Оглавление

[**Задание 1.** 4](#_Toc168422446)

[*Реализуйте структуру данных "Односвязный список" на языке программирования по вашему выбору. Необходимо реализовать создание пустого списка, добавление элемента списка, удаление элемента, поиск элемента по номеру и по значению. Желательно использовать ООП. По возможности обеспечить все возможные проверки.* 4](#_Toc168422447)

[*«Односвязный список»:* 4](#_Toc168422448)

[**Задание 2.** 7](#_Toc168422449)

[*Спроектировать схему базы данных по текстовому описанию. Обосновать нормальность отношений. Задача «РЕСТОРАН». Постоянным клиентам предоставляется возможность заказать столик заранее. Официант указывает столик, открывает гостевой счет и вводит заказы в соответствии с меню. Далее заказ автоматически обрабатывается, формируются марки на приготовление выбранных блюд и направляют их на производство, в соответствующие цеха кухни, в бар. Расчеты с посетителем сводятся к простой операции: на бланке печатается итоговый счет. Если клиент – постоянный посетитель, то соответствующие привилегии рассчитываются автоматически, затем указываются способ оплаты и полученная от клиента сумма.* 7](#_Toc168422450)

[**Задание 3.** 10](#_Toc168422451)

[*При помощи библиотеки pandas создать два датафрэйма с индексами 'Moscow', 'Tula', 'Yaroslavl', 'Tver' и 'Moscow', 'Tula', 'Volgograd', 'Novgorod' и случайными значениями в столбцах 'report' (от 1 до 10) и 'sales' (от 100 до 1000). Написать программный код расчёта суммы продаж и суммарное количество отчетов по двум таблицам.* 10](#_Toc168422452)

[**Задание 4.** 11](#_Toc168422453)

[*Смоделируйте процесс «Оказание услуги» в нотации EPC. Для получения услуги клиент должен заполнить заявку на сайте поставщика услуги, выбрав вид услуги, желаемое время оказания и указав контактные данные (ФИО, телефон). Работник регистратуры, работая в ИС IBM BPM, проверяет заполненную заявку на корректность и в случае правильного ее заполнения формирует заказ, указывая дату и время оказания услуги. Каждый рабочий день компании – поставщика разделен на временные интервалы, которые заполняются заказами на получение услуг. Если все 15 временные интервалы желаемой даты заняты, работник регистратуры ищет свободные интервалы на ближайшие даты. Помимо этого, если клиент указал в заявке корректный номер телефона, то при формировании заказа работник регистратуры указывает в системе необходимость уведомления клиента по СМС (отправка уведомлений осуществляется через СМС - шлюз).* 11](#_Toc168422454)

[**Задание 5.** 11](#_Toc168422455)

[*По сетевым адресам и маскам определить, находятся эти два устройства в одной подсети или в разных.* 11](#_Toc168422456)

[**Задание 6.** 12](#_Toc168422457)

[*Дана матрица результатов классификации в задачах диагностики болезни, где классы 1 и 2 представляют различные варианты виды диагностируемой болезни, а класс 0 — отсутствие болезни.* 12](#_Toc168422458)

[**Задание 7.** 12](#_Toc168422459)

[Используя как можно меньше операций в исходном массиве "шахматная доска" np.zeros((8,8)), добиться расстановки кодов "шахматных фигур". 12](#_Toc168422460)

[**Задание 8.** 13](#_Toc168422461)

[Создайте список, содержащий по 3 объекта каждого класса (базовый класс People (атрибуты имя и возраст), производный класс Worker, имеющий атрибуты: должность (post) и зарплата (salary), и производный класс Student, имеющий атрибуты оценки (marks) и текущие изучаемые дисциплины (subjects). Для списка объектов: 13](#_Toc168422462)

[**Задание 9.** 13](#_Toc168422463)

[Гидрометцентр оповестил о снегопаде в г. Москве. Для того, чтобы могли расчистить дороги, следует выяснить, сколько снегоуборочных машин им потребуется. Известно, что на каждый сантиметр слоя выпавшего снега требуется 50 снегоуборочных машин и 20 машин для вывоза снега.  Помимо этого, следует учитывать загруженность дорог в Москве. На каждый балл загруженности дорог (от 0 до 10 включительно) следует выпускать на 5% больше снегоуборочных машин и машин для вывоза снега. Рассчитать сколько машин потребуется при прогнозируемом количестве осадков и ситуации на дорогах в г. Москве. Решите данную задачу в парадигме ООП (объектно-ориентированного программирования). Создать класс c полями: день недели, прогнозируемое количество осадков, пробки, а также метод для расчета необходимого количества машин для уборки в текущий день и метод вывода результатов на экран. 13](#_Toc168422464)

[**Задание 10.** 14](#_Toc168422465)

[Решите данную задачу в парадигме ООП (объектно-ориентированного программирования). Программа должна вычислять цену билетов в маршрутке для пассажиров разных категорий. Если возраст пассажира до 3-х лет, то бесплатный проезд. Если возраст пассажира 7-18 лет (школьник), то ему предоставляется скидка 50%. Также пенсионерам предполагается скидка 50%, а льготники имеют бесплатный проезд. Цена билета также зависит от расстояния, которое проезжает пассажир: до 20 км – стоимость 100руб; от 21 до 100 км – стоимость 150руб; от 101 км до 200 км – стоимость 200 рублей; >=201 км – стоимость 400руб. 14](#_Toc168422466)

[**Задание 11.** 14](#_Toc168422467)

[Даны 2 кольцевых списка с фамилиями шахматистов 2-х команд. Произвести жеребьевку. В первой команде выбирается каждый n-й игрок, а во второй - каждый k-й. Для решения задачи использовать ООП. 14](#_Toc168422468)

[**Задание 12.** 14](#_Toc168422469)

[Даны 2 списка: перечень товаров и фамилии покупателей. Каждый n-й покупатель покупает m-й товар. Вывести список покупок. Для решения задачи использовать ООП. 14](#_Toc168422470)

[**Задание 13.** 14](#_Toc168422471)

[Реализовать класс Стек с методами push, pop, top (определение вершины стека) и is\_empty (проверка пуст стек или нет) с помощью ООП. Удалить все элементы стека, равные первому. 14](#_Toc168422472)

[**Задание 14.** 15](#_Toc168422473)

[Решить задачу поиска наибольшего общего делителя (НОД) в двух парадигмах: структурной и функциональной. Показать на примере программного кода основные особенности данных парадигм. 15](#_Toc168422474)

[**Задание 15.** 15](#_Toc168422475)

[Спроектировать базу данных к 3НФ. База данных должна содержать следующую информацию: ФИО преподавателя, должность, номер группы, название кафедры, номер аудитории, число мест в аудитории, количество студентов в группе, специальность обучения, название предмета, вид занятий (лекция, практика, лабораторная работа), семестр. 15](#_Toc168422476)

[**Задание 16.** 15](#_Toc168422477)

[Сгенерировать двумерный массив *arr* размерности (4, 7), состоящий из случайных действительных чисел, равномерно распределенных в диапазоне от 0 до 20. Суммировать значения каждой строки массива *arr*. Для этого создать новый массив *arr\_sum*, в котором к исходному массиву *arr* добавлен еще один столбец справа, содержащий суммарные значения каждой строки массива *arr*. Суммарные значения округлить (с помощью np.round()) до двух десятичных знаков. Полученный результат отсортируйте по столбцу суммы по убыванию. Использовать numpy. 15](#_Toc168422478)

[**Задание 17.** 16](#_Toc168422479)

[Спроектировать ER-модель базы данных "Books" с обязательными полями (id\_book, title (название книги), author (автор), genre (жанр)). Дополните базу данных необходимыми полями. Напишите SQL-запросы, позволяющие осуществить: 16](#_Toc168422480)

[**Задание 18.** 16](#_Toc168422481)

[Смоделировать процесс получения книги в библиотеке в нотации IDEF0 с указанием предметной области и диаграммы декомпозиций. 16](#_Toc168422482)

[**Задание 19.** 16](#_Toc168422483)

[Смоделировать процесс защиты выпускной квалификационной работы в нотации IDEF0 с указанием предметной области и диаграммы декомпозиций. 16](#_Toc168422484)

[**Задание 20.** 16](#_Toc168422485)

[Смоделировать процесс покупки товара в нотации DFD. 16](#_Toc168422486)

[**Задание 21.** 17](#_Toc168422487)

[Спроектировать схему базы данных по текстовому описанию. Обосновать нормальность отношений. В воображаемой социальной сети есть Пользователи (id, имя), Фото (id, название, автор) и Комментарии К Фото (id, текст, автор, к какому Фото относится). Необходимо добавить возможность для Пользователей ставить лайки другим Пользователям, Фото или Комментариям К Фото. Нужно реализовать такие возможности: пользователь не может поставить 2 лайка одной и той же сущности (например одному и тому же Фото); пользователь может отозвать лайк; необходимо иметь возможность посчитать число полученных сущностью лайков и вывести список Пользователей, поставивших лайки; в будущем могут появиться новые виды сущностей которые можно лайкать 17](#_Toc168422488)

[**Задание 22.** 17](#_Toc168422489)

[Выполнить: 17](#_Toc168422490)

[перевод десятичного представления IP адреса в двоичный - 173.143.32.194 (5 баллов); 17](#_Toc168422491)

[**Задание 23.** 17](#_Toc168422492)

[С помощью модуля pandas создать csv-файл students.csv, содержащий информацию о студентах и их оценках по дисциплинам. 17](#_Toc168422493)

[Определить: 17](#_Toc168422494)

[**Задание 24.** 18](#_Toc168422495)

[Дана сеть: 40.212.114.0/24. Разбейте сеть на подсети меньшего размера (не менее двух вариантов). Определите максимальное количество хостов в каждой такой сети и максимальное количество сетей. Определите широковещательный адрес для каждой подсети. 18](#_Toc168422496)

[**Задание 25.** 18](#_Toc168422497)

[В контексте повышения производительности системы обработки данных возникла задача выбора наиболее подходящего алгоритма сортировки. Требуется провести анализ трех популярных алгоритмов сортировки: пузырьковой, вставками и быстрой сортировки. Оценить их производительность на наборах данных различного объема и степени упорядоченности, чтобы определить наиболее эффективный алгоритм для использования в системе. 18](#_Toc168422498)

# **Задание 1.**

# *Реализуйте структуру данных "Односвязный список" на языке программирования по вашему выбору. Необходимо реализовать создание пустого списка, добавление элемента списка, удаление элемента, поиск элемента по номеру и по значению. Желательно использовать ООП. По возможности обеспечить все возможные проверки.*

## *«Односвязный список»:*

Для реализации односвязного списка на языке Python с использованием объектно-ориентированного программирования, мы создадим два класса: Node и LinkedList. Класс Node будет представлять отдельный элемент списка, а класс LinkedList будет представлять сам список и содержать методы для добавления, удаления и поиска элементов.

Класс Node будет содержать два атрибута: value, хранящий значение узла, и next, указывающий на следующий узел в списке.

Класс LinkedList будет содержать методы для:

* Создания пустого списка
* Добавления элемента
* Удаления элемента
* Поиска элемента по номеру
* Поиска элемента по значению

class Node:

def \_\_init\_\_(self, value=None):

self.value = value

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def is\_empty(self):

return self.head is None

def add(self, value):

new\_node = Node(value)

if self.is\_empty():

self.head = new\_node

else:

current = self.head

while current.next:

current = current.next

current.next = new\_node

def delete(self, value):

if self.is\_empty():

print("List is empty. Nothing to delete.")

return

if self.head.value == value:

self.head = self.head.next

return

current = self.head

while current.next:

if current.next.value == value:

current.next = current.next.next

return

current = current.next

print(f"Value {value} not found in the list.")

def find\_by\_index(self, index):

if self.is\_empty():

print("List is empty.")

return None

current = self.head

count = 0

while current:

if count == index:

return current.value

count += 1

current = current.next

print(f"Index {index} out of range.")

return None

def find\_by\_value(self, value):

if self.is\_empty():

print("List is empty.")

return None

current = self.head

index = 0

while current:

if current.value == value:

return index

index += 1

current = current.next

print(f"Value {value} not found in the list.")

return None

def display(self):

if self.is\_empty():

print("List is empty.")

return

current = self.head

while current:

print(current.value, end=" -> ")

current = current.next

print("None")

# Пример использования:

ll = LinkedList()

ll.add(1)

ll.add(2)

ll.add(3)

ll.display() # Output: 1 -> 2 -> 3 -> None

print(ll.find\_by\_index(1)) # Output: 2

print(ll.find\_by\_value(3)) # Output: 2

ll.delete(2)

ll.display() # Output: 1 -> 3 -> None

ll.delete(4) # Output: Value 4 not found in the list.

# **Задание 2.**

# *Спроектировать схему базы данных по текстовому описанию. Обосновать нормальность отношений. Задача «РЕСТОРАН». Постоянным клиентам предоставляется возможность заказать столик заранее. Официант указывает столик, открывает гостевой счет и вводит заказы в соответствии с меню. Далее заказ автоматически обрабатывается, формируются марки на приготовление выбранных блюд и направляют их на производство, в соответствующие цеха кухни, в бар. Расчеты с посетителем сводятся к простой операции: на бланке печатается итоговый счет. Если клиент – постоянный посетитель, то соответствующие привилегии рассчитываются автоматически, затем указываются способ оплаты и полученная от клиента сумма.*

Для задачи "РЕСТОРАН" необходимо создать несколько таблиц, которые будут хранить информацию о клиентах, бронировании столиков, официантах, столиках, меню, заказах, цехах кухни и барах, счетах и способах оплаты. Мы будем стремиться к тому, чтобы наши отношения в базе данных были в нормальной форме, чтобы минимизировать избыточность и избежать аномалий при обновлении данных. Таблицы базы данных:

-Clients (Клиенты)

* client\_id (PK): Уникальный идентификатор клиента
* name: Имя клиента
* phone\_number: Номер телефона клиента
* email: Электронная почта клиента
* is\_regular: Флаг постоянного клиента (да/нет)

-Reservations (Бронирования)

* reservation\_id (PK): Уникальный идентификатор бронирования
* client\_id (FK): Идентификатор клиента, сделавшего бронирование
* table\_id (FK): Идентификатор забронированного столика
* reservation\_date: Дата и время бронирования

-Waiters (Официанты)

* waiter\_id (PK): Уникальный идентификатор официанта
* name: Имя официанта
* phone\_number: Номер телефона официанта

-Tables (Столики)

* table\_id (PK): Уникальный идентификатор столика
* number: Номер столика
* seats: Количество мест за столиком

-Menu (Меню)

* menu\_item\_id (PK): Уникальный идентификатор позиции меню
* name: Название блюда
* description: Описание блюда
* price: Цена блюда
* kitchen\_section: Секция кухни (например, бар, горячий цех, холодный цех)

-Orders (Заказы)

* order\_id (PK): Уникальный идентификатор заказа
* table\_id (FK): Идентификатор столика
* waiter\_id (FK): Идентификатор официанта
* order\_date: Дата и время заказа

-OrderDetails (Детали заказов)

* order\_detail\_id (PK): Уникальный идентификатор детали заказа
* order\_id (FK): Идентификатор заказа
* menu\_item\_id (FK): Идентификатор позиции меню
* quantity: Количество заказанных порций

-Bills (Счета)

* bill\_id (PK): Уникальный идентификатор счета
* order\_id (FK): Идентификатор заказа
* total\_amount: Общая сумма счета
* discount: Скидка (если применимо)
* payment\_method: Способ оплаты (наличные, карта и т.д.)
* received\_amount: Полученная от клиента сумма

Связи таблиц:

* Reservations (Бронирования) связана с таблицей Clients (Клиенты) по атрибуту client\_id, чтобы хранить информацию о клиентах, сделавших бронирование.
* Reservations (Бронирования) также связана с таблицей Tables (Столики) по атрибуту table\_id, чтобы связать конкретное бронирование с забронированным столиком.
* Orders (Заказы) связана с таблицей Tables (Столики) по атрибуту table\_id, чтобы определить, какой столик сделал заказ.
* Orders (Заказы) также связана с таблицей Waiters (Официанты) по атрибуту waiter\_id, чтобы установить, какой официант обслуживает заказ.
* OrderDetails (Детали заказов) связана с таблицей Orders (Заказы) по атрибуту order\_id, чтобы установить связь между заказом и его деталями.
* OrderDetails (Детали заказов) также связана с таблицей Menu (Меню) по атрибуту menu\_item\_id, чтобы определить, какие блюда входят в заказ.
* Bills (Счета) связана с таблицей Orders (Заказы) по атрибуту order\_id, чтобы привязать счет к соответствующему заказу.

Обоснование нормальности отношений:

Все отношения в нашей схеме будут приведены к третьей нормальной форме (3NF):

Первая нормальная форма (1NF): Каждое значение атрибута в таблице является атомарным. Например, в таблице Clients каждое значение атрибута (имя, номер телефона, электронная почта) является атомарным.

Вторая нормальная форма (2NF): Все неключевые атрибуты зависят от всего первичного ключа, а не от его части. Например, в таблице OrderDetails неключевые атрибуты (количество) зависят от всего первичного ключа (идентификатор детали заказа).

Третья нормальная форма (3NF): Все неключевые атрибуты зависят только от первичного ключа и не зависят транзитивно от других неключевых атрибутов. Например, в таблице Bills неключевые атрибуты (общая сумма, скидка) зависят только от первичного ключа (идентификатор счета).

# **Задание 3.**

# *При помощи библиотеки pandas создать два датафрэйма с индексами 'Moscow', 'Tula', 'Yaroslavl', 'Tver' и 'Moscow', 'Tula', 'Volgograd', 'Novgorod' и случайными значениями в столбцах 'report' (от 1 до 10) и 'sales' (от 100 до 1000). Написать программный код расчёта суммы продаж и суммарное количество отчетов по двум таблицам.*

Для решения этой задачи мы используем библиотеку pandas и numpy для создания случайных значений. Сначала создадим два датафрейма с заданными индексами и случайными значениями, а затем произведем расчет суммы продаж и суммарного количества отчетов по двум таблицам. Вот программный код для этого:

import pandas as pd

import numpy as np

# Создаем первый DataFrame с индексами 'Moscow', 'Tula', 'Yaroslavl', 'Tver'

df1 = pd.DataFrame({

'report': np.random.randint(1, 11, size=4),

'sales': np.random.randint(100, 1001, size=4)

}, index=['Moscow', 'Tula', 'Yaroslavl', 'Tver'])

# Создаем второй DataFrame с индексами 'Moscow', 'Tula', 'Volgograd', 'Novgorod'

df2 = pd.DataFrame({

'report': np.random.randint(1, 11, size=4),

'sales': np.random.randint(100, 1001, size=4)

}, index=['Moscow', 'Tula', 'Volgograd', 'Novgorod'])

# Выводим созданные DataFrame для проверки

print("Первый DataFrame:")

print(df1)

print("\nВторой DataFrame:")

print(df2)

# Объединяем два DataFrame, суммируя значения по совпадающим индексам

df\_combined = df1.add(df2, fill\_value=0)

# Рассчитываем сумму продаж и суммарное количество отчетов

total\_sales = df\_combined['sales'].sum()

total\_reports = df\_combined['report'].sum()

print("\nОбъединенный DataFrame:")

print(df\_combined)

print(f"\nСумма продаж по двум таблицам: {total\_sales}")

print(f"Суммарное количество отчетов по двум таблицам: {total\_reports}")

Этот код выполняет следующие шаги:

* Импортирует необходимые библиотеки pandas и numpy.
* Создает первый DataFrame df1 с заданными индексами и случайными значениями в столбцах 'report' и 'sales'.
* Создает второй DataFrame df2 с заданными индексами и случайными значениями в столбцах 'report' и 'sales'.
* Объединяет два DataFrame, используя метод add, который суммирует значения по совпадающим индексам, а для отсутствующих значений использует 0.
* Рассчитывает суммарные значения продаж и отчетов в объединенном DataFrame.
* Выводит результаты.

# **Задание 4.**

# *Смоделируйте процесс «Оказание услуги» в нотации EPC. Для получения услуги клиент должен заполнить заявку на сайте поставщика услуги, выбрав вид услуги, желаемое время оказания и указав контактные данные (ФИО, телефон). Работник регистратуры, работая в ИС IBM BPM, проверяет заполненную заявку на корректность и в случае правильного ее заполнения формирует заказ, указывая дату и время оказания услуги. Каждый рабочий день компании – поставщика разделен на временные интервалы, которые заполняются заказами на получение услуг. Если все 15 временные интервалы желаемой даты заняты, работник регистратуры ищет свободные интервалы на ближайшие даты. Помимо этого, если клиент указал в заявке корректный номер телефона, то при формировании заказа работник регистратуры указывает в системе необходимость уведомления клиента по СМС (отправка уведомлений осуществляется через СМС - шлюз).*

Изображение выглядит как текст, диаграмма, снимок экрана, дизайн

Автоматически созданное описание

# **Задание 5.**

# *По сетевым адресам и маскам определить, находятся эти два устройства в одной подсети или в разных.*

*DeviceA: 172.16.17.30/20*

*DeviceB: 172.16.28.15/20*

Чтобы определить, находятся ли два устройства в одной подсети, нужно выполнить побитовое И между IP-адресом устройства и его маской подсети для обоих устройств. Побитовое И (логическое И, также называемое битовым И) — это операция, которая сравнивает соответствующие биты двух чисел и возвращает 1, если оба бита равны 1, и 0, если хотя бы один из битов равен 0. Если результат для обоих устройств одинаков, значит, они находятся в одной подсети. В противном случае они находятся в разных подсетях.

Маска подсети /20 означает, что первые 20 бит используются для обозначения сети, а оставшиеся 12 бит для обозначения узлов. Теперь рассчитаем сетевые адреса для обоих устройств.

-DeviceA: 172.16.17.30/20

* IP-адрес в двоичном виде: 10101100.00010000.00010001.00011110
* Маска подсети /20 в двоичном виде: 11111111.11111111.11110000.00000000

Применим побитовое И между IP-адресом и маской подсети:

10101100.00010000.00010001.00011110

AND 11111111.11111111.11110000.00000000

------------------------------------

10101100.00010000.00010000.00000000

И получаем сетевой адрес DeviceA: 172.16.16.0

-DeviceB: 172.16.28.15/20

IP-адрес в двоичном виде: 10101100.00010000.00011100.00001111

Маска подсети /20 в двоичном виде: 11111111.11111111.11110000.00000000

Применим побитовое И между IP-адресом и маской подсети:

10101100.00010000.00011100.00001111

AND 11111111.11111111.11110000.00000000

------------------------------------

10101100.00010000.00010000.00000000

Сетевой адрес DeviceB: 172.16.16.0

Поскольку оба устройства имеют одинаковый сетевой адрес 172.16.16.0, можно сделать вывод, что устройства находятся в одной подсети.

# **Задание 6.**

# *Дана матрица результатов классификации в задачах диагностики болезни, где классы 1 и 2 представляют различные варианты виды диагностируемой болезни, а класс 0 — отсутствие болезни.*

|  |  |  |  |
| --- | --- | --- | --- |
| *Расчетный класс* | *0 (Эмпирический)* | *1 (Эмпирический)* | *2 (Эмпирический)* |
| *0* | *111* | *3* | *5* |
| *1* | *2* | *123* | *17* |
| *2* | *19* | *0* | *20* |

*Необходимо рассчитать метрики полноты и точности, F-метрику по каждому классу в отдельности, средние и средневзвешенные. Обосновать выбор наиболее релевантной метрики учитывая специфику задачи, несбалансированность классов. Предложить основные подходы для усовершенствования классификации.*

Обозначим:

* TP (True Positives): Истинные положительные
* FP (False Positives): Ложные положительные
* FN (False Negatives): Ложные отрицательные

Для класса 0:

* TP = 111
* FP = 2 + 19 = 21
* FN = 3 + 5 = 8

Для класса 1:

* TP = 123
* FP = 3 + 0 = 3
* FN = 2 + 17 = 19

Для класса 2:

* TP = 20
* FP = 5 + 17 = 22
* FN = 19 + 0 = 19

Теперь, используя эти показатели, рассчитаем метрики для каждого класса:

Полнота (Recall): Recall=𝑇𝑃/(𝑇𝑃+𝐹𝑁)

Точность (Precision): Precision=𝑇𝑃/(𝑇𝑃+𝐹𝑃)

F1-метрика: 𝐹1=2⋅(Precision⋅Recall)/(Precision+Recall)

Метрики для класса 0:

* Recall\_0 = 111 / (111 + 8) ≈ 0.933
* Precision\_0 = 111 / (111 + 21) ≈ 0.841
* F1\_0 = 2 \* (0.933 \* 0.841) / (0.933 + 0.841) ≈ 0.885

Метрики для класса 1:

* Recall\_1 = 123 / (123 + 19) ≈ 0.866
* Precision\_1 = 123 / (123 + 3) ≈ 0.976
* F1\_1 = 2 \* (0.866 \* 0.976) / (0.866 + 0.976) ≈ 0.917

Метрики для класса 2:

* Recall\_2 = 20 / (20 + 19) ≈ 0.513
* Precision\_2 = 20 / (20 + 22) ≈ 0.476
* F1\_2 = 2 \* (0.513 \* 0.476) / (0.513 + 0.476) ≈ 0.494

Среднее арифметическое значение метрик:

* Средняя Recall = (0.933 + 0.866 + 0.513) / 3 ≈ 0.771
* Средняя Precision = (0.841 + 0.976 + 0.476) / 3 ≈ 0.764
* Средняя F1 = (0.885 + 0.917 + 0.494) / 3 ≈ 0.765

Для вычисления средневзвешенных значений используем количество истинных примеров (TP + FN) для каждого класса в качестве весов:

Вес для класса 0 = 119

Вес для класса 1 = 142

Вес для класса 2 = 39

Общий вес = 119 + 142 + 39 = 300

Средневзвешенные метрики:

* Recall\_w = (119 \* 0.933 + 142 \* 0.866 + 39 \* 0.513) / 300 ≈ 0.850
* Precision\_w = (119 \* 0.841 + 142 \* 0.976 + 39 \* 0.476) / 300 ≈ 0.852
* F1\_w = (119 \* 0.885 + 142 \* 0.917 + 39 \* 0.494) / 300 ≈ 0.860

Обоснование выбора метрики:

Учитывая специфику задачи диагностики болезни, наиболее важной метрикой может быть полнота (Recall), особенно для класса 0, так как пропуск случаев отсутствия болезни (ложноотрицательные результаты) может привести к нежелательным последствиям. Высокая полнота означает, что большинство настоящих случаев болезни будет выявлено.

Основные подходы для усовершенствования классификации:

* Балансировка классов: Применение методов для балансировки данных, таких как oversampling (увеличение числа примеров для недопредставленных классов) или undersampling (уменьшение числа примеров для перепредставленных классов).
* Улучшение модели: Использование более сложных моделей, таких как ансамблевые методы (Random Forest, Gradient Boosting), которые могут лучше улавливать сложные закономерности в данных.
* Feature Engineering: Создание новых признаков (features), которые могут улучшить способность модели различать классы.
* Cross-validation: Применение методов перекрестной проверки для обеспечения устойчивости и надежности результатов модели.

# **Задание 7.**

# *Используя как можно меньше операций в исходном массиве "шахматная доска" np.zeros((8,8)), добиться расстановки кодов "шахматных фигур".*

*Ожидаемый результат:*

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Код:

import numpy as np

# Создаем пустую доску

board = np.zeros((8, 8))

# Расставляем фигуры

board[0, :] = [-4, 3, 2, -6, -5, -2, 3, -4] # Белые фигуры

board[1, :] = -1 # Белые пешки

board[6, :] = 1 # Черные пешки

board[7, :] = [4, 3, 2, 6, 5, 2, 3, 4] # Черные фигуры

print(board)

# **Задание 8.**

# *Создайте список, содержащий по 3 объекта каждого класса (базовый класс People (атрибуты имя и возраст), производный класс Worker, имеющий атрибуты: должность (post) и зарплата (salary), и производный класс Student, имеющий атрибуты оценки (marks) и текущие изучаемые дисциплины (subjects). Для списка объектов:*

*• выведите информацию о каждом человеке с помощью метода info;*

*• выведите фамилии тех, кто моложе 30 лет. Выведите количество объектов Worker, имеющих зарплату ниже среднего. Выведите информацию о дисциплинах каждого объекта Student.*

Код:

class People:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def info(self):

print("Name:", self.name)

print("Age:", self.age)

class Worker(People):

def \_\_init\_\_(self, name, age, post, salary):

super().\_\_init\_\_(name, age)

self.post = post

self.salary = salary

class Student(People):

def \_\_init\_\_(self, name, age, marks, subjects):

super().\_\_init\_\_(name, age)

self.marks = marks

self.subjects = subjects

# Создаем список объектов

people\_list = [

Worker("John Doe", 25, "Developer", 50000),

Worker("Jane Smith", 35, "Manager", 70000),

Worker("Michael Johnson", 28, "Designer", 45000),

Student("Alice Brown", 20, {"Math": 90, "English": 85}, ["Math", "English", "Physics"]),

Student("Robert Lee", 22, {"Math": 75, "English": 80}, ["Chemistry", "Biology", "History"]),

Student("Emily Wang", 26, {"Math": 95, "English": 92}, ["Computer Science", "Statistics", "Economics"])

]

# Выводим информацию о каждом человеке

for person in people\_list:

person.info()

print()

# Выводим фамилии тех, кто моложе 30 лет

print("People younger than 30:")

for person in people\_list:

if person.age < 30:

print(person.name.split()[-1])

# Находим количество объектов Worker с зарплатой ниже среднего

worker\_salaries = [worker.salary for worker in people\_list if isinstance(worker, Worker)]

average\_salary = sum(worker\_salaries) / len(worker\_salaries)

below\_average\_count = sum(1 for salary in worker\_salaries if salary < average\_salary)

print("Number of Workers with salary below average:", below\_average\_count)

# Выводим информацию о дисциплинах каждого объекта Student

print("Subjects for each Student:")

for person in people\_list:

if isinstance(person, Student):

print(person.name + "'s subjects:", person.subjects)

# **Задание 9.**

# *Гидрометцентр оповестил о снегопаде в г. Москве. Для того, чтобы могли расчистить дороги, следует выяснить, сколько снегоуборочных машин им потребуется. Известно, что на каждый сантиметр слоя выпавшего снега требуется 50 снегоуборочных машин и 20 машин для вывоза снега.  Помимо этого, следует учитывать загруженность дорог в Москве. На каждый балл загруженности дорог (от 0 до 10 включительно) следует выпускать на 5% больше снегоуборочных машин и машин для вывоза снега. Рассчитать сколько машин потребуется при прогнозируемом количестве осадков и ситуации на дорогах в г. Москве. Решите данную задачу в парадигме ООП (объектно-ориентированного программирования). Создать класс c полями: день недели, прогнозируемое количество осадков, пробки, а также метод для расчета необходимого количества машин для уборки в текущий день и метод вывода результатов на экран.*

Код:

class SnowRemoval:

def \_\_init\_\_(self, day\_of\_week, snowfall, traffic):

# Инициализируем объект класса с полями: день недели, количество осадков и уровень пробок

self.day\_of\_week = day\_of\_week

self.snowfall = snowfall

self.traffic = traffic

def calculate\_machines(self):

# Метод для расчета необходимого количества машин для уборки

# Каждый сантиметр снега требует 50 снегоуборочных машин и 20 машин для вывоза снега

# Учитываем уровень пробок: на каждый балл загруженности дорог добавляем 5% машин

snow\_removal\_machines = 50 \* self.snowfall \* (1 + 0.05 \* self.traffic)

snow\_transport\_machines = 20 \* self.snowfall \* (1 + 0.05 \* self.traffic)

return snow\_removal\_machines, snow\_transport\_machines

def display\_results(self):

# Метод для вывода результатов на экран

removal, transport = self.calculate\_machines()

print(f"On {self.day\_of\_week}:")

print(f"Snow removal machines needed: {removal}")

print(f"Snow transport machines needed: {transport}")

# Пример использования класса

if \_\_name\_\_ == "\_\_main\_\_":

# Создаем объекты для двух разных дней

monday = SnowRemoval("Monday", 5, 6) # Понедельник: 5 см снега, пробки на уровне 6

friday = SnowRemoval("Friday", 10, 8) # Пятница: 10 см снега, пробки на уровне 8

# Выводим результаты для каждого дня

monday.display\_results()

friday.display\_results()

# **Задание 10.**

# *Решите данную задачу в парадигме ООП (объектно-ориентированного программирования). Программа должна вычислять цену билетов в маршрутке для пассажиров разных категорий. Если возраст пассажира до 3-х лет, то бесплатный проезд. Если возраст пассажира 7-18 лет (школьник), то ему предоставляется скидка 50%. Также пенсионерам предполагается скидка 50%, а льготники имеют бесплатный проезд. Цена билета также зависит от расстояния, которое проезжает пассажир: до 20 км – стоимость 100руб; от 21 до 100 км – стоимость 150руб; от 101 км до 200 км – стоимость 200 рублей; >=201 км – стоимость 400руб.*

Код:

class TicketPriceCalculator:

def \_\_init\_\_(self, age, distance):

self.age = age

self.distance = distance

def calculate\_price(self):

# Определяем цену в зависимости от расстояния

if self.distance < 20:

price = 100

elif 20 <= self.distance <= 100:

price = 150

elif 101 <= self.distance <= 200:

price = 200

else:

price = 400

# Применяем скидки в зависимости от возраста и категории пассажира

if self.age < 3:

final\_price = 0 # Бесплатно для детей до 3 лет

elif 7 <= self.age <= 18:

final\_price = price / 2 # Скидка 50% для школьников

elif self.age >= 65:

final\_price = price / 2 # Скидка 50% для пенсионеров

else:

final\_price = price # Без скидок для взрослых

return final\_price

# Пример использования класса

if \_\_name\_\_ == "\_\_main\_\_":

age = int(input("Введите возраст пассажира: "))

distance = int(input("Введите расстояние поездки (в км): "))

calculator = TicketPriceCalculator(age, distance)

price = calculator.calculate\_price()

print(f"Цена билета: {price} руб.")

# **Задание 11.**

# *Даны 2 кольцевых списка с фамилиями шахматистов 2-х команд. Произвести жеребьевку. В первой команде выбирается каждый n-й игрок, а во второй - каждый k-й. Для решения задачи использовать ООП.*

Код:

class ChessTeam:

def \_\_init\_\_(self):

self.players = []

def add\_player(self, name):

self.players.append(name)

def draw\_lots(self, n, k):

# Выбираем каждого n-го игрока из первой команды

first\_team = [self.players[i % len(self.players)] for i in range(0, len(self.players), n)]

# Выбираем каждого k-го игрока из второй команды

second\_team = [self.players[i % len(self.players)] for i in range(0, len(self.players), k)]

return first\_team, second\_team

# Пример использования класса

if \_\_name\_\_ == "\_\_main\_\_":

# Создаем кольцевые списки шахматистов для каждой команды

team1 = ChessTeam()

team1.add\_player("Player1")

team1.add\_player("Player2")

team1.add\_player("Player3")

team1.add\_player("Player4")

team2 = ChessTeam()

team2.add\_player("Player5")

team2.add\_player("Player6")

team2.add\_player("Player7")

team2.add\_player("Player8")

# Задаем значения n и k для жеребьевки

n = 2

k = 3

# Производим жеребьевку

first\_team, second\_team = team1.draw\_lots(n, k)

# Выводим результаты

print("Первая команда:")

print(first\_team)

print("Вторая команда:")

print(second\_team)

# **Задание 12.**

# *Даны 2 списка: перечень товаров и фамилии покупателей. Каждый n-й покупатель покупает m-й товар. Вывести список покупок. Для решения задачи использовать ООП.*

Код:

class ShoppingList:

def \_\_init\_\_(self):

self.products = []

self.customers = []

def add\_product(self, product):

self.products.append(product)

def add\_customer(self, customer):

self.customers.append(customer)

def generate\_shopping\_list(self, n, m):

shopping\_list = []

for i in range(0, len(self.customers), n):

for j in range(i, min(i + n, len(self.customers))):

for k in range(j \* m, min((j + 1) \* m, len(self.products))):

shopping\_list.append((self.customers[j], self.products[k]))

return shopping\_list

# Пример использования класса

if \_\_name\_\_ == "\_\_main\_\_":

# Создаем объект класса ShoppingList

shopping\_list = ShoppingList()

# Добавляем товары

shopping\_list.add\_product("Товар1")

shopping\_list.add\_product("Товар2")

shopping\_list.add\_product("Товар3")

shopping\_list.add\_product("Товар4")

# Добавляем покупателей

shopping\_list.add\_customer("Покупатель1")

shopping\_list.add\_customer("Покупатель2")

shopping\_list.add\_customer("Покупатель3")

shopping\_list.add\_customer("Покупатель4")

shopping\_list.add\_customer("Покупатель5")

# Указываем значения n и m для создания списка покупок

n = 2

m = 2

# Генерируем список покупок

purchases = shopping\_list.generate\_shopping\_list(n, m)

# Выводим список покупок

print("Список покупок:")

for purchase in purchases:

print(f"{purchase[0]} купил(а) {purchase[1]}")

# **Задание 13.**

# *Реализовать класс Стек с методами push, pop, top (определение вершины стека) и is\_empty (проверка пуст стек или нет) с помощью ООП. Удалить все элементы стека, равные первому.*

Код:

class Stack:

def \_\_init\_\_(self):

self.stack = []

def push(self, item):

self.stack.append(item)

def pop(self):

if not self.is\_empty():

return self.stack.pop()

else:

print("Стек пустой. Невозможно выполнить операцию pop.")

return None

def top(self):

if not self.is\_empty():

return self.stack[-1]

else:

print("Стек пустой. Вершины нет.")

return None

def is\_empty(self):

return len(self.stack) == 0

def remove\_all\_equal\_to\_first(self):

if not self.is\_empty():

first\_item = self.stack[0]

self.stack = [item for item in self.stack if item != first\_item]

print(f"Удалены все элементы, равные первому ({first\_item}).")

else:

print("Стек пустой. Нечего удалять.")

# Пример использования класса

if \_\_name\_\_ == "\_\_main\_\_":

stack = Stack()

# Добавляем элементы в стек

stack.push(5)

stack.push(3)

stack.push(7)

stack.push(5)

stack.push(9)

# Выводим вершину стека

print("Вершина стека:", stack.top())

# Удаляем все элементы, равные первому

stack.remove\_all\_equal\_to\_first()

# Печатаем стек после удаления элементов

print("Элементы стека после удаления:")

while not stack.is\_empty():

print(stack.pop())

# **Задание 14.**

# *Решить задачу поиска наибольшего общего делителя (НОД) в двух парадигмах: структурной и функциональной. Показать на примере программного кода основные особенности данных парадигм.*

Структурная парадигма:

class GCD:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def find\_gcd(self):

while self.b:

self.a, self.b = self.b, self.a % self.b

return self.a

# Пример использования класса

if \_\_name\_\_ == "\_\_main\_\_":

gcd = GCD(24, 36)

result = gcd.find\_gcd()

print("Наибольший общий делитель:", result)

В этом примере мы создаем класс GCD, который содержит метод find\_gcd, выполняющий алгоритм Евклида для нахождения наибольшего общего делителя. Мы передаем два числа в конструктор класса и вызываем метод find\_gcd для получения результата.

Функциональная парадигма:

def gcd(a, b):

while b:

a, b = b, a % b

return a

# Пример использования функции

if \_\_name\_\_ == "\_\_main\_\_":

result = gcd(24, 36)

print("Наибольший общий делитель:", result)

В функциональной парадигме мы определяем функцию gcd, которая выполняет тот же алгоритм Евклида для нахождения наибольшего общего делителя. Мы передаем два числа в функцию и получаем результат.

Основная особенность структурной парадигмы заключается в том, что мы описываем структуры данных и операции над ними с помощью классов и методов. В функциональной парадигме мы определяем функции, которые выполняют операции над данными без необходимости создавать объекты классов.

# **Задание 15.**

# *Спроектировать базу данных к 3НФ. База данных должна содержать следующую информацию: ФИО преподавателя, должность, номер группы, название кафедры, номер аудитории, число мест в аудитории, количество студентов в группе, специальность обучения, название предмета, вид занятий (лекция, практика, лабораторная работа), семестр.*

Для проектирования базы данных к 3НФ (третьей нормальной форме) необходимо разделить информацию на связанные таблицы таким образом, чтобы каждый столбец содержал только атомарные значения, а также чтобы все неключевые атрибуты были функционально зависимы от ключа. В данном случае, я предлагаю следующую структуру таблиц:

-Таблица "Преподаватели":

* ID (идентификатор преподавателя, первичный ключ)
* ФИО преподавателя
* Должность
* Название кафедры

-Таблица "Группы":

* ID (идентификатор группы, первичный ключ)
* Номер группы
* Количество студентов в группе
* Специальность обучения

-Таблица "Аудитории":

* ID (идентификатор аудитории, первичный ключ)
* Номер аудитории
* Число мест в аудитории

-Таблица "Предметы":

* ID (идентификатор предмета, первичный ключ)
* Название предмета
* Вид занятий (лекция, практика, лабораторная работа)
* Семестр

-Таблица "Расписание":

* ID (идентификатор записи расписания, первичный ключ)
* ID преподавателя (внешний ключ, ссылается на таблицу "Преподаватели")
* ID группы (внешний ключ, ссылается на таблицу "Группы")
* ID аудитории (внешний ключ, ссылается на таблицу "Аудитории")
* ID предмета (внешний ключ, ссылается на таблицу «Предметы»)

Такая структура позволит эффективно хранить информацию о преподавателях, группах, аудиториях, предметах и расписании занятий, при этом избегая избыточности данных и обеспечивая соответствие третьей нормальной форме.

# **Задание 16.**

# *Сгенерировать двумерный массив arr размерности (4, 7), состоящий из случайных действительных чисел, равномерно распределенных в диапазоне от 0 до 20. Суммировать значения каждой строки массива arr. Для этого создать новый массив arr\_sum, в котором к исходному массиву arr добавлен еще один столбец справа, содержащий суммарные значения каждой строки массива arr. Суммарные значения округлить (с помощью np.round()) до двух десятичных знаков. Полученный результат отсортируйте по столбцу суммы по убыванию. Использовать numpy.*

Код:

import numpy as np

# Генерация двумерного массива arr размерности (4, 7) из случайных чисел в диапазоне от 0 до 20

arr = np.random.uniform(0, 20, (4, 7))

# Вычисление суммы значений каждой строки массива arr

row\_sums = np.sum(arr, axis=1)

# Создание нового массива arr\_sum с дополнительным столбцом с суммарными значениями каждой строки

arr\_sum = np.column\_stack((arr, np.round(row\_sums, 2)))

# Сортировка массива arr\_sum по столбцу с суммами по убыванию

arr\_sum\_sorted = arr\_sum[arr\_sum[:, -1].argsort()[::-1]]

print("Исходный массив arr:")

print(arr)

print("\nМассив arr\_sum с суммарными значениями и отсортированный по суммам:")

print(arr\_sum\_sorted)

Этот код генерирует случайный двумерный массив arr размерности (4, 7) с помощью метода np.random.uniform(), затем вычисляет суммы значений каждой строки с помощью np.sum(). Создается новый массив arr\_sum, в котором к исходному массиву arr добавляется дополнительный столбец с суммарными значениями каждой строки. Затем массив arr\_sum сортируется по столбцу с суммами по убыванию с помощью метода argsort().

# **Задание 17.**

# *Спроектировать ER-модель базы данных "Books" с обязательными полями (id\_book, title (название книги), author (автор), genre (жанр)). Дополните базу данных необходимыми полями. Напишите SQL-запросы, позволяющие осуществить:*

1. *­вывод списка авторов без повторений*
2. *вывод списка авторов, у которых фамилия содержит "ов"*
3. *вывод списка книг, у которых количество страниц меньше 100 и жанр – повесть*

ER-модель:

+--------------+------------------+------+-----+---------+

| Field | Type | Null | Key | Default |

+--------------+------------------+------+-----+---------+

| id\_book | int(10) unsigned | NO | PRI | NULL |

| title | varchar(255) | NO | | NULL |

| author | varchar(255) | NO | | NULL |

| genre | varchar(255) | NO | | NULL |

| publication | date | YES | | NULL |

| pages | int(11) | YES | | NULL |

| language | varchar(50) | YES | | NULL |

| publication | date | YES | | NULL |

| publisher\_id | int(10) unsigned | YES | MUL | NULL |

+--------------+------------------+------+-----+---------+

а) Вывод списка авторов без повторений:

SELECT DISTINCT author FROM Books;

б) Вывод списка авторов, у которых фамилия содержит "ов":

SELECT DISTINCT author

FROM Books

WHERE author LIKE '%ов%';

c) Вывод списка книг, у которых количество страниц меньше 100 и жанр – повесть:

SELECT \* FROM Books

WHERE pages < 100 AND genre = 'повесть';

# **Задание 18.**

# *Смоделировать процесс получения книги в библиотеке в нотации IDEF0 с указанием предметной области и диаграммы декомпозиций.*

Процесс "Получение книги в библиотеке".

Входы:

* Выбор книги и формирование заявки.
* Поиск книги в каталоге.
* Подготовка книги к выдаче.

Управление:

* Нет прямого управления в данном процессе.

Выходы:

* Выдача книги пользователю.

(в таком виде должна быть(пример):

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание)

Декомпозиция процессов:

A1. Выбор книги и формирование заявки

* Входы: Запрос, Карта читателя, Каталог книг
* Управление: Правила и требования библиотеки, Инструкция по выдаче книг
* Выходы: Заявка на книгу

A2. Поиск книги в каталоге

* Входы: Заявка на книгу, Каталог книг
* Управление: Инструкция по поиску книг
* Выходы: Найденная книга, Информация о доступности

A3. Подготовка книги к выдаче

* Входы: Найденная книга, Информация о доступности
* Управление: Правила и требования библиотеки
* Выходы: Подготовленная книга

A4. Выдача книги пользователю

* Входы: Подготовленная книга, Карта читателя
* Управление: График работы библиотеки, Инструкция по выдаче книг
* Выходы: Полученная книга, Документация

# **Задание 19.**

# *Смоделировать процесс защиты выпускной квалификационной работы в нотации IDEF0 с указанием предметной области и диаграммы декомпозиций.*

+----------------------------------------------------------+

| |

| A0 Защита ВКР |

| |

+----+----------------------+--------------------------+---+

| | |

| | |

| | |

v v v

Входы: Управление: Выходы:

- Тема работы - Правила и - Оценка работы

- ВКР требования - Решение комиссии

- Презентация к защите - Защищенная работа

- Расписание защит

- Требования к презентации

- Критерии оценки

Декомпозиция процессов:

Подготовка студента к защите (A1)

* Входы: Тема работы, ВКР, Презентация
* Управление: Правила и требования к защите, Требования к презентации
* Выходы: Готовая ВКР, Презентация

Оценка работы (A2)

* Входы: Готовая ВКР
* Управление: Критерии оценки, Правила и требования к защите
* Выходы: Предварительная оценка, Рецензия

Проведение защиты (A3)

* Входы: Готовая ВКР, Презентация, Предварительная оценка
* Управление: Расписание защит, Критерии оценки
* Выходы: Ответы на вопросы комиссии, Демонстрация результатов

Принятие решения о защите (A4)

* Входы: Ответы на вопросы комиссии, Демонстрация результатов
* Управление: Критерии оценки
* Выходы: Оценка работы, Решение комиссии, Защищенная работа

# **Задание 20.**

# *Смоделировать процесс покупки товара в нотации DFD.*

Диаграмма DFD (Data Flow Diagram) моделирует процесс покупки товара, показывая поток данных между различными элементами системы. Мы будем использовать четыре основных компонента DFD: процессы, хранилища данных, внешние сущности и потоки данных.

Контекстная диаграмма:

Изображение выглядит как снимок экрана, текст, линия, Шрифт

Автоматически созданное описаниеДекомпозиция:

Изображение выглядит как снимок экрана, текст

Автоматически созданное описаниеОписание процессов:

1.1 Обработка заказа

* Входы: Заказ от клиента
* Выходы: Уведомление о заказе в 1.2 Подтверждение оплаты, Статус заказа в 1.3 Уведомление о доставке

1.2 Подтверждение оплаты

* Входы: Уведомление о заказе от 1.1 Обработка заказа
* Выходы: Обновленная база данных заказов

1.3 Уведомление о доставке

* Входы: Статус заказа от 1.1 Обработка заказа
* Выходы: Уведомление клиента о доставке

# **Задание 21.**

# *Спроектировать схему базы данных по текстовому описанию. Обосновать нормальность отношений. В воображаемой социальной сети есть Пользователи (id, имя), Фото (id, название, автор) и Комментарии К Фото (id, текст, автор, к какому Фото относится). Необходимо добавить возможность для Пользователей ставить лайки другим Пользователям, Фото или Комментариям К Фото. Нужно реализовать такие возможности: пользователь не может поставить 2 лайка одной и той же сущности (например одному и тому же Фото); пользователь может отозвать лайк; необходимо иметь возможность посчитать число полученных сущностью лайков и вывести список Пользователей, поставивших лайки; в будущем могут появиться новые виды сущностей которые можно лайкать*

Пользователи (Users)

* id (INT, PRIMARY KEY)
* name (VARCHAR)

Фото (Photos)

* id (INT, PRIMARY KEY)
* title (VARCHAR)
* author\_id (INT, FOREIGN KEY REFERENCES Users(id))

Комментарии (Comments)

* id (INT, PRIMARY KEY)
* text (VARCHAR)
* author\_id (INT, FOREIGN KEY REFERENCES Users(id))
* photo\_id (INT, FOREIGN KEY REFERENCES Photos(id))

Лайки (Likes)

* id (INT, PRIMARY KEY)
* user\_id (INT, FOREIGN KEY REFERENCES Users(id))
* entity\_type (VARCHAR) -- Тип сущности (например, "User", "Photo", "Comment")
* entity\_id (INT) -- ID сущности
* UNIQUE (user\_id, entity\_type, entity\_id) -- Гарантирует, что пользователь не может поставить 2 лайка одной и той же сущности

a) Вывод списка Пользователей, поставивших лайки:

SELECT DISTINCT u.id, u.name

FROM Users u

JOIN Likes l ON u.id = l.user\_id;

b) Вывод числа полученных сущностью лайков:

SELECT entity\_type, entity\_id, COUNT(\*) as like\_count

FROM Likes

GROUP BY entity\_type, entity\_id;

# **Задание 22.**

# *Выполнить:*

# *перевод десятичного представления IP адреса в двоичный - 173.143.32.194 (5 баллов);*

* *двоичного представления IP адреса в десятичный - 01011101.10111011.01001000.00110000;*
* *записать маску (в двоичном и десятичном представлении) для проекта сети: сеть 192.168.0.0. должна содержать 4 подсетей и 60 хостов;*
* *определить классы предложенных в задании сетей.*

IP адрес: 173.143.32.194

Перевод каждого октета в двоичный:

173: 10101101

143: 10001111

32: 00100000

194: 11000010

Двоичное представление: 01011101.10111011.01001000.00110000

Перевод каждого октета в десятичный:

01011101: 93

10111011: 187

01001000: 72

00110000: 48

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

# **Задание 23.**

# *С помощью модуля pandas создать csv-файл students.csv, содержащий информацию о студентах и их оценках по дисциплинам.*

# *Определить:*

*– всех студентов с оценкой «пять» по программированию;*

*– всех студентов с оценкой «пять» по программированию и математике;*

*– всех студентов младше 20 лет;*

*– всех студентов по фамилии Иванов.*

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

Автоматически созданное описание

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

Автоматически созданное описание

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

Автоматически созданное описание

# **Задание 24.**

# *Дана сеть: 40.212.114.0/24. Разбейте сеть на подсети меньшего размера (не менее двух вариантов). Определите максимальное количество хостов в каждой такой сети и максимальное количество сетей. Определите широковещательный адрес для каждой подсети.*

Разделение на подсети с маской /26

Маска /26 позволяет создать 4 подсети, каждая из которых может иметь до 62 хостов (64 адреса - 2 служебных).

Маска: 255.255.255.192 (или /26)

Количество подсетей: 4

Максимальное количество хостов в каждой подсети: 62

Широковещательные адреса для каждой подсети:

Первая подсеть:

Сеть: 40.212.114.0/26

Диапазон адресов: 40.212.114.0 - 40.212.114.63

Широковещательный адрес: 40.212.114.63

Вторая подсеть:

Сеть: 40.212.114.64/26

Диапазон адресов: 40.212.114.64 - 40.212.114.127

Широковещательный адрес: 40.212.114.127

Третья подсеть:

Сеть: 40.212.114.128/26

Диапазон адресов: 40.212.114.128 - 40.212.114.191

Широковещательный адрес: 40.212.114.191

Четвертая подсеть:

Сеть: 40.212.114.192/26

Диапазон адресов: 40.212.114.192 - 40.212.114.255

Широковещательный адрес: 40.212.114.255

# **Задание 25.**

# *В контексте повышения производительности системы обработки данных возникла задача выбора наиболее подходящего алгоритма сортировки. Требуется провести анализ трех популярных алгоритмов сортировки: пузырьковой, вставками и быстрой сортировки. Оценить их производительность на наборах данных различного объема и степени упорядоченности, чтобы определить наиболее эффективный алгоритм для использования в системе.*

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

import numpy as np

import time

import pandas as pd

def bubble\_sort(arr):

n = len(arr)

comparisons = 0

swaps = 0

for i in range(n):

for j in range(0, n-i-1):

comparisons += 1

if arr[j] > arr[j+1]:

arr[j], arr[j+1] = arr[j+1], arr[j]

swaps += 1

return comparisons, swaps

def insertion\_sort(arr):

n = len(arr)

comparisons = 0

swaps = 0

for i in range(1, n):

key = arr[i]

j = i - 1

while j >= 0 and key < arr[j]:

comparisons += 1

arr[j + 1] = arr[j]

j -= 1

swaps += 1

arr[j + 1] = key

if j >= 0:

comparisons += 1

return comparisons, swaps

def quick\_sort(arr):

comparisons = 0

swaps = 0

def \_quick\_sort(arr, low, high):

nonlocal comparisons, swaps

if low < high:

pi, comps, swps = partition(arr, low, high)

comparisons += comps

swaps += swps

\_quick\_sort(arr, low, pi-1)

\_quick\_sort(arr, pi+1, high)

def partition(arr, low, high):

comps = 0

swps = 0

pivot = arr[high]

i = low - 1

for j in range(low, high):

comps += 1

if arr[j] < pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

swps += 1

arr[i+1], arr[high] = arr[high], arr[i+1]

swps += 1

return i+1, comps, swps

\_quick\_sort(arr, 0, len(arr)-1)

return comparisons, swaps

def generate\_datasets():

sizes = [10, 100, 1000, 10000]

datasets = {}

for size in sizes:

datasets[f'random\_{size}'] = np.random.randint(0, 100, size)

datasets[f'sorted\_{size}'] = np.arange(size)

datasets[f'reverse\_sorted\_{size}'] = np.arange(size, 0, -1)

return datasets

def analyze\_algorithms(datasets):

results = []

for name, data in datasets.items():

for algorithm in [bubble\_sort, insertion\_sort, quick\_sort]:

arr = data.copy()

start\_time = time.time()

comparisons, swaps = algorithm(arr)

elapsed\_time = time.time() - start\_time

results.append({

'dataset': name,

'algorithm': algorithm.\_\_name\_\_,

'time': elapsed\_time,

'comparisons': comparisons,

'swaps': swaps

})

return pd.DataFrame(results)

datasets = generate\_datasets()

results = analyze\_algorithms(datasets)

print(results)