**Вопросы ДМТИ**

**5-8**

**№5**

**Обобщенный алгоритм Евклида** используется для нахождения наибольшего общего делителя (НОД) двух целых чисел, а также для нахождения коэффициентов Безу, которые выражают НОД в виде линейной комбинации этих чисел.

Пусть даны два целых числа a*a* и b*b*. Алгоритм находит такие целые числа x*x* и y*y*, что:

gcd(a,b)=a⋅x+b⋅ygcd(*a*,*b*)=*a*⋅*x*+*b*⋅*y*

**Шаги алгоритма:**

1. **Инициализация**:
   * Задаем начальные значения:

r0=a, r1=b

x0=1, x1=0

y0=0, y1=1

1. **Основной цикл**:
   * Пока r1≠0, выполняем следующие шаги:
     + Вычисляем частное *q* от деления r0​ на *r*1​:

q=⌊r0/r1⌋

* + - Обновляем значения:

r2=r0−q⋅r1

​x2=x0−q⋅x1

*y*2​=*y*0​−*q*⋅*y*1​

* + - Сдвигаем значения:

r0=r1, r1=r2

​x0=x1, x1=x2

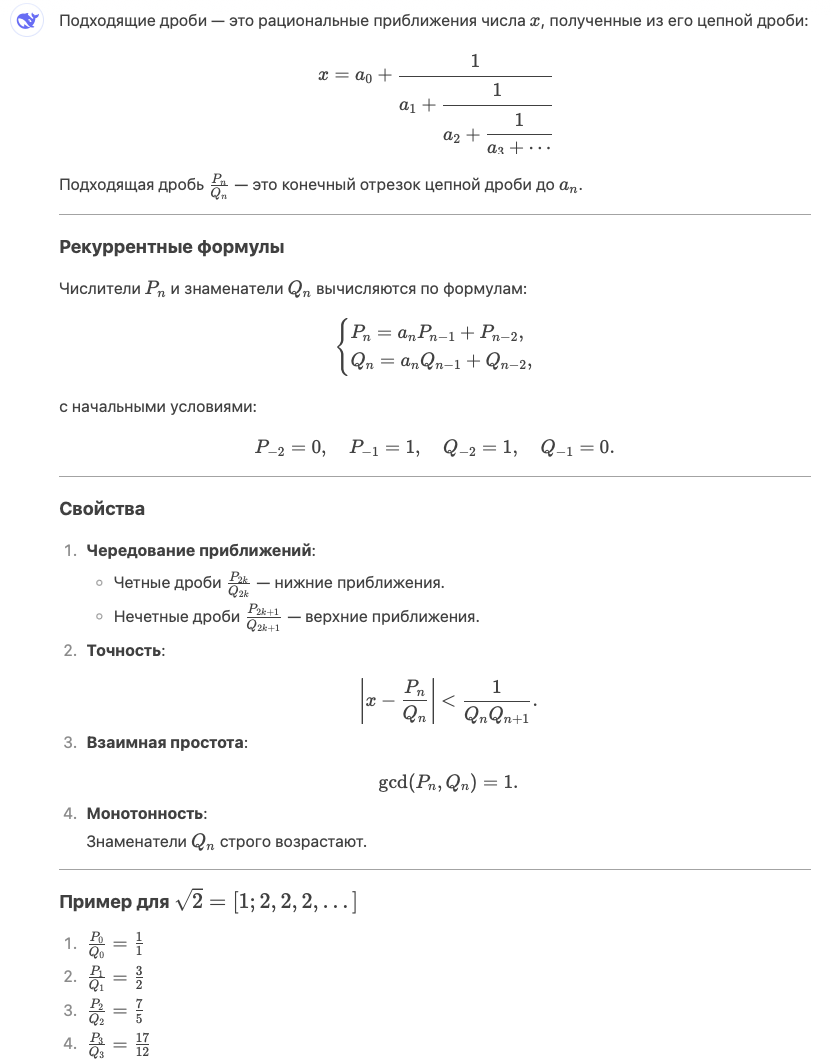
*y*0​=*y*1​ *y*1​=*y*2​

1. **Завершение**:
   * Когда *r*1​=0, алгоритм завершается. В этом случае:

gcd(*a*,*b*)=*r*0

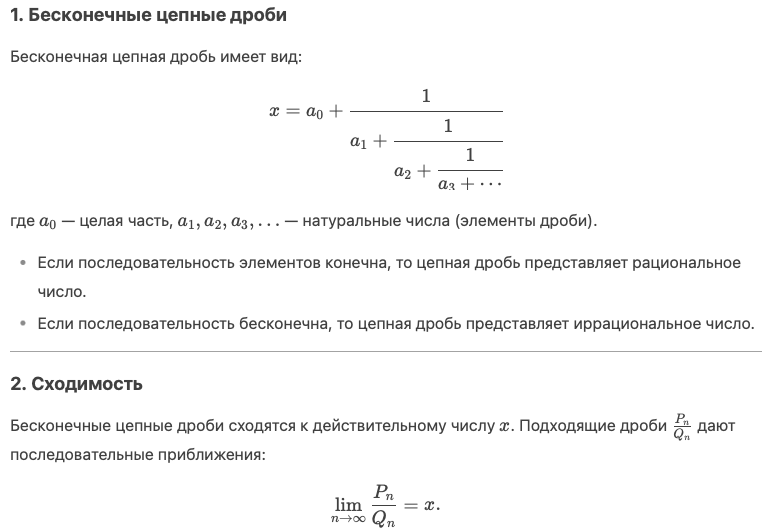
​*x*=*x*0​, *y*=*y*0​

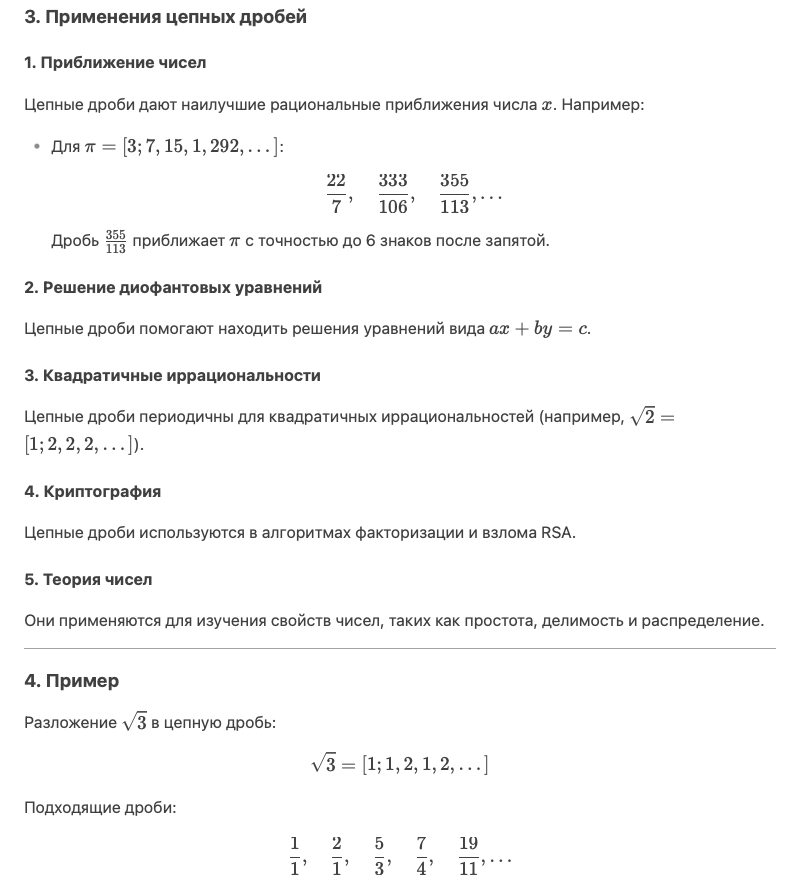
**№6**



**№7**

Бесконечные цепные дроби — это представление действительных чисел в виде бесконечной последовательности элементов. Они используются для приближения чисел, решения уравнений и в других областях математики.

****

****

Цепные дроби — мощный инструмент для приближения чисел, решения уравнений и изучения свойств чисел.

**№8**

Числа Фибоначчи — это последовательность чисел, где каждое число равно сумме двух предыдущих:

****

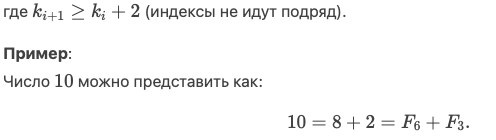
Первые числа Фибоначчи:

0,1,1,2,3,5,8,13,21,34,55,…0,1,1,2,3,5,8,13,21,34,55,…

**2. Теорема Цекендорфа**

**Формулировка**:  
Любое натуральное число N*N* можно единственным образом представить в виде суммы непоследовательных чисел Фибоначчи:

****

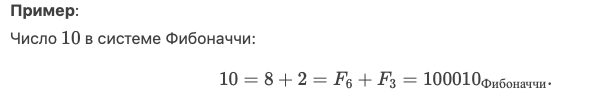
****

**3. Система счисления Фибоначчи**

Система счисления Фибоначчи — это способ представления чисел с использованием чисел Фибоначчи в качестве базиса. Каждое число записывается в виде:

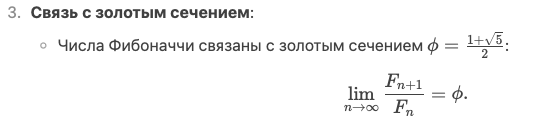
****

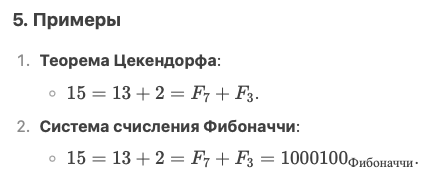
****

****

**4. Свойства**

1. **Единственность представления**:
   * В системе Фибоначчи каждое число имеет единственное представление (следует из теоремы Цекендорфа).
2. **Эффективность**:
   * Система Фибоначчи используется в алгоритмах сжатия данных и теории кодирования.

****

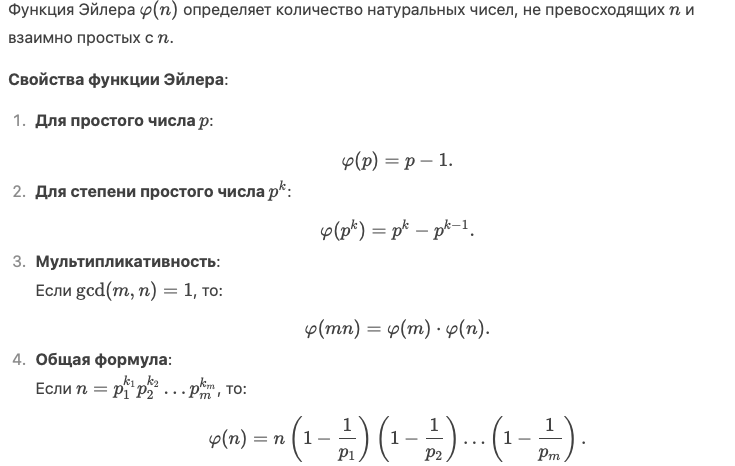
****

Числа Фибоначчи, теорема Цекендорфа и система счисления Фибоначчи находят применение в математике, информатике и теории чисел.

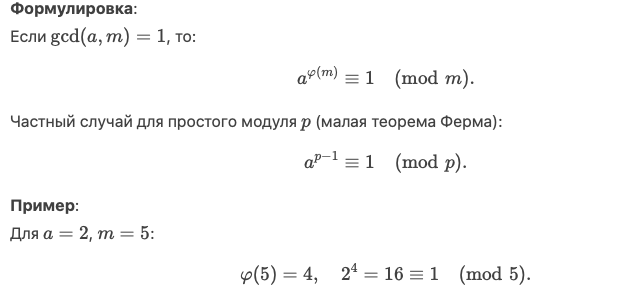
**13-16**

**№13**

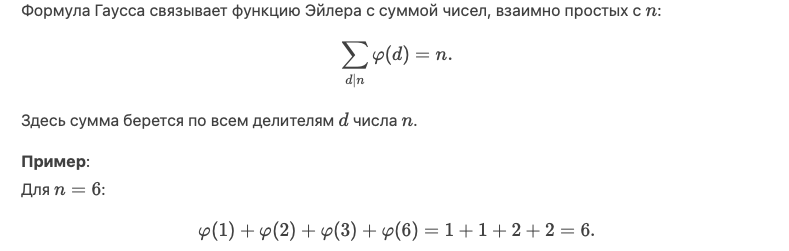
**1. Функция Эйлера**

****

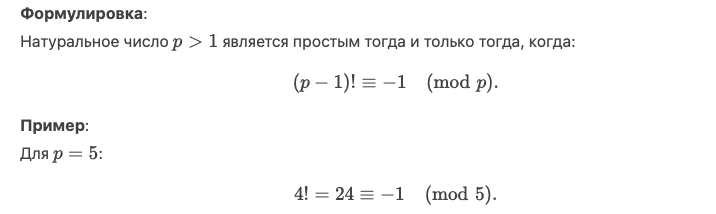
**2. Теорема Эйлера-Ферма**

****

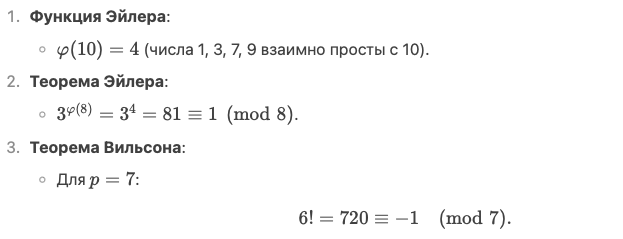
**3. Формула Гаусса**

****

**4. Теорема Вильсона**

****

**5. Примеры**

****

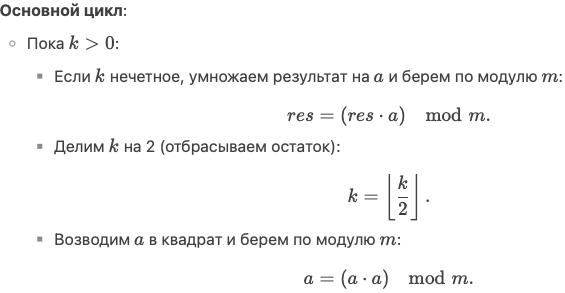
Функция Эйлера, теорема Эйлера-Ферма, формула Гаусса и теорема Вильсона являются важными инструментами в теории чисел и находят применение в криптографии и алгебре.

**№14**

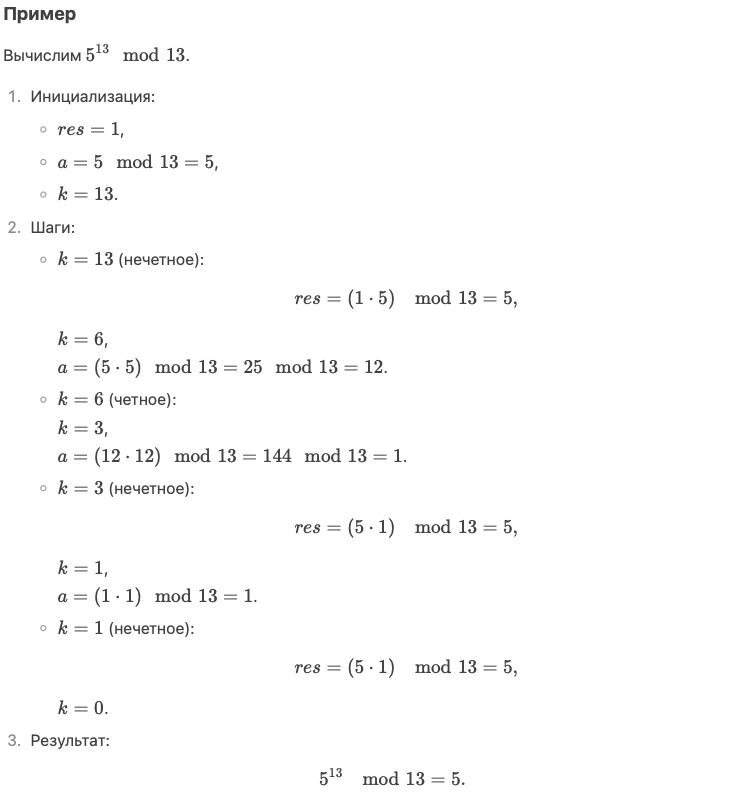
Быстрое возведение числа в степень в кольце Z*m*​ — это алгоритм, позволяющий эффективно вычислять *ak* mod *m* даже для больших степеней *k*. Основная идея заключается в использовании свойств модульной арифметики и двоичного разложения степени *k*.

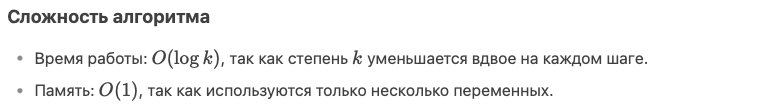
**Алгоритм быстрого возведения в степень**

1. **Входные данные**:
   * Число *a* (основание).
   * Степень *k* (натуральное число).
   * Модуль *m* (натуральное число).
2. **Инициализация**:
   * Результат *res*=1.
   * Основание *a*=*a* mod *m* (сокращаем основание по модулю *m*).

****

1. **Результат**:
   * Возвращаем *res*.

****

****

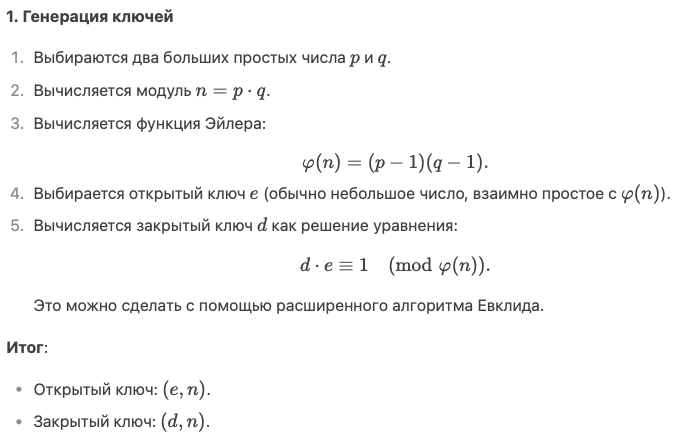
**Применение**

Алгоритм используется в криптографии (например, в RSA), где требуется быстрое вычисление больших степеней по модулю.

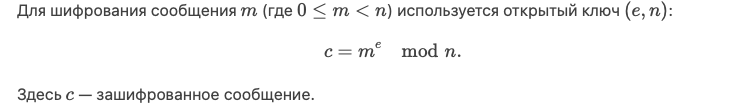
**№15**

Система шифрования RSA (Rivest-Shamir-Adleman) — это один из первых и наиболее широко используемых алгоритмов асимметричного шифрования. Он основан на сложности факторизации больших чисел и применяется для шифрования, цифровой подписи и обмена ключами.

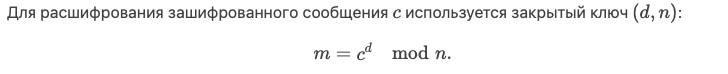
**1. Основные этапы RSA**

****

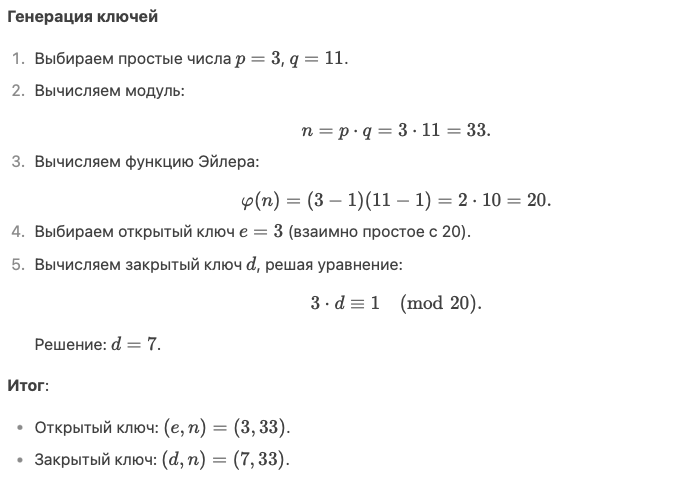
**2. Шифрование**

****

**3. Расшифрование**

****

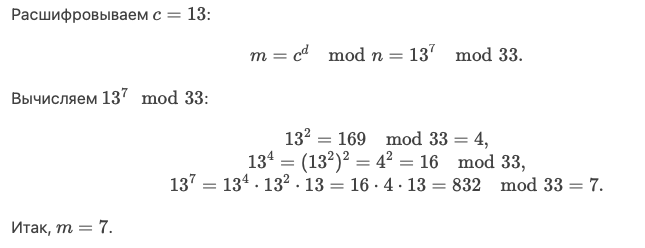
**2. Пример работы RSA**

****

**Шифрование**

****

**Расшифрование**

****

**3. Безопасность RSA**

* Основана на сложности факторизации больших чисел *n*=*p*⋅*q*.
* Для взлома RSA необходимо разложить *n* на множители, что является вычислительно сложной задачей для больших *n*.
* Рекомендуемая длина ключа: 2048 бит и более.

**4. Применение RSA**

1. **Шифрование**:
   * Передача зашифрованных сообщений.
2. **Цифровая подпись**:
   * Подтверждение авторства и целостности данных.
3. **Обмен ключами**:
   * Используется в гибридных системах (например, TLS/SSL).

**5. Ограничения RSA**

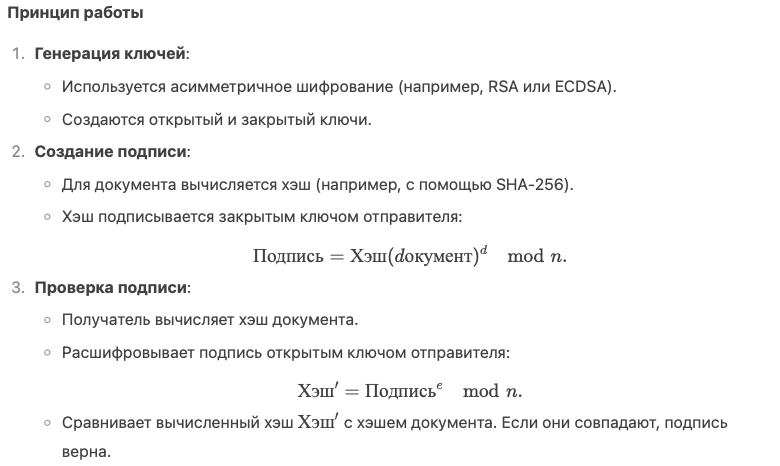
1. Вычислительная сложность:
   * Шифрование и расшифрование требуют больших вычислений.
2. Ограничение на размер сообщения:
   * Сообщение *m* должно быть меньше *n*.
3. Уязвимость к атакам:
   * При неправильной реализации возможны атаки (например, атака на малый модуль *n*).

RSA остается одним из самых популярных алгоритмов благодаря своей надежности и широкому спектру применений.

**№16**

**1. Электронная подпись**

Электронная подпись (ЭП) — это аналог собственноручной подписи, используемый для подтверждения авторства и целостности электронных документов. Она основана на криптографических методах.

****

**Преимущества**

* **Аутентификация**: подтверждение авторства.
* **Целостность**: гарантия неизменности документа.
* **Неотрекаемость**: отправитель не может отрицать подписание.

**Пример**

1. Отправитель создает документ и подписывает его своим закрытым ключом.
2. Получатель проверяет подпись с помощью открытого ключа отправителя.

**2. Электронные деньги**

Электронные деньги — это цифровой аналог наличных, используемый для онлайн-платежей. Они могут быть централизованными (например, PayPal) или децентрализованными (например, Bitcoin).

**Основные принципы**

1. **Централизованные системы**:
   * Управляются центральным органом (банком или платежной системой).
   * Примеры: PayPal, Яндекс.Деньги, WebMoney.
2. **Децентрализованные системы**:
   * Основаны на блокчейне и криптографии.
   * Примеры: Bitcoin, Ethereum.

**Технологии**

1. **Блокчейн**:
   * Распределенная база данных, хранящая информацию о транзакциях.
   * Каждый блок содержит хэш предыдущего блока, что обеспечивает неизменность данных.
2. **Криптография**:
   * Используется для защиты транзакций и создания цифровых подписей.

**Пример работы Bitcoin**

1. **Транзакция**:
   * Пользователь А отправляет Bitcoin пользователю Б.
   * Транзакция подписывается закрытым ключом пользователя А.
2. **Подтверждение**:
   * Транзакция включается в блок и проверяется майнерами.
   * После подтверждения блок добавляется в блокчейн.
3. **Безопасность**:
   * Основана на сложности майнинга (решения криптографических задач).

**Преимущества электронных денег**

1. **Удобство**:
   * Быстрые и простые платежи.
2. **Глобальность**:
   * Возможность международных переводов.
3. **Анонимность** (в децентрализованных системах):
   * Пользователи могут оставаться анонимными.

**Недостатки**

1. **Централизованные системы**:
   * Зависимость от центрального органа.
   * Риск блокировки счетов.
2. **Децентрализованные системы**:
   * Высокая волатильность курса.
   * Медленные транзакции (в некоторых системах).

**3. Связь электронной подписи и электронных денег**

* Электронная подпись используется для подтверждения транзакций в системах электронных денег.
* В децентрализованных системах (например, Bitcoin) каждая транзакция подписывается закрытым ключом отправителя.

Электронная подпись и электронные деньги играют ключевую роль в цифровой экономике, обеспечивая безопасность, удобство и глобальность финансовых операций.

**21-24**

**№21**

Алгоритм Хаффмана — это метод сжатия данных без потерь, основанный на использовании переменной длины кодов для символов. Часто встречающиеся символы кодируются более короткими кодами, а редко встречающиеся — более длинными. Это позволяет уменьшить общий размер данных.

**1. Основные этапы алгоритма Хаффмана**

**1. Построение частотного словаря**

1. Для каждого символа в исходных данных вычисляется частота его встречаемости.
2. Символы сортируются по возрастанию частоты.

**2. Построение дерева Хаффмана**

1. Создается список узлов, каждый из которых содержит символ и его частоту.
2. Пока в списке больше одного узла:
   * Выбираются два узла с наименьшими частотами.
   * Создается новый узел, частота которого равна сумме частот выбранных узлов.
   * Новый узел становится родителем для выбранных узлов.
   * Новый узел добавляется в список, а выбранные узлы удаляются.
3. Корень дерева — это узел, содержащий сумму всех частот.

**3. Построение кодов**

1. Начиная с корня дерева, каждому левому ребру присваивается 0, а правому — 1.
2. Для каждого символа код определяется как последовательность битов от корня до листа, соответствующего символу.

**4. Кодирование данных**

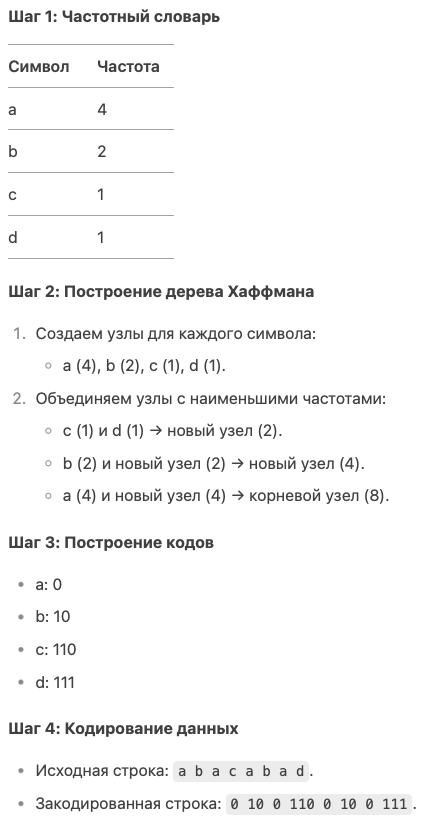
1. Каждый символ в исходных данных заменяется его кодом Хаффмана.
2. Коды записываются в выходной файл.

**2. Пример работы алгоритма**

**Исходные данные**

Пусть строка для кодирования: "abacabad".

**Шаг 1: Частотный словарь**

****

**3. Декодирование**

Для декодирования используется то же дерево Хаффмана:

1. Начиная с корня, двигаемся по дереву в зависимости от битов закодированной строки.
2. Когда достигаем листа, получаем исходный символ.

**5. Недостатки алгоритма Хаффмана**

1. **Зависимость от частот**:
   * Требуется предварительный анализ данных для построения частотного словаря.
2. **Ограниченная эффективность**:
   * Для данных с равномерным распределением частот сжатие может быть незначительным.

**6. Применение**

* Сжатие текстовых данных (например, в архиваторах).
* Сжатие изображений (в формате JPEG).
* Передача данных (например, в протоколах связи).

Алгоритм Хаффмана остается одним из самых популярных методов сжатия данных благодаря своей простоте и эффективности.

**№22**

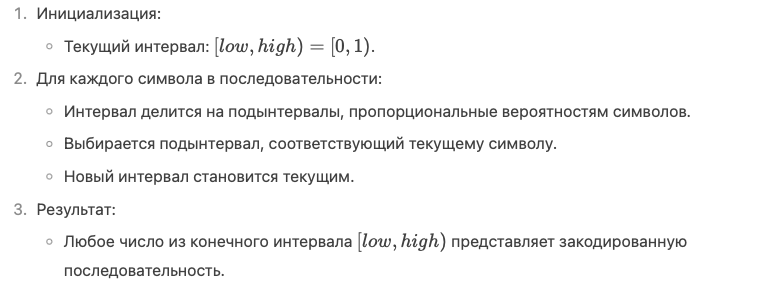
Арифметическое кодирование — это метод сжатия данных без потерь, который позволяет кодировать последовательности символов в одно дробное число. В отличие от алгоритма Хаффмана, арифметическое кодирование может достигать теоретического предела сжатия (энтропии Шеннона) для данных с известным распределением вероятностей.

**1. Основные этапы арифметического кодирования**

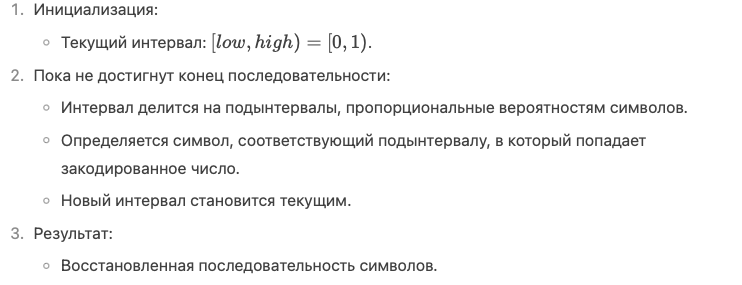
**1. Построение модели данных**

1. Для каждого символа в алфавите задается вероятность его появления.
2. Символы упорядочиваются, и для каждого определяется интервал на отрезке [0,1)[0,1).

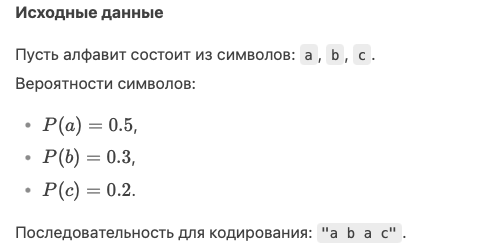
**2. Кодирование**

****

**3. Декодирование**

****

**2. Пример работы арифметического кодирования**

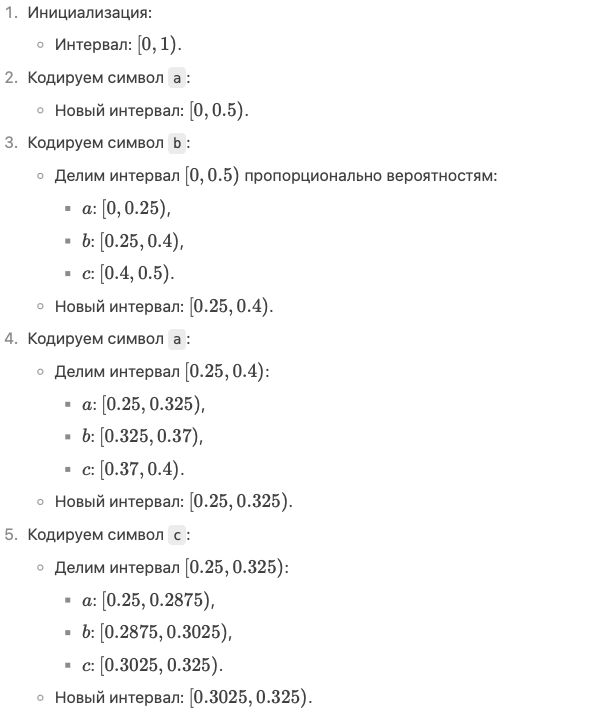
****

**Шаг 1: Построение модели данных**

Интервалы для символов:

* *a*: [0,0.5),
* *b*: [0.5,0.8),
* *c*: [0.8,1).

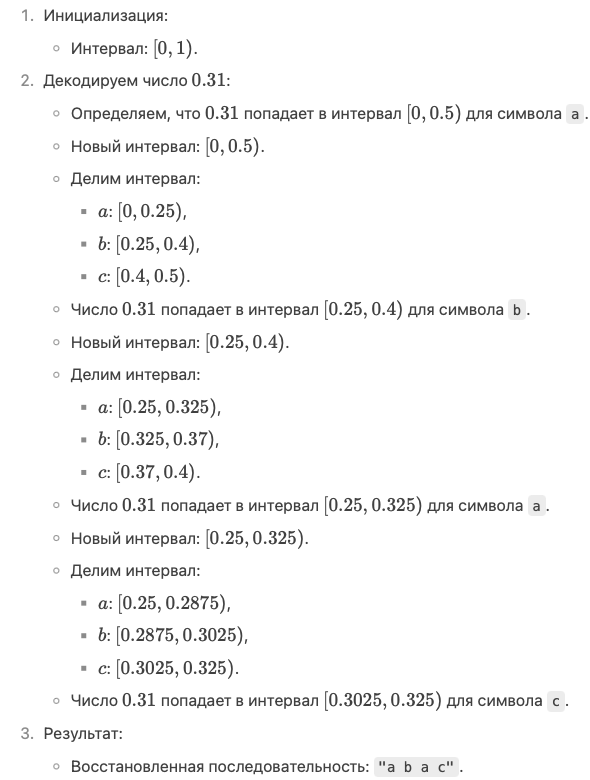
**Шаг 2: Кодирование**



**Шаг 3: Результат кодирования**

Любое число из интервала [0.3025, 0.325) может быть использовано для представления последовательности "a b a c". Например, число 0.31.

**Шаг 4: Декодирование**



**3. Преимущества арифметического кодирования**

1. **Высокая степень сжатия**:
   * Может достигать теоретического предела сжатия (энтропии Шеннона).
2. **Гибкость**:
   * Подходит для данных с любым распределением вероятностей.
3. **Адаптивность**:
   * Может использовать адаптивные модели, где вероятности символов обновляются по мере обработки данных.

**4. Недостатки арифметического кодирования**

1. **Вычислительная сложность**:
   * Требует больше ресурсов, чем алгоритм Хаффмана.
2. **Чувствительность к ошибкам**:
   * Ошибка в одном бите может привести к некорректному декодированию всей последовательности.

**5. Применение**

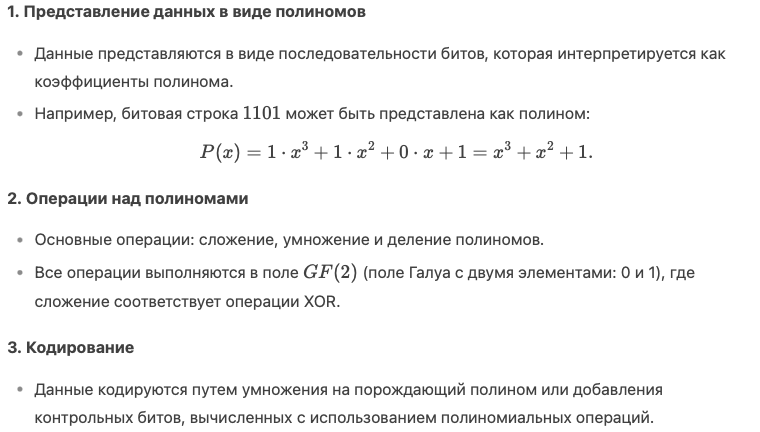
* Сжатие текстовых данных.
* Сжатие изображений (например, в формате JPEG 2000).
* Сжатие аудио и видео.

Арифметическое кодирование — это мощный метод сжатия данных, который широко используется в современных алгоритмах сжатия.

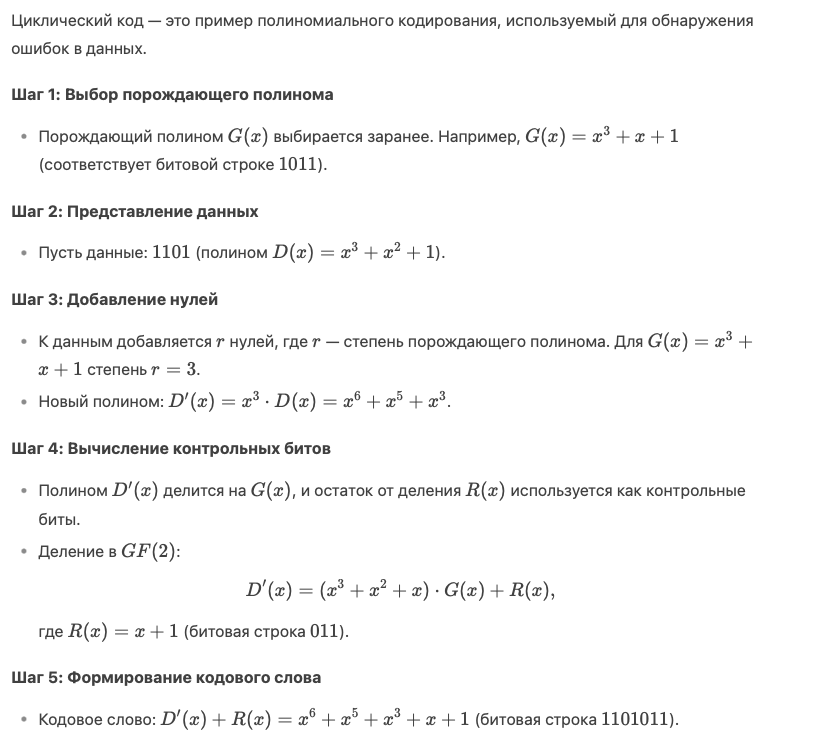
**№23**

Полиномиальное кодирование — это метод кодирования данных, основанный на представлении информации в виде полиномов (многочленов). Этот подход широко используется в теории кодирования, криптографии и обработке сигналов. Одним из самых известных примеров полиномиального кодирования является **циклический код** (например, CRC — Cyclic Redundancy Check).

**1. Основные идеи полиномиального кодирования**



**2. Пример: Циклический код (CRC)**



**3. Декодирование**

* Принятое кодовое слово делится на порождающий полином *G*(*x*).
* Если остаток от деления равен нулю, ошибок нет. В противном случае обнаружена ошибка.

**4. Преимущества полиномиального кодирования**

1. **Обнаружение ошибок**:
   * Позволяет обнаруживать ошибки в данных (например, CRC используется в сетевых протоколах).
2. **Эффективность**:
   * Простые вычисления, основанные на операциях XOR.
3. **Гибкость**:
   * Подходит для различных типов данных и приложений.

**5. Недостатки**

1. **Ограниченная коррекция ошибок**:
   * CRC и другие полиномиальные коды обычно используются только для обнаружения ошибок, а не для их исправления.
2. **Зависимость от порождающего полинома**:
   * Качество кодирования зависит от выбора *G*(*x*).

**6. Применение**

* **Сетевые протоколы** (Ethernet, Wi-Fi, USB) для обнаружения ошибок.
* **Хранение данных** (например, в файловых системах).
* **Криптография** (например, в алгоритмах хэширования).

Полиномиальное кодирование — это мощный инструмент для работы с данными, который находит применение в различных областях благодаря своей простоте и эффективности.

**№24**

**1. Лексикографический и антилексикографический порядок**

**Лексикографический порядок**

Лексикографический порядок — это способ упорядочивания наборов элементов, аналогичный порядку слов в словаре. Для двух наборов *A* и *B*:

1. Сравниваются первые элементы. Если они разные, набор с меньшим элементом считается меньшим.
2. Если первые элементы равны, сравниваются вторые элементы, и так далее.
3. Если один набор является префиксом другого, меньшим считается более короткий набор.

**Пример**:

* Наборы чисел: (1,2,3), (1,2,4), (1,3,1).
* Лексикографический порядок: (1,2,3), (1,2,4), (1,3,1).

**Антилексикографический порядок**

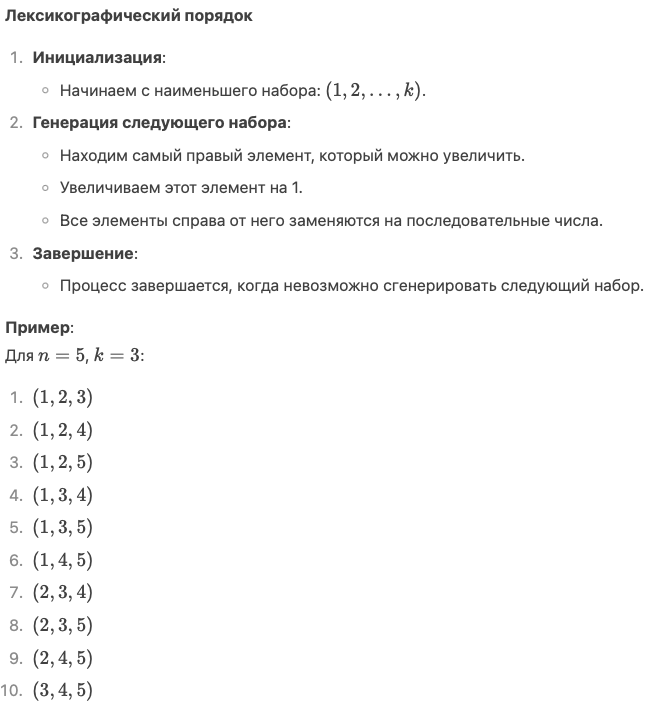
Антилексикографический порядок — это обратный лексикографическому порядок. Сравнение начинается с последних элементов:

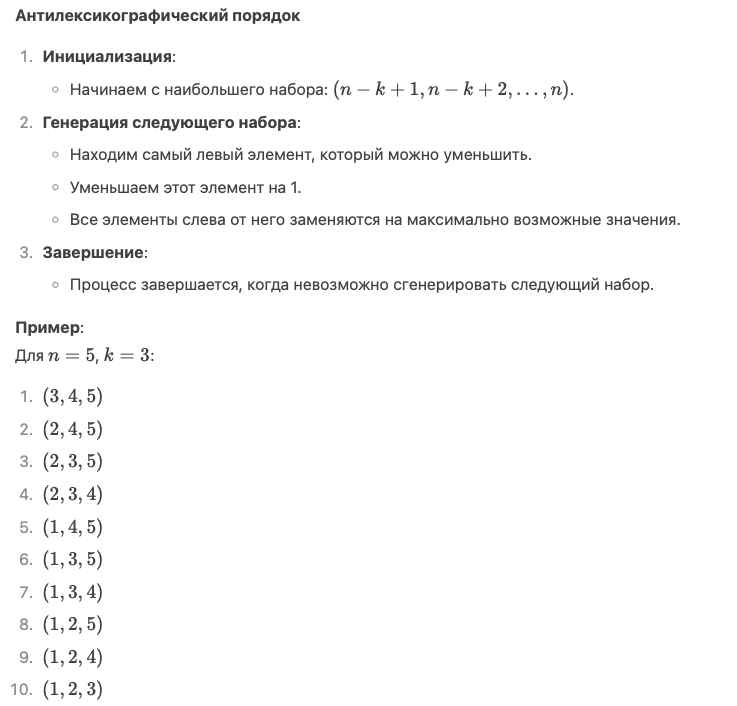
1. Сравниваются последние элементы. Если они разные, набор с меньшим элементом считается меньшим.
2. Если последние элементы равны, сравниваются предпоследние элементы, и так далее.
3. Если один набор является суффиксом другого, меньшим считается более короткий набор.

**Пример**:

* Наборы чисел: (1,2,3), (1,2,4), (1,3,1).
* Антилексикографический порядок: (1,3,1), (1,2,3), (1,2,4).

1. **Генерация***k***-элементных подмножеств**





**3. Применение**

* **Комбинаторика**: генерация подмножеств, сочетаний, перестановок.
* **Алгоритмы**: задачи на перебор, оптимизация, поиск.
* **Криптография**: генерация ключей, тестирование алгоритмов.

**4. Пример кода (лексикографический порядок)**

def next\_lex\_subset(subset, n):

k = len(subset)

for i in range(k - 1, -1, -1):

if subset[i] < n - k + i + 1:

subset[i] += 1

for j in range(i + 1, k):

subset[j] = subset[j - 1] + 1

return subset

return None

# Пример использования

n = 5

k = 3

subset = list(range(1, k + 1))

while subset is not None:

print(subset)

subset = next\_lex\_subset(subset, n)

**5. Пример кода (антилексикографический порядок)**

def next\_antilex\_subset(subset, n):

k = len(subset)

for i in range(k):

if subset[i] > i + 1:

subset[i] -= 1

for j in range(i - 1, -1, -1):

subset[j] = n - k + j + 1

return subset

return None

# Пример использования

n = 5

k = 3

subset = list(range(n - k + 1, n + 1))

while subset is not None:

print(subset)

subset = next\_antilex\_subset(subset, n)

Лексикографический и антилексикографический порядки позволяют эффективно генерировать k*k*-элементные подмножества, что полезно в комбинаторных задачах и алгоритмах.