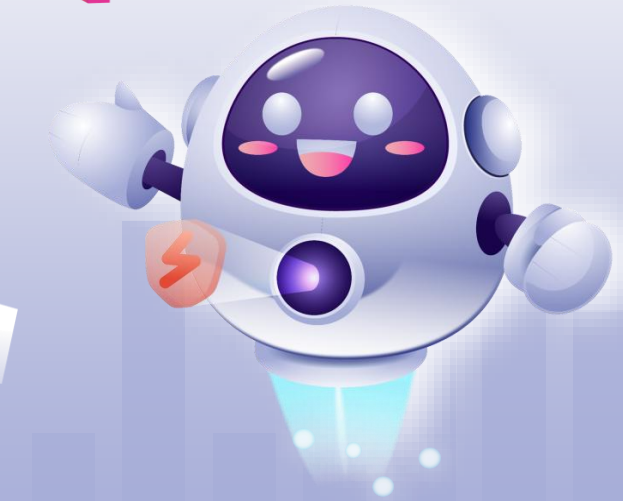
3 апреля 2026 г.

День открытых дверей РМА

Эффективные стратегии подготовки к ГИА



Региональный методист, учитель информатики МАОУ СОШ №92 города Тюмени
Мальцева Ирина Анатольевна

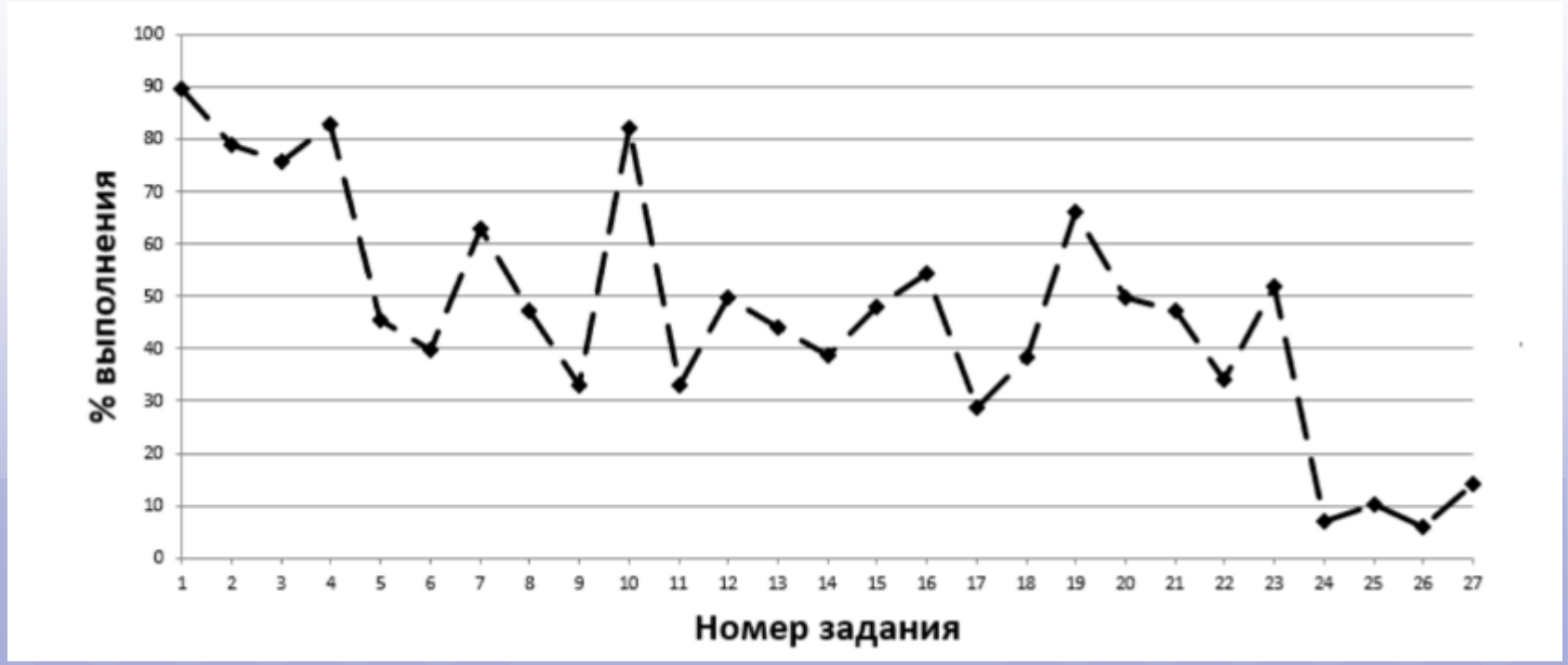
ЕГЭ по информатике

Всего в экзаменационную работу входит 27 заданий, которыми охватываются следующие содержательные разделы курса информатики:

Информация и ее кодирование	Элементы теории алгоритмов
Моделирование и компьютерный эксперимент	Программирование
Системы счисления	Обработка числовой информации
Логика и алгоритмы	Технологии поиска и хранения информации

Диагностические возможности данной экзаменационной модели проверять соответствие уровня подготовки участников экзамена к предметным результатам.

Средние проценты выполнения заданий:



Что можно сказать из анализа статистики:

01 Самыми сложными остаются **24** и **26** задания.

02 Логика и комбинаторика — слабое место большинства выпускников.

03 Работа с электронными таблицами — обязательный навык для средней оценки.

04 Программирование остается критически важным – без него невозможно получить высокий балл.

Рекомендации ученикам и родителям для подготовки к ЕГЭ по информатике:

- Не откладывайте подготовку на последний год. Информатика — это не тот предмет, который можно освоить за последние пару месяцев!
- Регулярно пишите код (минимум 2-3 раза в неделю), уделяйте время решению нестандартных логических задач.
- Осваивайте LibreOffice Calc (замена MS Excel). Помимо профильных заданий на знание табличного редактора, многие другие задачи можно решить с его помощью.
- Отслеживайте изменения формулировок самых решаемых заданий - большая вероятность, что на экзамене могут внести изменения в условия задачи.

базовый уровень сложности

Задание 1

Проверяет навыки анализа и интерпретации информации, представленной в виде схем, таблиц или матриц смежности

Описание задания

Задания №1 (по графам) из ЕГЭ по информатике проверяет навыки анализа и интерпретации информации, представленной в виде схем, таблиц или матриц смежности. Они могут включать:

- определение соответствия вершин и номеров в таблице;
- вычисление сумм длин дорог между пунктами;
- нахождение количества рёбер, степени вершины, маршрутов или других свойств графа.

Условие задачи

На рисунке показана схема дорог между населёнными пунктами (граф), а в таблице – матрица смежности, где строки и столбцы соответствуют каждому пункту. Звёздочка (*) или числовое значение в ячейке таблицы указывает на наличие дороги или её протяжённость между двумя пунктами.

Требуется **определить указанные свойства графа** – например, номера вершин, длину дороги, сумму протяжённостей между пунктами или количество соединений. Результат записывается в виде числа, набора чисел или другого формата, указанного в условии.

Основные понятия, необходимые для решения:

Схема (граф) – вершины обозначают пункты, рёбра — дороги между ними.

Таблица (матрица смежности) – N × N таблица, где ячейка показывает наличие дороги или её длину.

Неориентированный граф – дорога может использоваться в любом направлении; иногда встречаются ориентированные графы, где направление имеет значение.

Для успешного решения важно уметь:

анализировать связи на графе;

сопоставлять данные схемы и таблицы;

выполнять вычисления с числами, если указаны длины

Подход к решению подобных задач можно разделить на несколько шагов:

1. Анализ графа:
Определите степень каждой вершины – количество соединений с другими пунктами.
Если указаны длины дорог, запишите их отдельно для каждой пары вершин.
Найдите уникальные вершины или дороги – они помогут быстрее сопоставить схему и таблицу.

2. Сопоставление данных схемы и таблицы:
Сравните связи между вершинами на схеме и данные в таблице смежности.
Для вершин, соединённых с известными пунктами, определите недостающие свойства (номера вершин в таблице, протяжённость дорог, количество соединений или маршрутов).

3. Вычисление нужных величин:
Если требуется сумма протяжённостей, сложите длины нужных рёбер.
Если нужно подсчитать маршруты, соединения или степени вершин, отметьте все учтённые рёбра, чтобы не ошибиться.
Используйте таблицу смежности для проверки правильности вычислений.

4. Запись ответа. Записывайте результат в формате, указанном в условии:
целое число (например, сумма длин дорог);
набор чисел в порядке возрастания (например, номера вершин);
иные форматы, если это указано в задаче.

Советы для новичков:

- Начинайте с уникальных вершин или рёбер, чтобы быстрее определить сопоставления.
- Всегда перепроверяйте данные на схеме и в таблице, чтобы не допустить ошибок.
- Для задач с длинными путями или суммами используйте пометки или таблицу, чтобы не пропустить ни одного ребра.
- Этот алгоритм универсален: подходит для всех заданий по графам на ЕГЭ, включая сопоставление вершин, вычисление сумм длин дорог и подсчёт соединений.

02

базовый уровень сложнос

Задание 2

Проверяет умение строить таблицы истинности и работать с логическими выражениями

Python

Алгоритм решения

Решение задачи в Python включает 5 шагов: преобразование функции, построение таблицы истинности и её сопоставление с исходной для определения соответствий переменных.

Подробное пошаговое решение задания в **Python**

Шаг 1. Разбор логических операций

¬ (отрицание) – в Python записывается как not;

→ (импликация, или следование) – в Python её можно заменить на <=;

∨ (логическое «ИЛИ») – записывается в Python как or;

≡ (логическое равенство или эквивалентность) – выражается в Python как ==.

Подробное пошаговое решение задания в **Python**

Шаг 1. Разбор логических операций

Также в подобных задачах могут встречаться другие операции, которые будут записываться следующим образом:

∧ (логическое «И»), заменяется на and;

↓ (стрелка Пирса, NOR) – записывается как not (a or b);

≡ (штрих Шеффера, NAND) – реализуется как not (a and b);

⊕ (исключающее «ИЛИ», XOR) – выражается через ^ или !=.

Подробное пошаговое решение задания в **Python**

Шаг 2. Преобразование функции в Python

Используя логические операторы, логическую функцию F переписать в виде Python-выражения.

Добавить == 0, если по условию функция должна быть ложной (то есть равной 0) для всех строк таблицы истинности или == 1, если по условию функция должна быть истинной (то есть равной1) для всех строк таблицы истинности .

Шаг 3. Создание таблицы истинности

Далее создать таблицу истинности с помощью Python, перебирая все возможные значения переменных w, x, y и z. Для этого воспользуйтесь вложенными циклами, которые создают все комбинации из 0 и 1 для каждой переменной.

Подробное пошаговое решение задания в **Python**

Как работает код:

1.Заголовок таблицы. print('w x y z') выводит заголовок, который делает таблицу наглядной и обозначает, какие переменные соответствуют каждому столбцу в таблице истинности.

2.Вложенные циклы. Каждый цикл for проходит по всем возможным значениям (0 и 1) для переменных w, x, y, и z. Это реализовано через range(2), который генерирует два значения – 0 и 1. Данный подход создаёт все возможные комбинации значений для четырёх переменных, в итоге давая 2^4 = 16 вариантов.

3.Проверка условия функции F. Для каждой комбинации значений w, x, y, z код вычисляет значение логической функции F, заданной формулой. Если выражение равно 0, то функция F принимает значение "ложь", Если выражение равно 1, то функция F принимает значение «истина«. Текущие значения переменных выводятся с помощью print(w, x, y, z).

Подробное пошаговое решение задания в **Python**

Шаг 4. Анализ полученной таблицы и убирания из неё тех строк которые не соответствуют исходной

После запуска программы получаем таблицу истинности, которую можно использовать для определения соответствия переменных столбцам исходной таблицы задачи. Таблица, полученная в результате выполнения кода, может содержать больше строк, чем таблица в условии. В этом случае нужно посмотреть какие строчки не соответствуют исходной таблицы и просто их убрать из полученной таблицы.

Шаг 5. Анализ и сопоставление с исходной таблицей

Для точного сопоставления переменных с данными из условия ориентируемся не на порядок значений, а на количество нулей и единиц в каждом столбце и строке. Это помогает определить, какая переменная соответствует каждому столбцу.

Types of cybersecurity threats

Phishing

Despite being red, Mars is actually a cold planet

Ransomware

Venus has extremely high temperatures

Malware

Jupiter is the biggest planet in the Solar System

Hacking

Saturn is a gas giant and has several rings

базовый уровень сложности

Задание **3**

Проверяет умение работать с реляционными базами данных в Excel. В этом задании требуется находить и анализировать информацию с помощью встроенных функций и формул

Основные понятия баз данных

База данных (БД) – это организованное хранилище данных, позволяющее эффективно их сохранять, управлять и извлекать. Основные признаки БД: структурированность, связность и доступность для обработки.

Система управления базами данных (СУБД) – программное обеспечение, используемое для создания, управления и работы с БД.

Модели данных описывают способ организации информации. Наиболее распространённая – **реляционная модель**, где данные хранятся в таблицах, связанных между собой.

Таблица БД состоит из **полей** (столбцов) и **записей** (строк). Поле определяет атрибут данных с указанным **типом поля** (число, текст, дата и др.), а запись хранит конкретные значения.

Ключевое поле – уникальный идентификатор записи. **Первичный ключ** однозначно определяет строку в таблице, а **внешний ключ** связывает записи разных таблиц, указывая на первичный ключ другой таблицы.

Связи между таблицами в реляционной модели бывают разных типов, и понимание этих связей важно для правильного проектирования базы данных

Один к одному (1:1): В этой связи каждая запись в одной таблице может быть связана только с одной записью в другой таблице, и наоборот. Пример: таблица с информацией о сотрудниках и таблица с их паспортными данными. Каждый сотрудник имеет только один паспорт, и каждому паспорту соответствует только один сотрудник.

Один ко многим (1:N): Одна запись в первой таблице может быть связана с несколькими записями во второй таблице, но каждая запись во второй таблице может быть связана только с одной записью в первой таблице. Пример: таблица клиентов и таблица заказов. Один клиент может сделать много заказов, но каждый заказ связан только с одним клиентом.

Связи между таблицами в реляционной модели бывают разных типов, и понимание этих связей важно для правильного проектирования базы данных

Многие ко многим (M:N): Записи в обеих таблицах могут быть связаны с несколькими записями в другой таблице. Чтобы реализовать такую связь, часто используется промежуточная таблица, которая хранит соответствие между записями обеих таблиц.

Пример: таблица студентов и таблица курсов. Каждый студент может посещать несколько курсов, и каждый курс может включать несколько студентов.

Эти связи позволяют организовать данные в БД таким образом, чтобы минимизировать дублирование информации и обеспечивать её целостность.

Функция ВПР: быстрый поиск и сопоставление данных

Функция ВПР (или VLOOKUP) – важный инструмент электронных таблиц, который помогает находить данные в одном столбце и извлекать связанные значения из другого. Она особенно полезна при анализе баз данных, объединении таблиц и автоматизации работы с большими наборами информации.

Что делает ВПР? ВПР (от англ. Vertical Lookup – вертикальный поиск):

1. Находит заданное значение в **первом столбце** диапазона данных.
2. Возвращает соответствующее значение из **другого столбца той же строки**.

Функция работает только по вертикали (строкам), что и отражено в её названии.

Синтаксис функции ВПР:

ВПР(Искомое_значение; Диапазон_таблицы; Номер_столбца; [Тип_совпадения])

Искомое_значение

значение, которое нужно найти в первом столбце указанного диапазона

Диапазон_таблицы

область данных, в которой выполняется поиск (включая первый столбец с искомыми значениями)

Номер_столбца

номер столбца диапазона, из которого возвращается результат (нумерация начинается с 1)

Тип_совпадения

(опционально) – укажите ЛОЖЬ для точного совпадения (рекомендуется для анализа данных) или ИСТИНА для приближенного (по умолчанию используется редко)

Задание № 4

базовый уровень сложности

*Направлено на проверку знаний в области кодирования и декодирования информации с использованием неравномерных двоичных кодов. В этом задании ключевым является понимание **условия Фано**, которое гарантирует правильность и уникальность кодирования.*

Условие Фано

Условие Фано – это принцип, который гарантирует уникальность и однозначность кодов в системе. Оно заключается в следующем:

- **Уникальность кодов.** Каждый код должен быть уникальным и не может быть префиксом другого кода в данной системе. Это означает, что ни одно кодовое слово не может быть началом другого.
- **Однозначность декодирования.** Благодаря этому свойству любая последовательность закодированных символов может быть расшифрована без ошибок или неоднозначностей. Например, если два кодовых слова – "101" и "1010", то второй код не может быть использован, так как он начинается с первого.

Условие Фано

Условие Фано устраняет такие проблемы, как перекрытие кодов, и обеспечивает корректное и безопасное кодирование и декодирование данных. Это свойство особенно важно в системах передачи данных, где точность и надежность расшифровки критичны.

Дерево Фано и основные принципы его построения

Дерево Фано – это наглядный и эффективный метод для решения задач, связанных с созданием неравномерных двоичных кодов. Оно позволяет строить уникальные коды, которые соответствуют **условию Фано**:

Ни один код не является началом другого.

Это свойство гарантирует, что закодированный набор символов можно однозначно декодировать, исключая любые неоднозначности.

Почему следует использовать дерево Фано?

Метод дерева Фано привлекателен благодаря своей логике, наглядности и эффективности. Он делает процесс кодирования понятным и удобным даже для сложных задач.

Наглядность

Н

Пошаговое построение дерева показывает процесс кодирования, позволяя наглядно видеть назначение уникальных кодов каждому символу и проверять их правильность.

Удобство

У

Сначала размещаются символы с известными кодами, а затем дерево достраивается для остальных. Этот структурированный подход исключает хаос и облегчает процесс кодирования.

Эффективность

Э

Логичная последовательность действий минимизирует ошибки и ускоряет решение задач. Метод позволяет быстро и точно найти ответ, сводя процесс к простым рассуждениям.

Основные принципы построения дерева Фано:

Дерево начинается с вершины, от которой отходят две ветви, поскольку оно является двоичным. Левой ветви, например, присваивается бит 0, а правой – 1.

Если ветвь занята символом, она блокируется и больше не участвует в разветвлениях. Это необходимо для соблюдения уникальности кодов и предотвращения пересечений.

Начало построения

Разветвление

Заполнение и блокировка ветвей

Достроение дерева

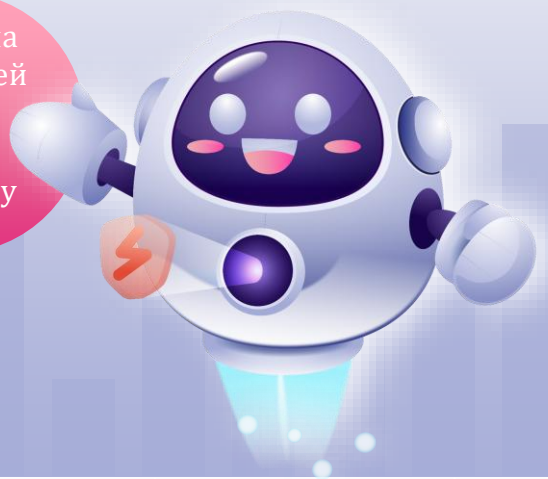
Каждый узел дерева может порождать две новые ветви. Ветвь, уходящая влево, например, по аналогии обозначается битом 0, а правая – 1.

После размещения символов с известными кодами дерево достраивается для кодирования остальных букв. Новые ветви продолжают следовать принципу двоичного разветвления: например, 0 – для левого направления и 1 – для правого.

базовый уровень сложности

Задание **5**

Посвящено числовым алгоритмам. Связано с переводом числа из десятичной системы счисления в другую, его модификацией и переводом обратно. Вопросы могут касаться поиска максимального, минимального результата алгоритма или определения исходного числа, которое приводит к заданному результату.



Основная идея

Перебрать множество начальных значений алгоритма, вычислить для всех конечный результат и получить ответ.

Главное преимущество такого способа — не нужно пользоваться сложной аналитикой. Задача сводится к тому, чтобы корректно записать алгоритм, указанный в задании, на языке программирования. Основная сложность такого решения связана с тем, что действия алгоритма могут быть разнообразными: дописывание символов, их замена, вычисление остатков деления, нахождение суммы цифр и перевод числа в другую систему счисления.

```
# Перевод в двоичную
b = bin(x)[2:]
b = f'{x:b}'
b = format(x, 'b')
# Перевод в восьмеричную
b = oct(x)[2:]
b = f'{x:o}'
b = format(x, 'o')
# Перевод в шестнадцатеричную
b = hex(x)[2:]
b = f'{x:x}'
b = format(x, 'x')

# Перевод в десятичную
a = int(x, b)
# где b — основание системы счисления (от 2 до 36), в которой записано число x (в виде строки)
```

Основная идея

Для перевода чисел в произвольную систему счисления рекомендуем использовать собственную функцию. Вот две её реализации: с хранением результата в виде списка и в виде строки:

```
def to_base(x, b):
s = []
while x > 0:
    s.append(x % b)
    x //= b
s.reverse()
return s
```

```
def to_base(x, b):
s = ''
while x > 0:
    s = str(x % b) + s
    x //= b
return s
```

Функция со строками будет корректно работать только с основаниями до 10.

Функция, которая возвращает результат в виде списка, будет корректно работать со всеми основаниями систем счисления (но вместо буквенных цифр будут числа).

Советы по решению:

- Следуйте алгоритму. Выполняйте шаги строго в указанном порядке, не упуская деталей.
- Внимательно читайте условие. Обратите внимание, какой результат требуется указать в ответе (в данном случае – минимальное число N в десятичной системе).
- Навыки программирования. Для выполнения задания важно уверенно пользоваться одним из языков программирования. Глубоких знаний информатики не требуется.

Основная идея

Следующий важный навык — работа со строками. Чтобы решить задание № 5, нужно уметь дописывать числа в начало или в конец строки. Помните, что строки в Python — неизменяемые объекты, поэтому в каждом случае нужно будет создавать новые. Необходимо также знать, как работают срезы.

```
s = '012345'
# строка со второго символа
s[1:]
'12345'
# строка с третьего символа
s[2:]
'2345'
# строка без последнего символа
s[:-1]
'01234'
# строка без двух последних символов
s[:-2]
'0123'
```

Для закрепления материала предлагаю три подборки задач:

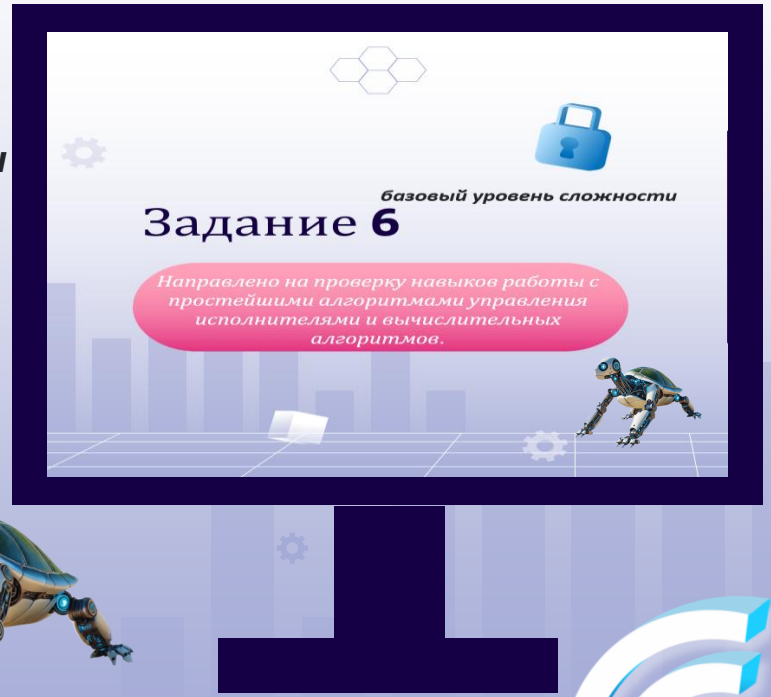
- отработайте базовые алгоритмы модификации записей на **задачах с двоичной системой счисления**
- реализуйте собственную функцию перевода и более сложную логику обработки строк на **задачах с другими системами счисления**
- освойте работу с цифрами числа через целочисленную арифметику и операции деления с остатком на **задачах с десятичной системой счисления**

Как решать задание № 5 из ЕГЭ по информатике аналитически (статья Андрея Рогова): https://education.yandex.ru/ege/inf/article/119-kak-reshat-zadanie-5-iz-yege-po-informatike-analiticheski

базовый уровень сложности

Задание 6

Направлено на проверку навыков работы с простейшими алгоритмами управления исполнителями и вычислительных алгоритмов.



Описание задачи

Исполнитель «Черепаха» выполняет команды на плоскости с декартовой системой координат. В начальный момент времени Черепаха находится в точке (0, 0), её голова направлена вдоль положительного направления оси OY, а хвост опущен, что означает, что при движении она оставляет след на плоскости в виде линии.

Черепаха умеет выполнять **следующие команды**:

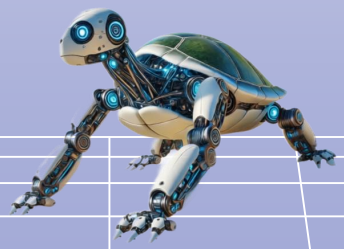
- Поднять хвост – черепаха перестаёт рисовать линию, перемещаясь без следа;
- Опустить хвост – черепаха переходит в режим рисования, оставляя след;
- Вперёд n – черепаха перемещается на n единиц в текущем направлении;
- Назад n – черепаха перемещается на n единиц в противоположном направлении;
- Направо m – черепаха поворачивается на m градусов по часовой стрелке;
- Налево m – черепаха поворачивается на m градусов против часовой стрелки.

Дополнительно используется **конструкция:**

Повтори k [Команда1 Команда2 ... КомандаS] – черепаха повторяет последовательность из S команд k раз.

Об аналитическом решении задания № **6** можно прочитать здесь
https://education.yandex.ru/ege/article/2-reshenie-zadanii-№-6-po-informatike-analiticheskim-metodom---vruchnuiu

О программном решении задания № **6** можно прочитать здесь
https://education.yandex.ru/ege/inf/article/160-reshenie-zadanii-6-po_informatike-kodom



базовый уровень сложности

Задание № 7

Проверяет навыки расчёта объёма памяти, необходимого для хранения графической или звуковой информации, а также времени, требуемого для передачи данных по каналу связи

Полезные ссылки на статьи:

Кодирование графики в задании ЕГЭ по информатике № 7

https://education.yandex.ru/ege/inf/article/197-kodirovanie-grafiki-v-zadanii-yege-po-informatike-7

Кодирование звука в задании ЕГЭ по информатике № 7

https://education.yandex.ru/ege/inf/article/198-kodirovanie-zvuka-v-zadanii-yege-po-informatike-7

Передача информации в задании номер №7

https://education.yandex.ru/ege/inf/article/199-peredacha-informatsii-v-zadanii-nomer-7

базовый уровень сложности

Задание № 8

Проверяет знания базовых понятий и методов, связанных с измерением количества информации

Описание задания

Для успешного выполнения этого задания требуется понимание ключевых терминов, таких как алфавит сообщения, длина сообщения, мощность алфавита и объём информации.

Способы организации программы

Основная концепция задания: имеется некое множество объектов (обычно слова или числа), из которого необходимо выделить подмножество, обладающее определёнными признаками. Вычислительные мощности обычного компьютера позволяют перебрать все элементы множества и посчитать, сколько из них подходят под критерии. Для перебора элементов можно использовать два способа:

1. Более простой для понимания — вложенные циклы. В заданиях встречаются четырёхбуквенные слова, пятизначные числа и т. п. Количество символов в слове или строке и будет определять, сколько циклов необходимо. Как правило, количество варьируется в пределах от 4 до 6
2. Более простой для написания — функции модуля itertools. На самом деле, достаточно одной из них, product(), но в некоторых случаях её можно заменить permutations().

Общая структура программы выглядит та

1. Заводим необходимые счётчики (для количества подходящих слов или чисел, для нумерации)
2. Организуем перебор всех комбинаций
3. Проверяем, подходит ли под условие текущая комбинация
4. Меняем значения счётчиков
5. Выводим ответ

Модуль **itertools**

Встроенный в Python модуль itertools даёт доступ к итераторам, которые можно эффективно перебирать с помощью цикла. В модуле представлено множество итераторов, но на ЕГЭ обычно бывают полезны два: product() и permutations().

Product()

Официальная документация даёт довольно сложное определение этому итератору. Product() возвращает декартово произведение итерируемых объектов, подаваемых на вход.

Другими словами, с помощью product() можно найти произведение множества X и множества Y. То есть это множество, содержащее все упорядоченные пары (x, y), в которых x принадлежит множеству X, а y принадлежит множеству Y.

Permutations()

Представляет собой итератор, содержащий все возможные перестановки элементов.

базовый уровень сложности

Задание 9

Проверяет умение анализировать числовую информацию в электронных таблицах

Сюжет задания

Типичный сюжет задания предполагает, что исходные данные содержат от 4 до 7 чисел в одной строке, количество строк — от 3000 до 16 000. Необходимо определить, сколько строк будет подходить под определённое условие. Среди условий можно выделить следующие:

- выделить максимальное или минимальное число и сравнить его (их) с остальными;
- числа можно разбить на пары по определённому критерию;
- в строке повторяется определённое количество чисел;
- сравнить повторяющиеся и неповторяющиеся числа.

Основные ошибки

Решение этого задания довольно объёмное и состоит из нескольких этапов. Следует внимательно контролировать каждый шаг. К типичным ошибкам можно отнести следующие:

- неверные ссылки в формулах;
- невнимательная проверка условий в решении;
- учёт не всех возможных ситуаций.

Электронные таблицы — эффективный инструмент для обработки больших массивов числовых данных. Решение упрощается, если использовать дополнительные столбцы, а применение фильтров существенно ускоряет процесс отбора подходящих строк.

Рекомендации:

- тщательно анализируйте условия задачи до начала решения;
- продумайте систему промежуточных вычислений;
- проверяйте корректность формул на первых строках;
- внимательно следите за нумерацией строк, если вопрос про номер строки.

Подборки задач:

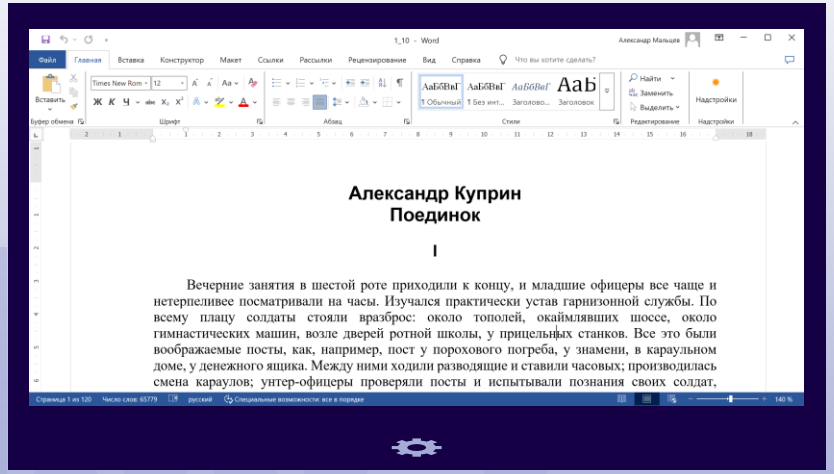
простые задачи для знакомства с № 9

интересные задачи на подумать

базовый уровень сложности

Задание **10**

Оценивает навыки информационного поиска с использованием текстового процессора



О задании

Долгое время в качестве основного текстового редактора использовался Microsoft Office Word. Но с 2026 года его больше нет в списке рекомендуемого программного обеспечения.

Теперь все текстовые файлы для 10 задания имеют расширение «odt» (OpenDocument Text) и открываются в свободно распространяемых текстовых редакторах. Одним из таких редакторов является LibreOffice Writer.

Полезная статья «Поиск в LibreOffice Writer»: https://future-step.ru/tutor/task-10-1/

О задании

Задание 10 официально не разделяется ни на какие типы. Для наглядности выделю **два** основных различия в формулировках.

Задания **первого типа** имеют в своей формулировке фразу «*отдельное слово*». То есть в них надо пользоваться обычным поиском с параметром «*Слова целиком*». Зачастую в таких заданиях требуется не учитывать другие формы искомого слова, что упрощает задачу в разы: https://future-step.ru/tutor/task-10-3/

Задания **второго типа** обычно содержат фразу «*в составе других слов*». В таком случае надо действовать по простому **алгоритму**: **общее** количество упоминаний искомого слова и **отнимать** от этого количества только те упоминания, в которых фигурирует это **слово целиком**, то есть как отдельное слово: https://future-step.ru/tutor/task-10-4/

повышенный уровень сложности

Задание № 11

Проверяет умение подсчитывать информационный объём сообщения

Важные особенности решения:

- Подсчёт бит на символ: необходимо учитывать размер алфавита и определять минимальное количество бит для кодирования одного символа.
- Перевод в байты и мегабайты: важно правильно округлять значения при переходе из бит в байты, а затем в мегабайты.
- Округление: все вычисления объёма памяти проводят с округлением до целого числа мегабайт.

Такой подход не только проверяет знания теории информации, но и требует внимательности к деталям при выполнении арифметических операций.

Краткая теория

На выполнение этого задания обычно отводится около трёх минут. Главное — понять суть вопроса. Часто речь идёт о посимвольном кодировании или переводе между единицами измерения информации. На каждом этапе решения следите за тем, чтобы использовать правильные единицы: сначала обычно считают количество бит, затем переводят в байты, а при необходимости — в килобайты или мегабайты.

Ещё одна причина, по которой выполнение задания может занять больше времени, — встреча со сложными прототипами, требующими более эффективных методов решения. В таких случаях бывает полезно подключить программирование для автоматизации расчётов.

Краткая теория

В информатике различают два способа кодирования: равномерное и неравномерное. При неравномерном кодировании кодовые слова могут иметь разную длину. Работа с таким кодом требуется в задании номер 4. А в задании номер 11 речь идёт о равномерном кодировании, когда все кодовые слова имеют равную длину.

Прежде всего необходимо разобраться с понятием мощности алфавита — это количество разных символов, которые используются для кодирования.

Зная мощность алфавита, можно определить, сколько двоичных разрядов потребуется для кодирования одного символа. Поскольку мощность алфавита часто не совпадает с целой степенью двойки, для расчёта используют формулу $N \leq 2^i$, где N — мощность алфавита, i — количество бит, требуемое для кодирования одного символа алфавита.

Краткая теория

Сообщения составляются из символов выбранного алфавита. В заданиях это могут быть пароли, серийные номера или идентификаторы; каждый объект включает несколько символов.

Чтобы рассчитать объём информации, необходимо умножить количество символов в сообщении на объём информации, приходящийся на один символ: $I = k \times i$.

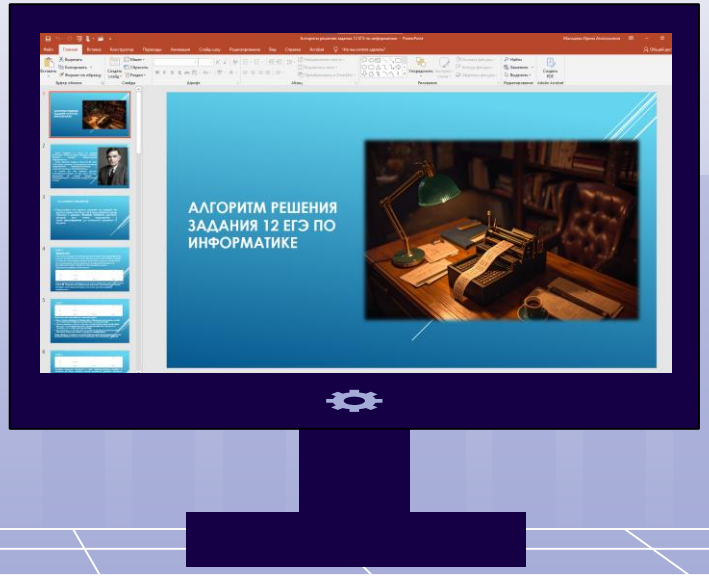
Рекомендации:

- внимательно читать условия и разбираться, в каких единицах требуется дать ответ;
- проверять, что найденное число соответствует условию задачи: это объём всех идентификаторов, максимальное или минимальное значение мощности алфавита или длины информационного объекта;
- уделять внимание величинам с одинаковыми единицами измерения информации: мощность алфавита и количество символов, вес одного символа и вес всего сообщения;
- при переводе бит в байты результат нужно правильно округлять.

повышенный уровень сложности

Задание **12**

Проверяет умение анализировать и реализовывать алгоритмы с использованием фиксированного набора команд



Полезная ссылка на презентацию «Алгоритм решения задания 12 ЕГЭ по информатике»: https://disk.yandex.ru/d/yUw26t3n6QJRHg

повышенный уровень сложности

Задание **13**

Проверяет твоё умение работать с IP-адресами и масками сети

В этом задании могут попасться разные форматы вопросов, но все они крутятся вокруг сетевых адресов и масок. Вот основные виды задач:

- **Подсчёт количества адресов в сети.** Нужно определить, сколько всего IP-адресов входит в данную сеть по заданной маске. В некоторых вариантах могут попросить посчитать именно количество доступных для узлов адресов (то есть исключить адрес сети и широковещательный адрес).
- **Определение маски сети.** Дают IP-адрес узла и адрес сети, а тебя просят найти маску подсети. Чаще всего спрашивают конкретный байт маски – например, третий слева.
- **Нахождение адреса сети или широковещательного адреса.** Нужно по IP-адресу и маске вычислить, какой адрес у самой сети или какой адрес является широковещательным.
- **Проверка принадлежности IP-адреса к сети.** Дают IP-адрес и сеть с маской, а задача – определить, входит ли этот IP-адрес в указанную сеть.

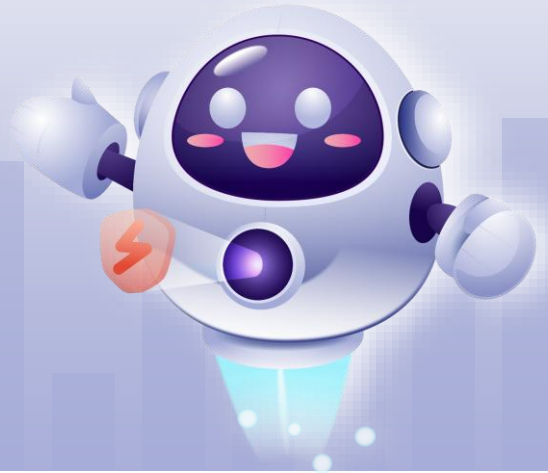
Чтобы ученики могли успешно его решать, они должны понимать, как устроены IP-адреса, что такое маска сети, как делится адрес на сеть и узел, и как вычислять адрес сети.

№ 13 кодом. Библиотека ipaddress
https://education.yandex.ru/ege/inf/article/280-13-kodom-biblioteka-ipaddress

повышенный уровень сложности

Задание **14**

Работать с **позиционными системами счисления** – такими как двоичная, восьмеричная, шестнадцатеричная и другие



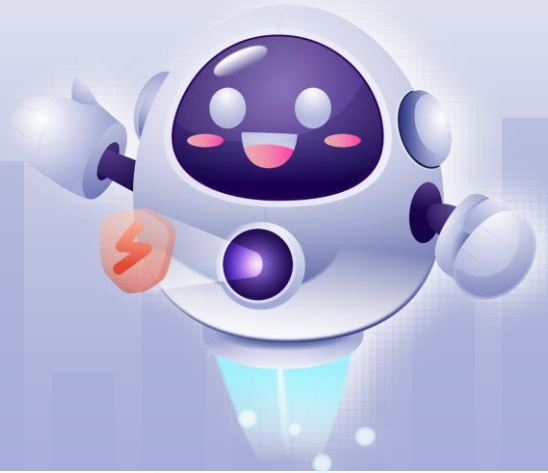
Программное решение задания 14 на языке Python

https://education.yandex.ru/ege/inf/article/267-programmnoe-reshenie-zadaniia-14-na_iazike-python

повышенный уровень сложности

Задание 15

Проверка знаний ключевых понятий, законов и приемов работы с логическими выражениями

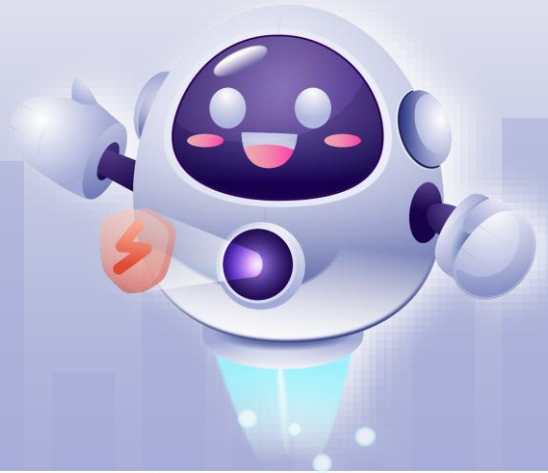


Задание № 15. Решение кодом
https://education.yandex.ru/ege/inf/article/283-zadanie-15-reshenie-kodom

повышенный уровень сложности

Задание **16**

Проверяет умение работать с рекурсивными алгоритмами и осуществлять вычисления с их помощью



№ 16 кодом. Оптимизация рекурсии
https://education.yandex.ru/ege/inf/article/281-16-kodom-optimizatsiia-rekursii

№ 16. Функции, рекурсия
https://education.yandex.ru/ege/inf/article/282-16-funktsii-rekursiia

повышенный уровень сложности

Задание 17

Проверка умений перебора последовательности целых чисел.

Основная **сложность** данного задания в его витиеватой и запутанной **формулировке**.

Наша цель – научить вдумчиво читать текст задания и **выписывать** все необходимые условия, которым должны отвечать подходящие числа.

В ответ всегда требуется дать **2 числа**:

- **количество** подходящих пар или троек чисел;
- **результат** какого-либо выражения с ними (максимальная сумма элементов, минимальная сумма квадратов и т.д.).

Разберём основные подходы для грамотной и безопасной работы с файлами, методы и функции для работы со списками и разберем алгоритм решения задания 17 на основе нескольких примеров.

Открываем файлы мы следующей конструкцией:

```
with open('file.txt') as f:
```

Как правильно извлекать данные из файла:

Есть 3 базовые функции, которые позволяют читать данные из текстового файла: read(), readline() и readlines(). Можно использовать любую из них. Так как потребуется работать с данными в виде списка целых чисел.

Можно считать все данные с помощью readlines(), которая поместит значение каждой строки в список. Но тогда придётся заново проходиться в цикле по каждому элементу этого списка и приводить его к целочисленному типу.

Более лаконичным и эффективным способом будет использование списочного включения. Дело в том, что файловые объекты в Python поддерживают протокол итерации, то есть мы можем буквально поместить этот объект в цикл for и в каждой итерации цикла получать по одной строке из файла.

В таком случае можно написать следующее списочное включение, которое позволит в цикле перебрать каждую строку из файла, привести значение этой строки к целочисленному типу и сохранить как элемент списка:

```
data = [int(line) for line in f]
```

Таким образом, весь код, который нам понадобится для чтения данных из файла и сохранения их в виде списка целых чисел представлен ниже:

```
with open('file.txt') as f:
    data = [int(line) for line in f]
```

Данные извлечены из файла и сохранены в список. Теперь нужно их как-то обработать.

Рассмотрим работу нескольких функций и методов для работы со списками.

Зачастую в 17 заданиях требуется найти максимальное или минимальное значение в списке. Для этого можно использовать функции max() и min() соответственно.

Причем это универсальные функции, которые могут работать как с числами, так и со строками. Для чисел они возвращают максимальное или минимальное значение числа, а для строк — самую длинную или короткую строку.

Если работаем со списком строк, то функция min() вернёт строку, которая является лексикографически наименьшей (то есть первой в алфавитном порядке или согласно порядку символов в Unicode).

Например среди строк: «вишня», «банан» и «апельсин» лексикографически наименьшей является строка, начинающаяся на букву «а» — «апельсин».

Если нужно вернуть количество элементов в списке, то можно использовать глобальную функцию len().

Для нахождения суммы чисел в списке целесообразно будет использовать уже готовую функцию sum().

Для формирования итогового списка, количество элементов которого надо записать в ответ, нужно добавлять в него новые значения, соответствующие всем условиям. Для добавления элементов в конец списка используется метод append().

Он вызывается у объекта списка (через точку) и принимает один аргумент — элемент, который нужно добавить в конец этого объекта.

Это все функции, которые понадобятся при решении 17 задания.

Но также стоит отметить еще один момент. При решении задания понадобится одновременно проверять пары или тройки элементов. То есть необходимо будет перебирать по 2 или 3 числа последовательно, с перекрытием.

Не помещайте в range() длину итерируемого объекта, чтобы работать с его индексами! Это очень неэффективный и времязатратный подход.

Для работы с индексами – используйте enumerate(). В 17 задании гораздо эффективнее, лаконичнее и быстрее достичь задуманной цели можно, используя функцию zip().

Функция zip() в Python позволяет объединять элементы из нескольких итерируемых объектов в кортежи. Причем каждый i-й кортеж содержит i-й элемент каждого из переданных итерируемых объектов.

Пример. Есть два списка:

Первый содержит числа от 1 до 3: [1, 2, 3]

Второй содержит только буквы: [‘a’, ‘b’, ‘c’]

Нам нужен список из кортежей, где каждой цифре соответствует буква. Реализуем это с использованием функции zip():

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']

result = zip(list1, list2)
print(list(result))

>>>[(1, 'a'), (2, 'b'), (3, 'c')]
```

Вернёмся к задаче. Логично предположить, что, передавая на вход функции zip() сначала весь список, затем список без первого элемента и список без второго элемента, мы можем получить за первую итерацию первую тройку элементов. И с каждой следующей итерацией значения из списка будут постепенно перебираться в виде таких троек.

```
nums = [1, 2, 3, 4, 5]

for x, y, z in zip(nums, nums[1:], nums[2:]):
    print(x, y, z)

>>>1 2 3
>>>2 3 4
>>>3 4 5
```

Именно такой подход используйте в решениях 17 задания.

повышенный уровень сложности

Задание **18**

Проверка умений использовать электронные таблицы для обработки целочисленных данных

О задании

В задании 18 ЕГЭ по информатике дано некоторое квадратное поле, по которому может перемещаться виртуальный робот-сборщик монет. Поле это представлено в виде электронной таблицы, где в каждой ячейке записано число от 1 до 100 – количество монет, которое достанется нашему роботу, если он попадёт на эту ячейку.

На перемещения робота накладываются некоторые ограничения. Так, в зависимости от задания, он может перемещаться только по определённым **направлениям**: вправо и вниз, влево и вверх или еще какие-либо вариации этих команд. Также на самом поле присутствуют стены, роль которых выполняют границы ячеек. Соответственно, проходить сквозь этим стены робот не может.

Правда в редких случаях авторы наделяют робота способностями к телепортации. Однако, такие формулировки заданий с телепортациями робота не встречаются в официальных вариантах ЕГЭ.

В результате данного задания необходимо подсчитать количество монет, которое может собрать робот, перемещаясь из начальной ячейки в конечную. Конечной будет называться ячейка, с обеих сторон ограниченная стенами, преодолеть которые робот не в состоянии. Например, при наличии команд «вправо» и «вниз» такой ячейкой будет правая нижняя.

В ответ надо записать два числа: максимальную и минимальную сумму, которую может собрать робот за время своего путешествия по ячейкам. Иными словами, нам здесь требуется найти оптимальный путь в матрице (таблице значений). А решение подобных задач сводится к динамичес

Динамическое программирование – это метод решения сложных задач путём ***разбиения*** их на более простые подзадачи и ***запоминания*** результатов этих подзадач, чтобы не решать их повторно.

В динамическом программировании существуют три основных принципа:

1. Одну и ту же подзадачу несколько раз, мы один раз решаем её и запоминаем ответ. Оптимальность подзадач: если мы знаем оптимальное решение для маленькой части задачи, мы можем использовать его для решения большей части.

2. Перекрывающиеся подзадачи: одни и те же маленькие задачи встречаются много раз при решении большой задачи.

3. Запоминание результатов: вместо того чтобы решать.

Алгоритм решения любого задания 18:

1. Ниже исходной таблицы начинаем строить новую, с суммами значений.

2. Значение начальной ячейки переносим. Строку и столбец с ней заполняем по формуле: <Значение предыдущей ячейки> + <Значение соответствующей ячейки исходной>.

3. Для остальных значений в таблице используем формулу МАКС(<Значение ячейки с одной стороны>,< Значение ячейки с другой стороны >)+ <Значение соответствующей ячейки исходной>. Если двигаемся только вправо и вниз то формула следующая: МАКС(<Значение ячейки слева>,< Значение ячейки сверху >)+ <Значение соответствующей ячейки исходной>

4. Добавляем стены из исходной таблицы с помощью функции «Формат по образцу».

5. Отмечаем конечные точки. Если двигаемся только вправо и вниз, то это точки со стенами справа и снизу.

6.Удаляем значения с тех ячеек, в которые Робот не может попасть. Например, сверху от стены.

7. Редактируем формулы вдоль стен (убираем функцию МАКС() и оставляем только сумму двух ячеек).

8. Выбираем наибольшее значение из конечных точек.

9. Меняем формулу МАКС() на МИН() с помощью инструмента «Заменить…».

10.Выбираем наименьшее значение из конечных точек.

базовый уровень сложности

Задание **19**

Проверка умения анализировать алгоритм логической игры

повышенный уровень сложности

Задание **20**

Проверка умения найти выигрышную стратегию игры

высокий уровень сложности

Задание **21**

Проверка умения построить дерево игры по заданному алгоритму и найти выигрышную стратегию

Полезные статьи с разбором заданий

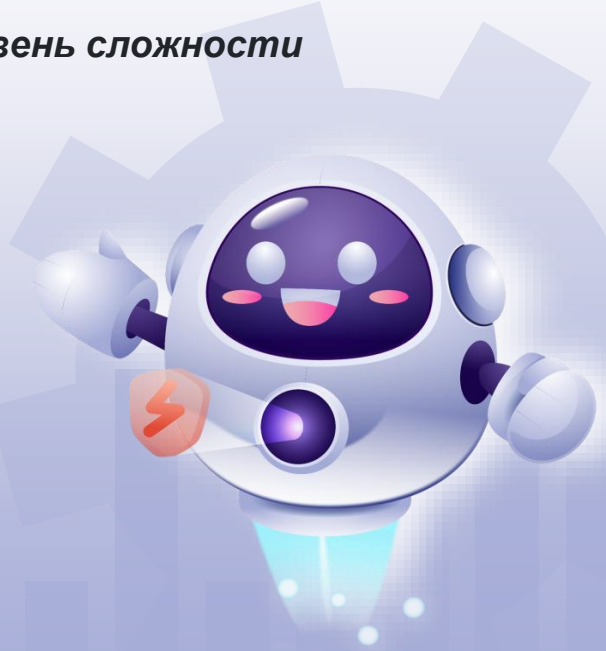
Решение заданий 19-21 на теорию игр в электронных таблицах
https://education.yandex.ru/ege/inf/article/271-reshenie-zadanii-na-teoriiu-igr-v-elektronnikh-tablitsakh

Рекурсивное программное решение заданий 19-21 на теорию игр в ЕГЭ
https://education.yandex.ru/ege/inf/article/274-rekursivnoe-programmnoe-reshenie-

повышенный уровень сложности

Задание № 22

Проверяет умение строить математические модели для решения практических задач



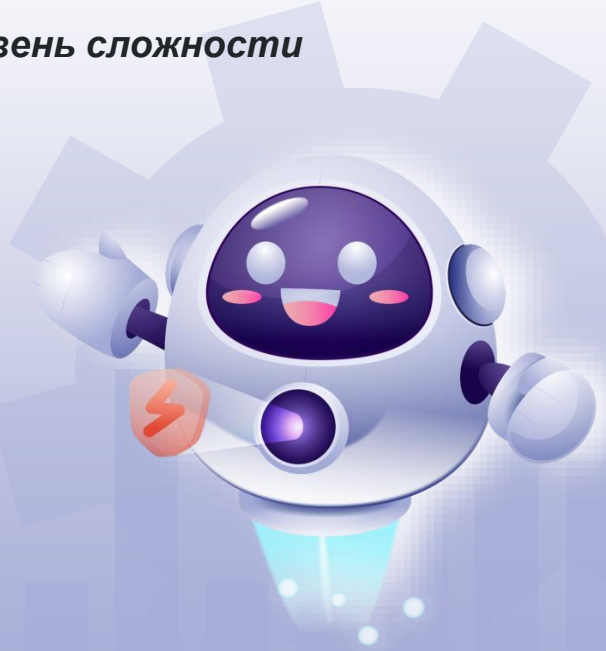
Решение задания № 22 без сдвигов процессов в электронных таблицах
https://education.yandex.ru/ege/inf/article/278-reshenie-zadaniia-22-bez-sdvigov-protsessov-v-elektronnikh-tablitsakh

Решение задания № 22 со сдвигами процессов в электронных таблицах
https://education.yandex.ru/ege/inf/article/279-reshenie-zadaniia-22-so-sdvigami-protsessov-v-elektronnikh-tablitsakh

повышенный уровень сложности

Задание № 23

Проверяет умение анализировать ход исполнения алгоритма



Задание 23 ЕГЭ по информатике направлено на проверку умений анализировать результаты исполнения алгоритма, содержащего ветвление и цикл. В данном задании снова предстоит столкнуться с динамическим программированием.

Методы динамического программирования, применялись при решении задания с «роботом» в задании 18. В котором работа велась исключительно с электронной таблицей.

В задании 23 представлен некий исполнитель, который имеет определённый набор команд. Подобные исполнители уже встречались нам: это и черепаха, и чертёжник, и редактор, и многие другие. То есть общий подход тут сохраняется: «переводим» команды с алгоритмического языка на Python.

Конкретно в 23 заданиях помимо реализации работы исполнителя еще предстоит вычислить количество возможных программ для преобразования одного числа в другое, путём математических операций.

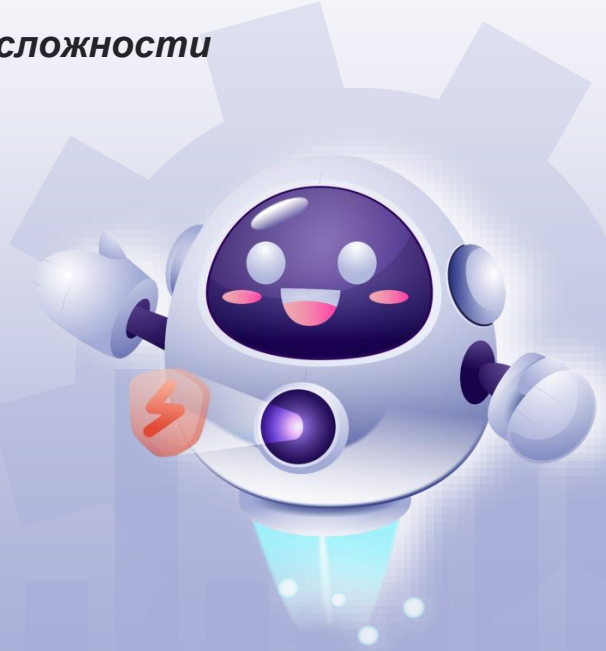
Решение задания
№ 23 ЕГЭ
по информатике
на языке Python

https://education.yandex.ru/ege/inf/article/270-reshenie-zadaniia-23-ege-po-informatike-na-iazike-python

высокий уровень сложности

Задание № 24

Проверяет умение создавать собственные программы (10-20) для обработки символьной информации



Способы решения

Первым делом здесь стоит отметить обилие способов решения 24 заданий. Чаще всего встречаются три самых эффективных:

1. Регулярные выражения
2. Метод двух указателей
3. Метод замены (replace) и разделения строки (split)

Регулярные выражения обычно используются для поиска подстроки в тексте по определённому шаблоны. Они позволяют описать шаблон такой подстроки буквально в одну строку кода. Но есть одна особенность: регулярные выражения используют собственный синтаксис. Так что для уверенного использования этого метода вам сначала нужно выучить, как представляются разные символы в регулярных выражениях, как работают проверки разного типа и как использовать квантификаторы.

Способы решения

Метод замены и разделения строки основан на простой идее: сначала мы преобразуем исходную строку, заменяя или удаляя ненужные символы, а затем разбиваем её на части. После этого остаётся только найти самую длинную часть. Этот метод интуитивно понятен и не требует знания специального синтаксиса.

Метод двух указателей использует два индекса, которые движутся по строке и отмечают начало и конец текущей подстроки. Этот подход особенно полезен для задач, где нужно подсчитывать количество определённых символов внутри подстроки. Он работает быстро даже на очень длинных строках.

Типизация

Первый тип — задания, в которых нужно найти максимальное количество идущих подряд последовательностей символов. Это могут быть как заранее заданные последовательности, например, NPO или KLMN. Так и просто некоторые буквы определённого вида: гласная + согласная, символы, представляющие числа в какой-то системе счисления и так далее. Такие задания уверенно решаются с помощью регулярных выражений.

Второй тип — задания с арифметическими выражениями. Здесь строка представляет собой математическое выражение, и нужно найти в нём подстроку с определёнными свойствами. Например, найти самое длинное корректное выражение или подвыражение с максимальным значением. И здесь не обойтись без регулярных выражений. Конечно, есть и другие способы решения, но гораздо проще и быстрее будет использование регулярных выражений.

Типизация

Третий тип — задания, в которых определённые символы не должны стоять рядом. Типичная формулировка: «Найдите самую длинную подстроку, в которой буквы A и B не стоят рядом». Для таких задач идеально подходит метод замены и разделения строки — мы находим все «запрещённые» комбинации, используем их как разделители и ищем максимальный фрагмент.

Четвёртый тип — задания, в которых символ или подстрока должны встречаться строго определённое количество раз. Например: «Найдите минимальное/максимальное количество идущих подряд символов, среди которых подстрока X встречается не менее/более N раз и при этом содержится ровно M символов Y».

Задание № **25.** Маски

В соответствии со спецификацией ФИПИ, у 25-го задания высокий уровень сложности. На его решение отведено 20 минут. Учитывая совсем небольшой объём кода, такие рамки обусловлены переборной сутью алгоритма.
Оценка сложности и подбор оптимального метода решения позволят решать задачи намного быстрее.


Применение
fnmatch

В заданиях под маской подразумевается шаблон, проверяющий строки.

В Python проверка уже реализована в модуле fnmatch.

Одноимённая функция fnmatch принимает строку и маску, после чего возвращает булево значение, указывающее, соответствует ли строка заданному шаблону.

В маске можно задавать один произвольный символ с помощью вопросительного знака (?).

Звёздочка (*) помогает задать любое количество произвольных символов (даже 0).

Задание 25

Средняя | Маркер

ФИПИ

Назовём маской числа последовательность цифр, в которой также могут встречаться следующие символы:

- символ «?» означает ровно одну произвольную цифру;
- символ «*» означает любую последовательность цифр произвольной длины; в том числе «*» может задавать и пустую последовательность.

Например, маске 123*4?5 соответствуют числа 123405 и 12300405.

Среди натуральных чисел, не превышающих 10^{10}, найдите все числа, соответствующие маске 3?12?14*5, делящиеся на 1917 без остатка.

В ответе запишите в первом столбце таблицы все найденные числа в порядке возрастания, а во втором столбце — соответствующие им результаты деления этих чисел на 1917.

Количество строк в таблице для ответа избыточно.

```
from fnmatch import fnmatch
mask = '3?12?14*5'
for n in range(1, 10**10):
    if n % 1917 == 0:
        if fnmatch(str(n), mask):
            print(n, n // 1917)
```

Оптимизация проверки на делимость

```
from fnmatch import fnmatch
mask = '3?12?14*5'
for n in range(0, 10**10, 1917):
    if fnmatch(str(n), mask):
        print(n, n // 1917)
```

fnmatch не спасает

Программа работает слишком долго...

Оптимизация перебора!

Задание 25

Простая | Маркер

Яндекс Учебник

Назовём маской числа последовательность цифр, в которой также могут встречаться такие символы:

- символ «?» означает ровно одну произвольную цифру
- символ «*» означает любую последовательность цифр произвольной длины; в том числе «*» может задавать и пустую последовательность

Например, маске 123*4?5 соответствуют числа 123405 и 12300405.

Среди натуральных чисел, не превышающих 10^{12}, найдите все числа, соответствующие маске 4?8?15?16?23, делящиеся на 123 с остатком 42.

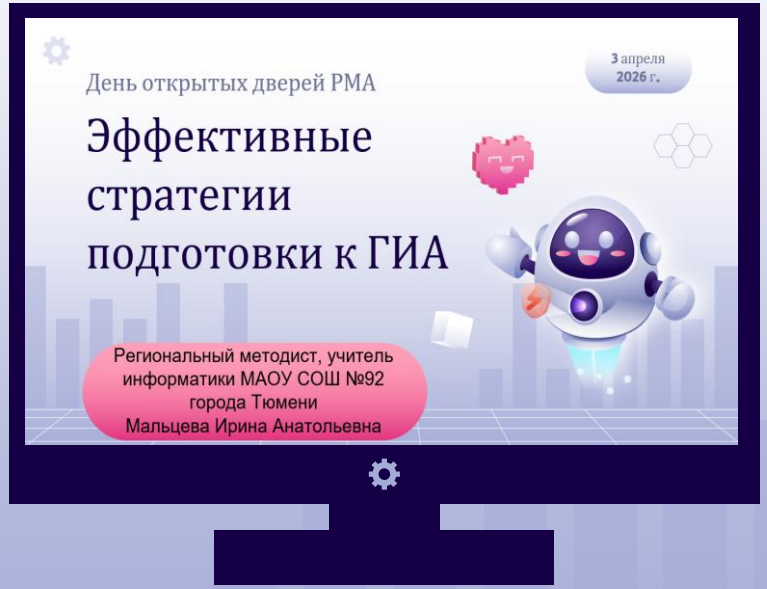
В ответе запишите два числа — количество найденных чисел и максимальное из них.

```
from fnmatch import fnmatch
mask = '4?8?15?16?23'
for n in range(42, 10**12, 123):
    if fnmatch(str(n), mask):
        print(n, n // 123)
```

Оптимизация перебора

```
# Массив для подходящих чисел
m = []
# Возможные цифры на месте "?"
digits = '0123456789'
for d1 in digits:
    for d2 in digits:
        for d3 in digits:
            for d4 in digits:
                # Формируем f-строку
                n = f'4{d1}8{d2}15{d3}16{d4}23'
                # Переводим в целое число
                n = int(n)
                # Проверяем остаток от деления
                if n%123 == 42:
                    # Добавляем в массив
                    m.append(n)
# Выводим длину массива и максимальный элемент в нём
print(len(m), max(m))
#85 498915816123
```

Перебор звёздочек



Задание 25

Средняя | Маркер

Яндекс Учебник

Назовём маской числа последовательность цифр, в которой также могут встречаться следующие символы:

- символ «?» означает ровно одну произвольную цифру;
- символ «*» означает любую последовательность цифр произвольной длины, в том числе «*» может задавать и пустую последовательность.

Например, маске 123*4?5 соответствуют числа 123405 и 12300405.

Среди натуральных чисел, не превышающих 10^{10}, найдите все числа, соответствующие маске 48*15*0, делящиеся на 42 без остатка.

В ответе запишите в первом столбце таблицы пять наибольших найденных чисел в порядке убывания, а во втором столбце — соответствующие им результаты деления этих чисел на 42.

Алгоритм перебора последовательностей цифр на месте звёздочки:

```
from itertools import product
# Цикл для возможной длины последовательности на месте звёздочки
for len_star_1 in range(6):
    # Создаём все возможные последовательности цифр длины len_star_1
    for x in product('0123456789', repeat=len_star_1):
        # Переводим кортеж в строку
        a = ''.join(x)
```

```python
from itertools import product
m = []
for len_star_1 in range(6):
    for len_star_2 in range(6):
        if (len_star_1 + len_star_2) <= 5:
            for x in product('0123456789', repeat=len_star_1):
                for y in product('0123456789', repeat=len_star_2):
                    # Формируем строку
                    n = '48'+''.join(x) + '15' + ''.join(y) + '0'
                    # Переводим в целое число
                    n = int(n)
                    # Если n делится на 42, добавляем его в массив, а также добавляем результат деления на 42
                    if n%42 == 0:
                        m.append([n, n//42])
# Сортируем массив по убыванию
m = sorted(m, reverse=True)
# Выводим первые 5 наибольших чисел и их результаты деления
print(m[:5])
```

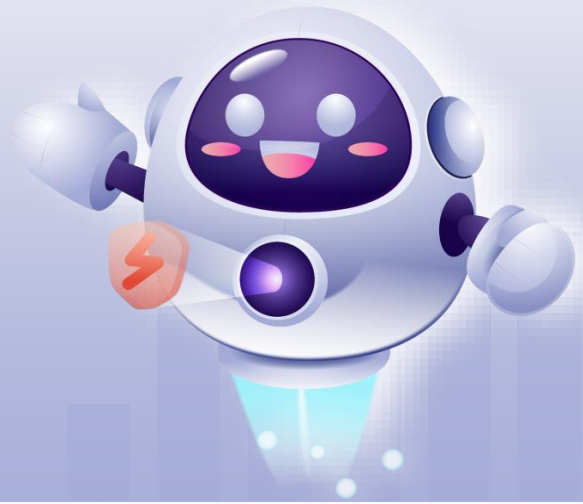
Итоги:

1. Используйте fnmatch для проверки по шаблону — это удобно, если маска в задании дана в том же формате.
2. Оптимизируйте перебор:
- если нужно проверять делимость, перебирайте числа с шагом, равным делителю (или с учётом остатка);
- если в маске есть «?», перебирайте все комбинации цифр на этих позициях — это значительно быстрее полного перебора;
- если в маске есть «*», оцените максимальную длину и перебирайте все возможные строки для «*» с помощью itertools.product.

3.Сначала проверяйте арифметические условия, потом маску. Это может ускорить работу, если условие на делимость отсекает много вариантов.

Форма обратной связи

https://forms.gle/EATGd1RhFe9LGk9z7



Спасибо за внимание!