Требования к данному заданию:

- **мы принимаем готовое тестовое задание только в формате ссылки на GitHu**
- быть готовым на собеседовании ответить на любой вопрос по отправленному решению
- права на редактирование данного задания получить нельзя, запрос отправлять бессмысленно
- в конце заданию даны рекомендации для подготовки к собеседованию

Блок 1: Теория вероятности и логика

Задание 1: Фермер

На ферме содержатся шесть разных видов животных, и каждый раз, когда фермер заходит в сарай, он видит одно случайное животное. За день фермер заходит в сарай 6 раз.

Каково математическое ожидание количества разных видов животных, которые фермер увидит за день?

Ответ округлить до сотых, например: 4,12

Мой ответ:

Вероятность P(увидеть 1 одно животное) = 1-(⅚)^6 = 0,6651

Мат.ожидание симметрично E(S) = 6*0,6651 = 3,99

Задание 2: Кулинарное соревнование

В конкурсе участвуют 80 шеф-поваров с уникальными уровнями мастерства. В первом этапе судьи случайным образом распределяют их по парам (в любом состязании двух шефов выигрывает тот, у кого выше уровень мастерства). На втором этапе шефы снова случайно образуют пары для финального раунда (пары могут повториться). Победную награду получают те, кто выиграл в обоих этапах.

Каково математическое ожидание числа победителей?

Ответ округлить до десятых, например: 33,5

Мой ответ:

Т.к. события независимы вероятность победить в обоих этапах

P(победителя) = ½*½=0,25

Тогда мат.ожидание числа победителей

E = 80*0,25 = 20,0 (ответ округлен до десятых)

Задание 3: Одинокая дорога

На пустынном шоссе вероятность появления автомобиля за 30-минутный период составляет 0.95.

Какова вероятность его появления за 10 минут? А за 27 минут?

Ответ дать в процентах, округлив до десятых через точку с запятой, например: 42,7; 95,0

Мой ответ: Поток автомобилей на пустынном шоссе - это случайные независимые события(пуассоновский поток). Значит вероятность появления за интервал времени t растет нелинейно, и считается через вероятность отсутствия.
Если P(хотя бы 1 авто)=0.95,
значит вероятность, что не будет ни одного авто за 30 минут: P0(30)=1-0.95=0.05
P0(t) = $e^{-\lambda t}$
$e^{-\lambda*30}$**=0.05**

Интенсивность $\lambda = \frac{ln(0.05)}{30} = 0.099856$

Вероятность за 10мин: $P(10) = 1 - e^{-0.99856} = 0.632$

Вероятность за 27мин: $P(27) = 1 - e^{-2.6961} = 0.9325$

Ответ: 63,2; 93,3.

Блок 2: Python

Задание 1: Изоморфизмы

Реализовать функцию (или тело функции), которая проверяет на изоморфность два слова. Пояснение: строки s и t называются изоморфными, если все вхождения каждого символа строки s можно последовательно заменить другим символом и получить строку t. Порядок символов при этом должен сохраняться, а замена — быть уникальной. Так, два разных символа строки s нельзя заменить одним и тем же символом из строки t, а вот одинаковые символы в строке s должны заменяться одним и тем же символом.

# Пример:
s = 'paper'
t = 'title'
print(is_isomorphic(s, t))
# Вывод:
True

Оценить оптимальность решения по времени и памяти и прикрепить текст кода.

```
def is_isomorphic(s: str, t: str) -> bool:
    """
    Проверяет, являются ли строки s и t изоморфными.

    Изоморфизм означает, что существует взаимно однозначное соответствие
    между символами s и t с сохранением порядка.
    """
    if len(s) != len(t):
        return False

    # Прямое отображение: символ из s -> символ из t
    s_to_t = {}
    # Множество уже использованных символов в t
    used_in_t = set()
```

```
    for cs, ct in zip(s, t):
        if cs in s_to_t:
            # Если символ уже отображали, проверяем, что отображается в тот же
            if s_to_t[cs] != ct:
                return False
        else:
            # Если символ из t уже использован для другого символа из s
            if ct in used_in_t:
                return False
            s_to_t[cs] = ct
            used_in_t.add(ct)

    return True

# Пример использования
s = 'paper'
t = 'title'
print(is_isomorphic(s, t))  # True

# Проверка на антипример
s2 = 'ab'
t2 = 'aa'
print(is_isomorphic(s2, t2))  # False
```

Output

True

False

Алгоритм проходит по каждому символу ровно один раз — это линейное время O(n). Внутри цикла все операции константны, потому что словари и множества в Python работают в среднем за O(1). По памяти мы храним два контейнера размером до количества уникальных символов. В худшем случае это тоже O(n), но обычно алфавит ограничен, поэтому можно говорить об O(1). Теоретический минимум для этой задачи — O(n), потому что нужно проверить каждый символ. Мое решение этого минимума достигает.

Задание 2: Натуральная последовательность

Реализовать функцию (или тело функции), которая находит единственное отсутствующее число из последовательности натуральных чисел 1,2,…,n.

```
# Пример:
nums = [1, 2, 3, 4, 5, 6, 8, 9, 10, 11]
print(missing_number(nums))
# Вывод:
7
```

Оценить оптимальность решения по времени и памяти и прикрепить текст кода.

```python
def missing_number(nums):
    """
    Находит единственное пропущенное число в последовательности 1..n.

    Аргументы:
        nums: список чисел от 1 до n, где одно число пропущено

    Возвращает:
        Пропущенное число
    """
    n = len(nums) + 1
    # Сумма арифметической прогрессии от 1 до n
    expected_sum = n * (n + 1) // 2
    # Сумма имеющихся чисел
```

```
    actual_sum = sum(nums)

    # Разница — пропущенное число

    return expected_sum - actual_sum


# Пример использования

nums = [1, 2, 3, 4, 5, 6, 8, 9, 10, 11]
print(missing_number(nums))  # 7
```

Output

7

Время: O(n), где n — длина исходного массива плюс 1. Алгоритм вычисляет сумму всех элементов массива за один проход, что требует линейного времени.

Память: O(1) — используется только несколько переменных для хранения промежуточных значений, не зависящих от размера входных данных.

Данное решение оптимально, так как мы не можем найти пропущенное число быстрее, чем просмотрев все элементы массива (нужно убедиться, что видели все числа). Память используется минимально возможная.

Задание 3: Факторизация

Реализовать функцию (или тело функции), которая при введении натурального числа n разбивает его на простые множители (представить его в виде простых чисел).

# Пример:
n = 56
print(prime_factors(n))
# Вывод:
[2, 2, 2, 7]

Оценить оптимальность решения по времени и памяти и прикрепить текст кода.

```
def prime_factors(n):

    """

    Разлагает натуральное число n на простые множители.
```

```
    Аргументы:
        n: натуральное число

    Возвращает:
        Список простых множителей в порядке возрастания
    """
    if n < 2:
        return []

    factors = []

    # Обрабатываем делитель 2 отдельно (чтобы потом шагать по 2)
    while n % 2 == 0:
        factors.append(2)
        n //= 2

    # Проверяем нечетные делители от 3 до sqrt(n)
    d = 3
    while d * d <= n:
        while n % d == 0:
            factors.append(d)
            n //= d
        d += 2

    # Если осталось число больше 1 — оно простое
    if n > 1:
        factors.append(n)
```

```
    return factors


# Примеры использования
print(prime_factors(56))   # [2, 2, 2, 7]
print(prime_factors(13))   # [13]
print(prime_factors(100))  # [2, 2, 5, 5]
print(prime_factors(1))    # []
```

Output

[2, 2, 2, 7]

[13]

[2, 2, 5, 5]

[]

Время: O(квадратный корень из n) в худшем случае. Алгоритм проверяет делители до квадратного корня из n. Если n простое, мы проверим все числа до квадратного корня из n.

Память: O(log n) в худшем случае — количество простых множителей.

Данное решение близко к оптимальному для данной задачи. Оптимизация с шагом 2 после обработки двойки сокращает количество проверок примерно вдвое.

Блок 3: SQL

Задание 1: Абитуриенты

Есть таблица examination с двумя полями: id (id абитуриента), scores (кол-во набранных баллов дополнительного вступительного испытания от 0 до 100).

Требуется реализовать запрос, который создаёт колонку с позицией абитуриента в общем рейтинге.

```
SELECT
    id,
    scores,
```

```
    DENSE_RANK() OVER (ORDER BY scores DESC) as position
FROM examination
ORDER BY scores DESC, id;
```

```
id | scores | position
1  | 100    | 1
2  | 95     | 2
3  | 95     | 2
4  | 90     | 3   -- ранг 3, а не 4
```

Одинаковые баллы — одинаковый ранг, следующий ранг без пропусков. Сделал так, потому что не понял по заданию (Как ранжировать абитуриентов с одинаковыми баллами). И по внутренней справедливости использовал DENSE_RANK, придав одинаковый ранг абитуриентам с одинаковыми баллами, без пропусков.

Задание 2: FULL JOIN

Представьте две таблицы: первая содержит 30 строк, а вторая — 20 строк. Мы выполняем операцию FULL JOIN между ними.

Какой диапазон возможного количества строк может быть в результирующей таблице, если учесть, что ключи для соединения могут быть как полностью совпадающими, так и абсолютно уникальными?

Ответ дать в краткой форме, например: минимально 10 и максимально 3000 строк
Ответ: минимально 50 и максимально 600.

Задание 3: Покупки

```
create table account
(
   id integer, -- ID счета
   client_id integer, -- ID клиента
   open_dt date, -- дата открытия счета
   close_dt date -- дата закрытия счета
)

create table transaction
(
   id integer,  -- ID транзакции
   account_id integer,  -- ID счета
   transaction_date date,  -- дата транзакции
   amount numeric(10,2), -- сумма транзакции
   type varchar(3) -- тип транзакции
)
```

Вывести ID клиентов, которые за последний месяц по всем своим счетам совершили покупок меньше, чем на 5000 рублей.

Без использования подзапросов и оконных функций.

```sql
SELECT

    a.client_id

FROM account a

JOIN transaction t ON a.id = t.account_id

WHERE

    t.transaction_date >= CURRENT_DATE - INTERVAL '1 month'

    AND t.type = 'buy' #предполагая что varchar(3) это ‘buy’

GROUP BY a.client_id

HAVING SUM(t.amount) < 5000;
```

Блок 4: Статистика и АБ-тесты

Задание 1: Воодушевленное руководство

Вы – аналитик компании Самокат (сервис по доставке продуктов на дом).

Команда решила протестировать гениальную идею замены транспортного средства доставщиков и провели АБ эксперимент в небольшом городке РФ. Результаты превзошли все ожидания: время доставки значимо снизилось в несколько раз! Руководству не терпится применить изменения по всей стране.

Делаем? Выберите все верные утверждения:

- ☐ Делаем. Только применяем изменения в таком же масштабе (количество транспортных средств с изменением), как в городе, где проводили тест.
- ☐ Тест нерепрезентативен, поэтому результаты применять по всей стране нельзя.
  Верно.
- ☐ Тест репрезентативен относительно своей генеральной совокупности: таких же небольших городов, можем применять только в подобных городах.

Верно, но с осторожностью. Но даже среди небольших городов есть различия (климат, дороги, плотность застройки). Лучше провести тесты еще в нескольких похожих городах.

- ☐ Наличие эффекта подтвердило потенциал идеи, поэтому сразу применять по всем городам не будем, но можем провести эксперимент в других городах.

  Верно, самый грамотный подход. Нужно провести тесты в других городах (разных типов: крупные, средние, с разной инфраструктурой).

Задание 2: Основной показатель в статистике

Что такое p-value в статистическом тестировании? Выберите одно верное утверждение:

- ☐ Вероятность, что нулевая гипотеза верна.
- ☐ Вероятность наблюдения такого или более экстремального результата при условии, что нулевая гипотеза верна.

  Верно.
- ☐ Значение уровня значимости, при котором отвергается нулевая гипотеза.
- ☐ Среднее значение выборки.

Задание 3: Параметрический тест

Вам необходимо провести аб-тестирование, целевая метрика - среднее число уникальных покупок на пользователя. Размер выборки велик (более 5 млн наблюдений в группах теста и контроля), значительные выбросы в выборках отсутствуют.

Можем ли мы применить t-критерий Стьюдента для проверки гипотезы о неравенстве средних в тестовой и контрольной группах при условии, что распределение уникальных покупок является логнормальным?

ДА, можем применить t-критерий Стьюдента.

Благодаря огромному размеру выборки (>5 млн) Центральная предельная теорема гарантирует нормальность распределения выборочного среднего, даже если исходные данные имеют логнормальное распределение. Отсутствие выбросов дополнительно повышает надежность критерия.

Блок 5: ML Base

Задание 1: Пони тоже кони

Вас просят разработать модель, классифицирующую лошадок и пони. Вместо разработки вы нашли на GitHub две интересные модели и после прогона на ваших данных одна из них показала ROC-AUC=0.7, а другая ROC-AUC=0.1. Какую модель вы возьмете для дальнейшей работы и что будете с ней делать?

Ответ написать развернутый, но кратко, например: "первую, отниму от метрики 0.1 для корректировки точности"

Мой ответ: Возьму вторую модель (ROC-AUC=0.1), инвертирую её предсказания (1 — p), получу ROC-AUC=0.9. Первую модель (0.7) можно использовать как baseline или для ансамбля.

Задание 2: Ручной счёт ROC_AUC

Классификатор выдал следующие прогнозируемые метки класса и вероятности принадлежности к классу "1". На основе полученных данных рассчитайте метрику ROC_AUC. Тезисно описать ход решения.

Ответ округлить до сотых, например: 4,12

Истинная метка класса	Порог классификации (0.6)	Оценка вероятности
1	1	0.95
0	1	0.9
1	1	0.85
0	1	0.8
1	1	0.75
1	1	0.7
1	1	0.65
1	1	0.6
0	0	0.55
0	0	0.5
0	0	0.45
1	0	0.4
0	0	0.35
0	0	0.3
0	0	0.25

Мой ответ:

1. **Разделил объекты на положительные (метка 1) и отрицательные (метка 0)**
2. **Для каждой пары (положительный, отрицательный) проверил, у кого выше предсказанная вероятность**
3. **Посчитал количество пар, где у положительного вероятность выше(49)**
4. **Разделил на общее число всех возможных пар(63)**
5. **Получил AUC = 0.78**

Задание 3: Ручной счёт корреляции

Рассчитайте линейную корреляцию Пирсона, на основе данных.

Какой вывод можно сделать на основе полученного результата? Можно ли утверждать, что существует причинно-следственная связь между количеством чашек кофе, выпитых студентами в течение экзаменационного дня, и их итоговым баллом за экзамен?

Ответ округлить до сотых, например: 4,12

Число выпитых чашек кофе	Балл за экзамен
1	85
1	88
2	79
2	81
2	84
2	65
3	67
3	58
3	76
4	49

Мой ответ:

Наблюдается сильная отрицательная корреляция (-0.85), но нельзя утверждать, что кофе вызывает низкие баллы — это может быть связано с влиянием третьих факторов (недосып, стресс) или обратной причинностью.

- **Обратная причинность: студенты, которые хуже знают материал, пьют больше кофе, чтобы взбодриться**

- **Третья переменная: например, недостаток сна (чем меньше спит, тем больше кофе и тем хуже сдает)**
- **Случайность: малая выборка (n=10) может давать случайные сильные корреляции**

Рекомендации по подготовке к собеседованию в случае успешного решения тестового задания

- Знать основы теории вероятностей;
- Знать основы математической статистики;
- Иметь практический навык реализации базовых запросов на SQL;
- Знать основы реализации алгоритмических задач на Python и оценки сложности алгоритмов;
- Приветствуется коммерческий опыт и знания в области планирования и проведения АБ-тестов.