Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа «Компьютерных технологий и информационных систем»

**ОТЧЕТ**

**Лабораторная работа №8**

по дисциплине «Системный подход в разработке программного обеспечения»

**Выполнил:**

студент группы 5130902/20201 \_\_\_\_\_\_\_\_\_\_\_\_\_ А. И. Сафонов

подпись

**Проверил:**

доцент к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_ С. А. Нестеров

подпись

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

Санкт-Петербург, 2025

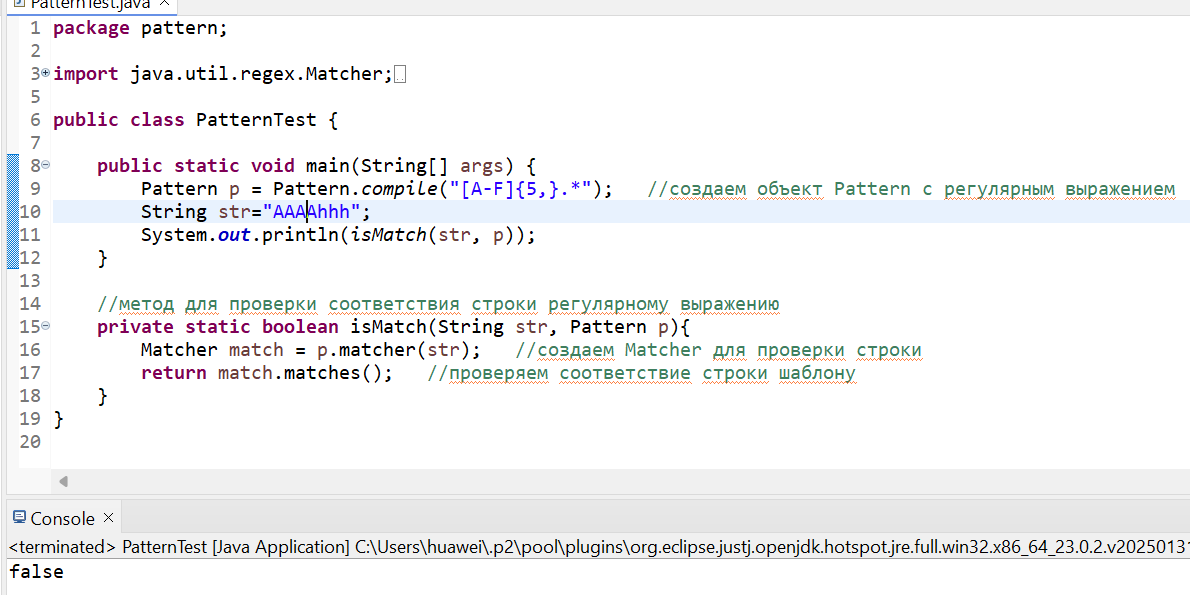
**Упражнения по слайдам 4.2**

**Упражнение 1.**

1. Проверьте, что строка str соответствует заданному шаблону p: matcher сравнивает str и pattern.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.



**Упражнение 2. RegEx Groups**

1. Создайте класс RegExGroups.

2. Отредактируйте метод getDate, чтобы использовать объект matcher

3. Обновите метод main, чтобы вызвать метод getDate() и вывести обновленные значения дат в американском формате.

4. Создайте этот метод в конце класса.

5. Отредактируйте цикл do-while метода getDate, чтобы принимались только корректные даты.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

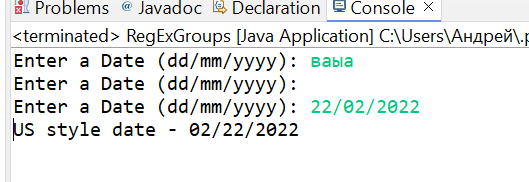
Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Результат:



**Упражнение 3. Замены с регулярными выражениями**

В примере ниже используется регулярное выражение для замены нескольких пробелов в строке одним.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Упражнение 4. Замены с использованием Matcher**

**Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.**

**Практика 4-2: Use regular expressions**

**Vocabulary:**

**Оператор повторений (\*?+{}) —** символ в регулярных выражениях, который указывает количество вхождений, которые должен встречать указанный символ в строке.

**Точка (.) —** символ регулярного выражения, который представляет любой символ, создавая совпадение.

**Pattern —** класс в пакете java.util.regex, который хранит формат регулярного выражения.

**Группы —** сегменты регулярных выражений, начинающиеся с “(” и заканчивающиеся на “)”, которые позже могут быть вызваны методом Matcher.group(groupNumber).

**Квадратные скобки —** используется в регулярных выражениях для обеспечения вариативности символов; может содержать либо заданный диапазон символов, либо группу опций символов.

**Matcher —** класс в пакете java.util.regex, который хранит возможные совпадения между Pattern и String.

**Регулярное выражение —** символ или последовательность символов, которая представляет строку или несколько строк и позволяет варьировать совпадения.

**Try It/Solve It:**

1. Перечислите **четыре символа**, **используемые в регулярных выражениях**, и опишите, что каждый из них представляет.

**. (Точка)** **—** соответствует любому одному символу, кроме символа новой строки.

**\* (Звездочка) —** соответствует 0 или более вхождениям предыдущего символа или группы.

**+ (Плюс) —** соответствует 1 или более вхождениям предыдущего символа или группы.

**? (Вопросительный знак) —** соответствует 0 или 1 вхождению предыдущего символа или группы.

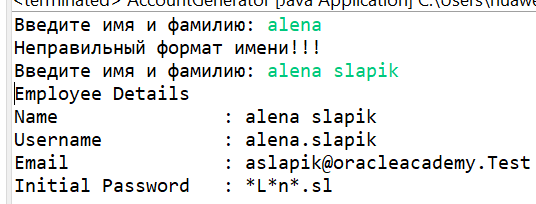
**2. Откройте программу accountgenerator, которая была начата в JP\_4\_1\_Practice.**  
 На данный момент, если ввести имя, а затем пробел, программа примет это как правильный ввод для имени, даже если фамилия не была введена. Используйте регулярное выражение, чтобы предотвратить это. Необходимо принимать только одно или более символов, за которыми идет пробел, а затем одно или более символов.

Разместите регулярное выражение в операторе if, который отобразит сообщение "Неправильный формат имени", если ввод не соответствует регулярному выражению.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Результат:



3. Ваш учитель только что дал вам ответы на финальный экзамен (гипотетически, конечно)! Единственная проблема в том, что ключ с ответами зашифрован, и в нем содержатся числа и другие символы, которые никогда не будут вариантами ответов.

Единственная подсказка, которую учитель дает вам в этот момент, заключается в том, что любой символ в ключе, который является a, A, b, B, c, C, d, D, e, E, f или F, является частью ключа ответа в том порядке, в котором он появляется в зашифрованном ключе. Любой другой символ не является частью ключа ответа. Каждый символ находится на своей строке в предоставленном учителем файле.

Чтобы расшифровать зашифрованный ключ, завершите программу ниже, чтобы она правильно читала каждую строку из файла **CodedAnswerKey** как строку **line** (эта часть сделана за вас), использовала регулярное выражение, чтобы проверить, является ли строка **line** одним из 12 вариантов ответа на экзамене. Если это так, добавьте его в строку **answers**. В конце выведите **answers**.

Файл:

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Код программы:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Результат:



4. Наступает время для вашего финального экзамена, и ваш учитель только что объявил, что ответы будут только в пределах [a-dA-D]! Она дала вам последнюю инструкцию для расшифровки ключа ответов: «Замените все e на b, все E на A, все f на c и все F на D. Затем преобразуйте строку с ответами в нижний регистр, чтобы вы могли использовать её на экзамене. Если ваши ответы не будут в нижнем регистре, вы не сможете использовать их на экзамене!»

Напишите статический метод **finalAnswers**, который принимает строку **answers**, полученную в задаче 2, и возвращает строку, измененную в соответствии с последними инструкциями учителя.

Код:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.



5. Даны следующие регулярные выражения. Определите, какие из значений для строки заставят метод matches вернуть true. Символ ? означает 0 или 1 вхождение любого символа, квадратные скобки [Bb] будут включать только одно вхождение символа 'B' или 'b', а символ \* означает 1 или более вхождений любого символа.

Изображение выглядит как текст, чек, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Вернет anana и banana

Изображение выглядит как текст, Шрифт, снимок экрана, чек

Контент, сгенерированный ИИ, может содержать ошибки.

Вернет banana



Изображение выглядит как текст, чек, Шрифт, алгебра

Контент, сгенерированный ИИ, может содержать ошибки.

Вернет montanana и \_anana

**Упражнения по слайдам 4.3**

**Упражнение 1. Прямые и рекурсивные последовательности**

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

Результат:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

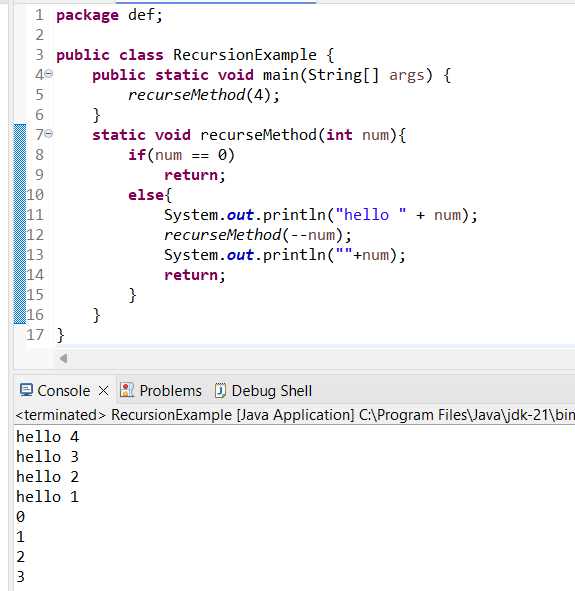
**Упражнение 2. Пример линейной рекурсии**

1. Создайте следующий проект (не запускайте его).

2. Пройдите по этому коду, отслеживая обратное выполнение на бумаге. Что будет выведено?

3. Запустите программу

4. Сравните результат работы с тем, который получился у вас на шаге 2. Совпадает ли он с тем, что вы посчитали?



**Упражнение 3. Линейная рекурсия: факториал**

5. Добавьте алгоритм вычисления факториала в код RecursionExample:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Вызов метода:



Итог:



**Упражнение 4. Нелинейная рекурсия: задача Фибоначчи**

Задача Фибоначчи: Содержит две или более рекурсивных ветви. Вызывает вначале первую ветвь, пока не достигнет базы рекурсии, затем вторую. Возвращает значения от нижнего рекурсивного вызова к исходному для каждой ветки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

**Практика 4-3: Recursion**

**Vocabulary:**

**Рекурсия** – процесс рекурсивного вызова копии того же метода до достижения базового случая. В некоторых случаях, как в задаче Фибоначчи, два значения могут вызывать разные базовые случаи, что приводит к **нелинейной рекурсии**. В такой рекурсии базовый случай должен учитывать несколько возвращаемых значений.

**Линейная рекурсия** – процесс вызова **только одной** копии того же метода внутри метода.

**Нелинейная рекурсия** – процесс вызова **двух или более** копий того же метода внутри метода. Как правило, вызовы разделены оператором в алгоритме.

**Сходимость** – процесс «обратного решения» проблемы путем рекурсивного вызова копии метода до достижения базового случая, после чего значения возвращаются **вверх по цепочке** для окончательного решения задачи.

**База рекурсии** – последний случай, обрабатываемый рекурсивной программой. В задаче Фибоначчи это первые два числа последовательности, которые вычисляются последними при рекурсивном решении.

**Рекурсивная ветвь** – альтернативный случай по отношению к базовому, выполняемый, если условия базового случая не выполняются. В этом случае программа вызывает еще одну копию или несколько копий себя для дальнейшего решения задачи.

**Try It/Solve It:**

1. Создайте класс для определения и тестирования линейной рекурсии. Назовите класс **Linear**. Реализуйте следующее:

* Публичный класс с именем **“Linear”**.
* Публичный статический метод с именем **“factorial”**, который принимает тип **double** и возвращает **double**.
* Базовый случай должен проверять, является ли входное значение меньше или равно 1.
* Рекурсивный случай должен возвращать входное значение, умноженное на результат рекурсивного вызова метода **factorial** с входным значением, уменьшенным на 1.

Создайте статический метод **main**, который тестирует следующее:

* переменную с именем **“d” типа double**.
* Проверьте, содержит ли список аргументов какие-либо значения, и присвойте значение нулевого аргумента переменной **d**. Если список аргументов пуст, присвойте значение 5.
* Выведите значение факториала и исходное входное значение.
* Хотя формат вывода может отличаться, это ожидаемый вывод:

**Factorial [120.0] of [5.0]**.

Код:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Итог:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

1. Создайте класс для определения и тестирования нелинейной рекурсии. Назовите класс **NonLinear**. Реализуйте следующее:

* Публичный класс с именем **NonLinear**.
* Публичный статический метод с именем **fibonacci**, который принимает значение типа **double** и возвращает значение типа **double**.
* Базовый случай должен проверять, если входное значение меньше 2.
* Рекурсивный случай должен возвращать следующее:  
  **return fibonacci(d - 1) + fibonacci(d - 2);**

Создайте статический метод **main**, который тестирует следующее:

* Переменную типа **double** с именем **d**.
* Проверьте, содержит ли список аргументов какие-либо значения, и присвойте нулевой аргумент локальной переменной **d**. Если список аргументов пуст, присвойте значение 5.
* Выведите значение числа Фибоначчи и исходное входное значение.
* Формат вывода может немного отличаться, но вот ожидаемый вывод:

Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Контент, сгенерированный ИИ, может содержать ошибки.

Код:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Результат:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

3. Проследите выполнение следующего кода с помощью диаграммы, а затем примените метод обратного мышления для нахождения факториала числа 7.

Изображение выглядит как текст, Шрифт, снимок экрана

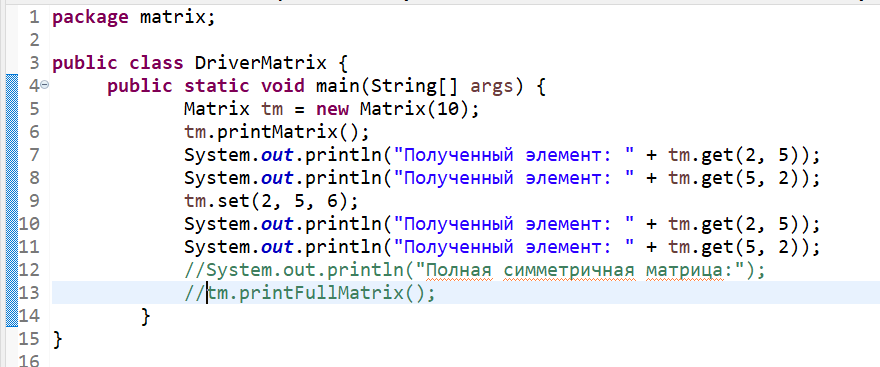
Контент, сгенерированный ИИ, может содержать ошибки.

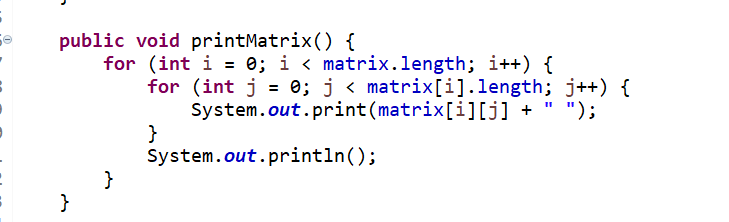
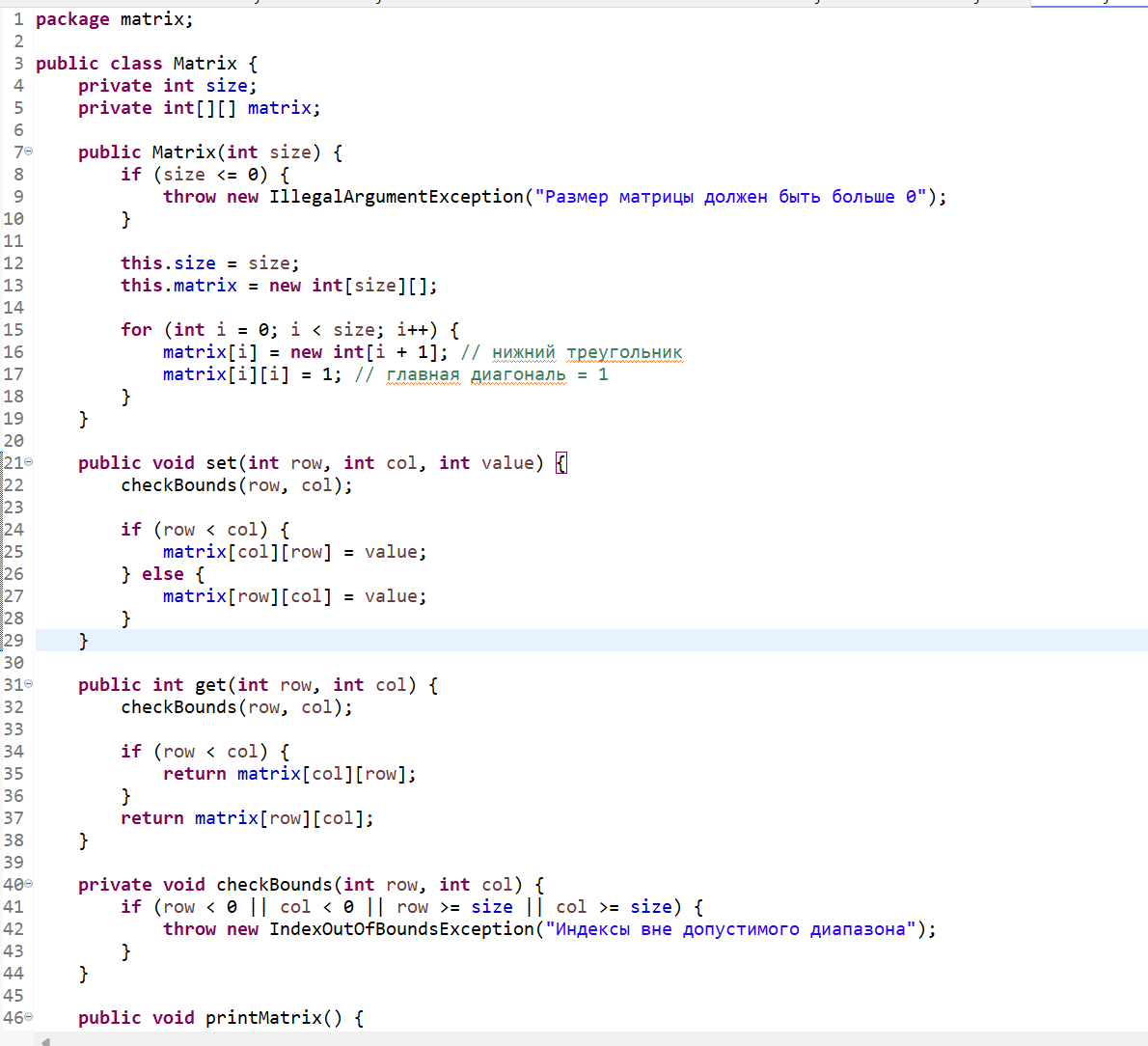
Отследим процесс нахождения факториала числа 7:

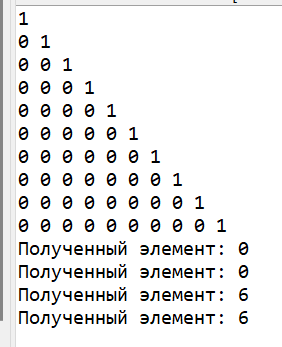
|  |  |  |
| --- | --- | --- |
| Вызов | Результат вызова | Фактическое значение |
| factorial(7) | 7 \* factorial(6) | 7 \* 720 = **5040** |
| factorial(6) | 6 \* factorial(5) | 6 \* 120 = 720 |
| factorial(5) | 5 \* factorial(4) | 5 \* 24 = 120 |
| factorial(4) | 4 \* factorial(3) | 4 \* 6 = 24 |
| factorial(3) | 3 \* factorial(2) | 3 \* 2 = 6 |
| factorial(2) | 2 \* factorial(1) | 2 \* 1 = 2 |
| factorial(1) | База рекурсии: 1 | 1 |

## **Задание в классе**

Пишем реализацию класса, подменяющего квадратную матрицу треугольной для хранения матрицы смежности (книга про алгоритмы с.105). Можно не разворачивать треугольную матрицу в одномерный массив, а использовать двумерный массив Java со строками разной длины. В конструкторе в качестве параметра получаем размерность (одно значение типа int), создаем нужного размера треугольную матрицу, диагональ заполняем 1, остальное - 0. Пишем методы set и get, устанавливающие или возвращающие значение элемента. Если запрошенный элемент над диагональю матрицы, подменяем его зеркальным элементом под диагональю. Обрабатываем (и при необходимости - выбрасываем) исключения - если размерность нулевая или отрицательная, запрашиваемые индексы за пределом размерности квадратной матрицы, устанавливаемое значение отлично от 0 или 1

****

****

****

**Конспект.**

**Глава 4. Массивы**

Массив — это большое количество непрерывной памяти, доступ к которой осуществляется с помощью индексов (их количество указывает на размерность массива). При создании массива программа, как правило, задает его размерность и границы этой размерности.

В одномерном массиве назначение индекса происходит просто: индекс i соответствует записи i. В двумерном аналогичная операция осуществляется одним из двух способов:

* путем развертывания по строкам
* путем развертывания столбцам

**Одномерные массивы**

**1. Введение в одномерные массивы**

Одномерные массивы (или линейные массивы) представляют собой коллекцию элементов, расположенных последовательно в памяти. Эти алгоритмы являются основой для работы с более сложными структурами данных, такими как связные списки, стеки и очереди.

**2. Алгоритмы поиска**

**Линейный поиск** используется, если элементы массива не отсортированы. Время работы алгоритма — O(N), так как приходится проверять каждый элемент массива по очереди.

**3. Нахождение минимальной, максимальной и средней величины**

Для всех этих операций время работы алгоритма составляет O(N), так как необходимо пройти по всем элементам массива.

**4. Нахождение срединного значения**

Поиск срединного значения в массиве требует сортировки данных или выполнения сложных вычислений, поскольку для этого нужно определить количество элементов, меньших и больших выбранного. Алгоритм, который делает это за O(N²), не эффективен для больших массивов, и может быть улучшен с использованием сортировки за O(N log N).

**5. Вставка элементов**

Вставка элемента в произвольную позицию массива требует сдвига всех последующих элементов. Время работы этого алгоритма — O(N), так как в худшем случае (вставка в начало массива) нужно сдвигать все элементы.

**6. Удаление элементов**

Удаление элемента из массива также требует сдвига всех элементов после удаляемого, чтобы они заполнили освободившуюся позицию. В худшем случае удаление первого элемента также займет O(N) времени.

**Треугольные массивы**

Треугольные массивы — это эффективная альтернатива прямоугольным массивам, когда необходимо сэкономить память. В таких массивах элементы, расположенные выше главной диагонали , заполняются значениями по умолчанию (например, 0 или null).

**Пример использования**: В матрице смежности для графов (например, для авиамаршрутов) записи для двух направлений, например, полет от аэропорта i до j и от j до i, могут быть одинаковыми. Вместо хранения обеих записей, можно использовать треугольный массив, что позволяет значительно сэкономить пространство.

**Массивы с разрывом**

Массивы с разрывом помогают эффективно экономить память, особенно когда количество записей значительно меньше общего числа возможных записей. Это решение подходит для случаев, когда массив содержит множество пустых или неиспользуемых значений, как, например, в матрице смежности для сети авиарейсов.

**Принцип работы**: Вместо хранения всех записей, массив с разрывом сохраняет только те элементы, которые имеют значение. Пустые места не хранятся, и если требуется доступ к отсутствующему элементу, возвращается значение по умолчанию (например, 0).

**Структура данных**: Массив с разрывом может быть реализован с использованием связных списков:

* **ArrayRow**: хранит номер строки, ссылку на следующую строку и ссылку на первый элемент в строке.
* **ArrayEntry**: хранит номер столбца, значение элемента и ссылку на следующий элемент в строке.

**Операции**:

1. Получение значения по строке и столбцу или возвращение значения по умолчанию.
2. Установка значения в строку и столбец.
3. Удаление значения из строки и столбца.

**Алгоритмы** для этих операций требуют поиска нужной строки и столбца, что упрощается благодаря использованию связных списков.

**Время работы**:

* **FindRowBefore**: O(N) — где N — количество строк.
* **FindColumnBefore**: O(M) — где M — количество значений в строке.
* **GetValue, SetValue, DeleteEntry**: O(N + M) — общее время для работы с массивом, где N — количество строк, M — количество значений.

**Матрицы**

Массивы широко применяются для представления матриц и выполнения с ними различного рода операций.