитывать частоту их использования и предоставлять быстрые подсказки автодополнения. Такой подход может быть применён в системах поиска и текстовых редакторах для повышения удобства работы с текстом.

Задача 12. Trie

Ход выполнения:

• Реализован класс узла Trie, содержащий:

• коллекцию потомков,

• флаг окончания слова,

• счётчик слов в поддереве.

• При добавлении слова каждый символ последовательно добавляется в дерево, а в конечном узле помечается окончание слова.

• Подсчёт количества вариантов по префиксу реализован путём перехода к узлу, соответствующему последнему символу префикса, и возврата значения счётчика слов в его поддереве.

• Удаление слова реализовано рекурсивно: после снятия флага окончания слова узлы, не участвующие в других словах, удаляются.

В ходе выполнения задания была изучена и реализована структура Trie, подходящая для работы со строками и префиксным поиском. Реализация подсчёта слов по префиксу и удаления элементов делает структуру пригодной для использования в системах автодополнения, поиска и обработки текстовых данных.

Задание 13. Графы

Целью этой задачи было изучить способы представления графов в памяти компьютера и реализовать основные алгоритмы обхода и поиска путей: BFS, DFS и поиск кратчайшего пути в невзвешенном графе.

Существует несколько способов хранения графов:

1. Матрица смежности

Матрица смежности — это двумерный массив размером N × N, где N — количество вершин графа.

Если между вершинами i и j есть ребро, в ячейке [i][j] хранится 1, иначе 0.

2. Список смежности

- BFS (Breadth-First Search, поиск в ширину)

Алгоритм BFS последовательно обходит вершины по уровням, начиная с исходной вершины. Для реализации используется очередь.

- DFS (Depth-First Search, поиск в глубину)

Алгоритм DFS углубляется в граф до тех пор, пока это возможно, после чего возвращается назад. Реализуется с помощью рекурсии или стека.

Задание 14. Острова

В ходе практики была рассмотрена задача поиска количества островов в двумерном массиве, состоящем из значений 0 и 1. Островом считается группа соседних ячеек со значением 1, соединённых по вертикали и горизонтали, что соответствует понятию компоненты связности в графе. Для решения задачи двумерный массив был представлен в виде графа, где каждая ячейка - это вершина, а связи между соседними ячейками - рёбра. Для обхода и поиска компонент связности был использован алгоритм поиска в ширину (BFS). Алгоритм последовательно находил не посещённую ячейку со значением 1, запускал BFS и помечал все достижимые связанные ячейки как посещённые. Каждое новое начало обхода соответствовало обнаружению нового острова. В результате применения алгоритма BFS

было корректно определено количество островов в массиве. Задача показала эффективность алгоритмов обхода графов при работе с двумерными структурами данных и позволила закрепить понятия компонент связности и поиска в ширину.

Кучи и приоритетные очереди

Задание 15. Куча

В ходе практики была реализована бинарная мин-куча - это структура данных, в которой значение в каждом узле меньше или равно значениям его потомков. Такая структура позволяет эффективно работать с приоритетными данными. Были реализованы основные операции: вставка элемента в кучу, извлечение минимального элемента и построение кучи из массива. Вставка выполнялась с последующим восстановлением свойства кучи путём подъёма элемента вверх. Извлечение минимума осуществлялось заменой корня последним элементом с последующим просеиванием вниз. Построение кучи из массива выполнялось за линейное время с использованием операции просеивания. В результате выполнения задания были закреплены принципы работы бинарной кучи и показано, что данная структура данных обеспечивает эффективную реализацию приоритетных очередей и алгоритмов сортировки и планирования.

Задача 16. Приоритетная очередь

В основе реализации использована бинарная мин-куча, где каждый элемент хранится в виде пары (priority, value). Операция push добавляет новый элемент в конец массива с последующим восстановлением свойства кучи методом подъёма вверх. Операция pop всегда возвращает элемент с минимальным приоритетом, заменяя корень последним элементом и выполняя просеивание вниз. В задаче планирования задач элементы извлекались из очереди в порядке возрастания приоритета, что обеспечивало корректную последовательность выполнения задач. Для поиска K минимальных элементов массива данные добавлялись в кучу с последующим извлечением первых K элементов.

В ходе выполнения задания была изучена и реализована приоритетная очередь на основе кучи. Использование мин-кучи позволяет эффективно управлять задачами с приоритетами и выполнять операции выбора минимальных элементов за логарифмическое время, что делает данную структуру данных универсальной и широко применяемой на практике.

Заключение

В ходе работы были изучены базовые операции работы со структурами данных: добавление элементов в начало и конец, вставка в произвольную позицию, удаление и поиск. Понимание функций push, insert, remove и find позволяет эффективно управлять данными и применять эти операции при решении практических задач программирования. Была выполнена комплексная работа по изучению и реализации основных структур данных и алгоритмов, широко применяемых в программировании. Были рассмотрены и реализованы: хеш-таблицы, Trie, бинарное дерево поиска, графы с различными способами хранения, алгоритмы BFS и DFS, задача поиска компонент связности («острова»), бинарная мин-куча и приоритетная очередь. Каждое задание сопровождалось практической реализацией и анализом полученных результатов. В процессе работы удалось закрепить понимание того, как абстрактные структуры данных применяются для решения реальных задач: автодополнение слов, поиск по префиксу, планирование задач, поиск кратчайших путей, обработка двумерных массивов и выбор минимальных элементов. Особое внимание было уделено эффективности алгоритмов и влиянию структуры данных на скорость работы программы. Практика оказалась полезной и познавательной. Работа с кодом помогла лучше понять внутреннюю логику алгоритмов, научиться находить и исправлять ошибки, а также выработать более уверенный подход к решению задач. Впечатления от практики положительные: задания были сложными, но интересными, а полученные навыки можно применять в дальнейшей учёбе и при разработке реальных программных проектов.

Список источников

1. Обычные и динамические массивы

https://skillbox.ru/media/code/chto-takoe-massiv-i-kak-on-ustroen/

https://it.vshp.online/disciplines/up01/lectures/01_arrays_lists.html

2. Односвязные и двусвязные списки

https://pythonchik.ru/osnovy/svyaznyy-spisok-v-python

3. Стек и очередь

https://tproger.ru/translations/stacks-and-queues-for-beginners

https://it.vshp.online/disciplines/up01/lectures/02_stack_queue.html

4. Карта, хэш-таблица и коллизии

https://ru.hexlet.io/courses/python-dicts/lessons/hash-table/theory_unit

5. Деревья и графы

https://it.vshp.online/disciplines/up01/lectures/04_trees_trie_graphs.html

https://proglib.io/p/derevya-i-grafy-chto-eto-takoe-i-pochemu-ih-obyazatelno-nuzhno-znat-kazhdomu-programmistu-2022-06-13

6. Кучи

https://docs-python.ru/standart-library/modul-heapq-python/

https://it.vshp.online/disciplines/up01/lectures/05_heap_priority_queue.html

Ссылка на репозиторий: https://github.com/sstrg/practic