

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  **информационных**  **технологий** | **Кафедра**  **управления и информатики**  **в технических системах** |

**Реферат**  
ПО ДИСЦИПЛИНЕ

«Алгоритмы и структуры данных»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТА | 2 | КУРСА | | БАКАЛАВРИАТА | ГРУППЫ | ИДБ-23-13 |
|  | | | *(уровень профессионального образования)* | |  | |

Е

|  |
| --- |
| **Таган Артема Максимовича** |
| *(ФИО)* |

ТЕМА РАБОТЫ

|  |
| --- |
| «Перечислите общие подходы к разработке алгоритмов. Опишите методы  разработки алгоритмов. Каковы основные преимущества и недостатки их  использования?» |

|  |  |
| --- | --- |
| Направление: | 09.03.03 Прикладная информатика |
| Профиль подготовки: | — |

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет сдан «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. | | | |
|  |  |  |  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  | | | |
| Преподаватель | Евдокимов С.А, доцент |  |  |
|  | *(Ф.И.О., должность, степень, звание.)* |  | *(подпись)* |

МОСКВА 2024

Содержание

Оглавление

[**Введение** 3](#_Toc178279690)

[**Основные подходы к разработке алгоритмов** 3](#_Toc178279691)

[**Метод грубой силы (brute fource)** 4](#_Toc178279692)

[**Метод декомпозиции( decomposition)** 5](#_Toc178279693)

# **Введение**

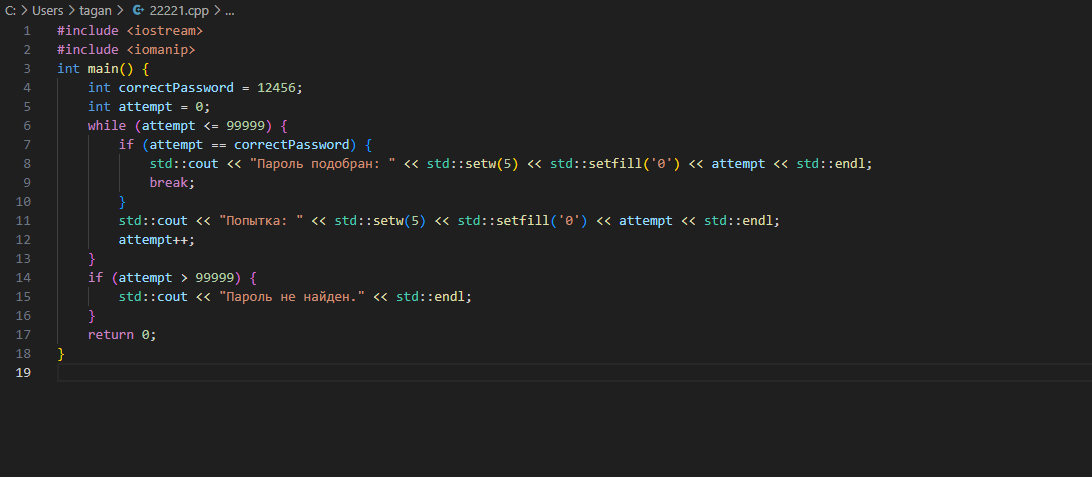
Алгоритм представляет собой последовательность действий или инструкций для выполнения задачи или решения проблемы. Разработка эффективных алгоритмов играет ключевую роль в программировании и компьютерных науках, так как правильный выбор метода решения может существенно повлиять на производительность и точность результата.

# **Основные алгоритмы**

1. Метод грубой силы (brute force, исчерпывающий поиск – полный перебор)
2. Декомпозиция (decomposition, “разделяй и властвуй”)
3. Жадные алгоритмы (greedy algorithms)

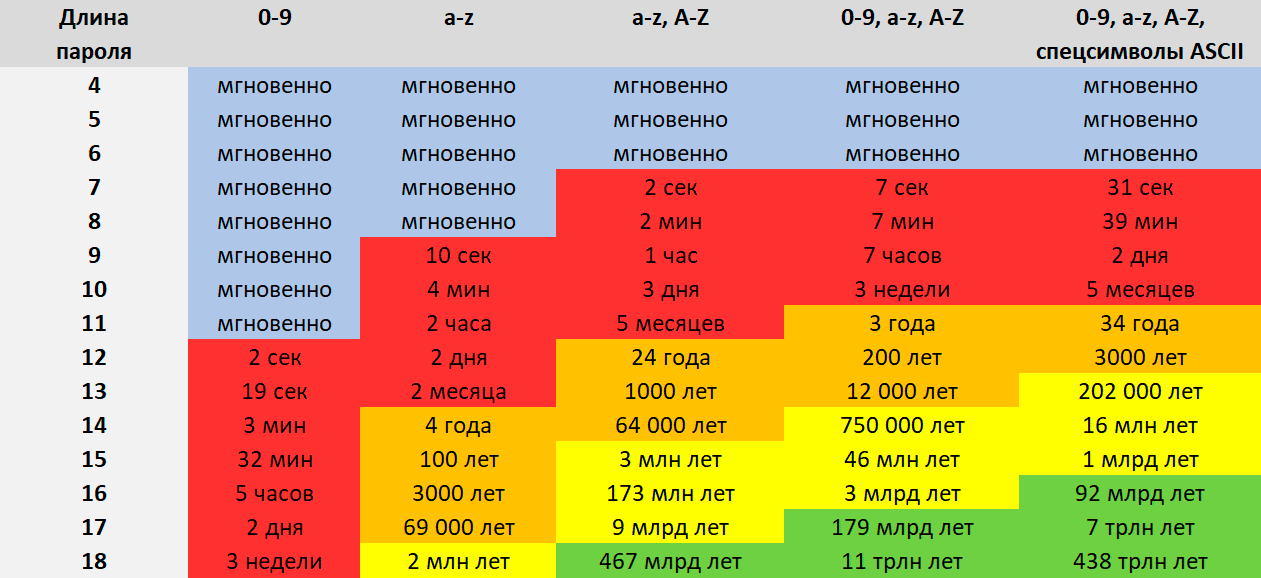
# **Метод грубой силы (brute fource)**

Метод грубой силы (brute force) – решение “в лоб”. Метод грубой силы представляет собой прямой подход к решению задачи, обычно основанный непосредственно на формулировке задачи и определениях используемых ею концепций. «Сила» в определении стратегии – сила компьютера, а не сила интеллекта, т. е. сила из пословицы: «Сила есть — ума не надо».



Рис(1) Пример кода “брутфорса” пароля

Метод брутфорса подходит для решения каких то задач, где невозможно сделать более эффективный алгоритм, или другие не показывают такой эффективности. Тот же пароль подбирать можно не только брутфорсом, но и банальным использованием словарей с самыми частыми паролями.



Рис(2) Пример сложности для “брутфорса” пароля

Как видно по картинке, брутфорс может показывать себя очень хорошо в некоторых ситуациях.

# **Метод декомпозиции( decomposition)**

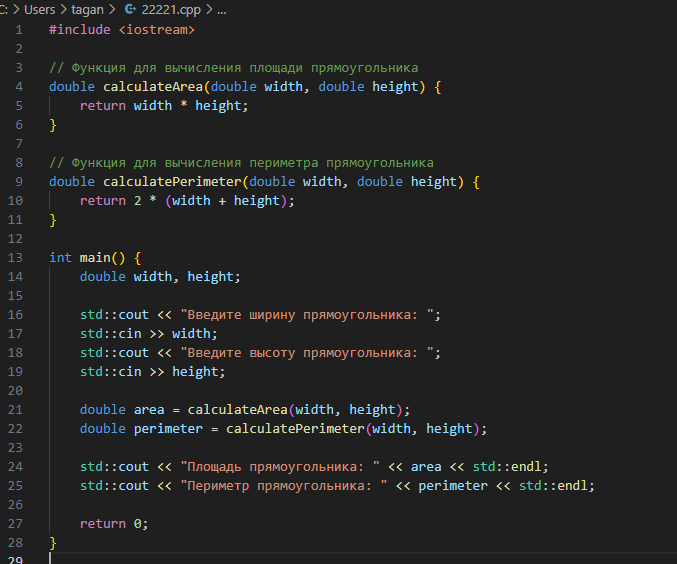
Декомпозиция — разделение целого на части. Также декомпозицией называется общий приём, применяемый при решении проблем, состоящий в разделении проблемы на множество частных проблем, а также задач, не превосходящих в совокупности по сложности исходной проблемы, соединение решений которых образует решение исходной

Структура алгоритмов, основанных на этом методе:

1. Задача разбивается на несколько меньших экземпляров той же задачи

2. Решаются сформированные меньшие экземпляры задачи (обычно рекурсивно)

3. При необходимости решение исходной задачи формируется как комбинация решений меньших экземпляров задачи



Рис(3) Пример кода c декомпозицией

Мы разбили задачу по подсчету площади и периметра треугольника на 2. Мы конечно могли бы всё сделать в одной функции, но в более сложных задачах это может сыграть с нами злую шутку. Если данные должны проходить через несколько функций гораздо проще будет отследить то где ошибка.

# **Жадные алгоритмы (greedy algorithms)**

Если кратко, то жадный алгоритм (greedy algorithm) — это алгоритм, который на каждом шагу делает локально наилучший выбор в надежде, что итоговое решение будет оптимальным.



Рис(3)Пример кода c жадным алгоритмом

Пример жадного алгоритма можно продемонстрировать на задаче "задача о рюкзаке" (Knapsack Problem) с использованием версии задачи с **дробными предметами** (Fractional Knapsack Problem). В этой задаче у нас есть набор предметов, каждый из которых имеет определённый вес и стоимость. Нужно максимально заполнить рюкзак с ограниченной вместимостью, чтобы получить максимальную стоимость.

# **Методы создания алгоритмов**

Составление алгоритмов для решения задач — это творческий процесс, не поддающийся универсализации. Не существует метода, который позволял бы легко и автоматически создавать алгоритмы для любых задач, ведь жизненные ситуации и проблемы настолько разнообразны и непредсказуемы! Если бы такой способ существовал, то можно было бы автоматизировать сам процесс создания алгоритмов, доверив его высокоинтеллектуальному компьютеру.

Однако при решении простых задач можно следовать определённой схеме. В вычислительной математике накоплен многолетний, а иногда и многовековой, опыт решения разнообразных вычислительных задач. Нет необходимости заново изобретать уже существующие алгоритмы — достаточно изучить их и применять на практике для решения своих задач. Примером могут служить методы нахождения корней нелинейных уравнений, вычисления определённых интегралов, численного решения дифференциальных уравнений, сортировки данных и другие.

Зачастую одну и ту же задачу можно решить несколькими численными методами. Выбор конкретного метода обычно основывается на следующих критериях:

* обеспечение оптимального времени решения задачи;
* обеспечение оптимального использования имеющихся ресурсов (памяти);
* обеспечение требуемой точности вычислений;
* минимальные стоимостные затраты;
* возможность использования стандартных подпрограмм.

# **Основные Плюсы и Минусы использования алгоритмов**

Плюсы:

1. Эффективность: Алгоритмы позволяют автоматизировать процессы и решать задачи быстрее и точнее, чем при ручном подходе. Например, сортировка больших объемов данных или нахождение оптимальных решений с использованием минимальных ресурсов.

2. Оптимизация ресурсов: Хорошо продуманные алгоритмы помогают сэкономить вычислительные ресурсы, такие как время выполнения программы и объем памяти, что важно для масштабируемых систем.

3. Стандартизация решений: Алгоритмы предоставляют универсальные способы решения задач. Использование общепринятых алгоритмов, таких как "Поиск в глубину" или "Быстрая сортировка", делает программы легче в поддержке и понятнее для других разработчиков.

4. Надежность и предсказуемость: Алгоритмы гарантируют предсказуемые результаты при заданных входных данных. Это делает их надежными для повторяющихся задач и обработки данных в реальном времени.

5. Масштабируемость: Множество алгоритмов могут эффективно работать с увеличивающимся объемом данных. Например, алгоритмы машинного обучения и анализа больших данных, такие как кластеризация и линейная регрессия, способны обрабатывать огромные данные, не теряя точности.

6. Многозадачность и параллелизм: Алгоритмы могут быть разработаны для выполнения параллельных вычислений, что увеличивает производительность на многоядерных системах и кластерах серверов.

Минусы:

1. Сложность разработки: Разработка и оптимизация эффективных алгоритмов может быть сложной задачей, требующей глубоких знаний в математике, теории графов, анализа алгоритмов и программирования. Ошибки в реализации могут привести к неправильным результатам.

2. Не всегда универсальны: Некоторые алгоритмы хорошо работают только для конкретных задач или в определенных условиях. Например, "Быстрая сортировка" хороша для случайных данных, но может быть менее эффективной при сортировке отсортированных данных.

3. Зависимость от входных данных: Многие алгоритмы сильно зависят от качества входных данных. Например, алгоритмы машинного обучения требуют чистых и хорошо структурированных данных, чтобы давать точные результаты.

4. Может быть неэффективным для маленьких задач: Иногда использование сложных алгоритмов для простых задач может привести к избыточности и ненужной нагрузке на систему, когда простое решение могло бы быть более эффективным.

5. Трудоемкость в понимании: Некоторые сложные алгоритмы могут быть трудными для понимания и реализации. Например, динамическое программирование или алгоритмы криптографии могут требовать серьез