Отчёт технического интервью

Кандидат: Candidate_1

Задач в сессии: 3 (Python)

1. История выполнения задач

Попытка #1

Задача: py-easy-4

Время выполнения: 33 ms

stdout: 'level' (pointers) -> True 'Level' (pointers) -> False

stderr:

Попытка #2

Задача: py-med-10

Время выполнения: 42 ms

stdout: Результат группировки по ключу 't': {1: [{'t': 1, 'value': 'a'}, {'t': 1, 'value': 'c'}], 2: [{'t': 2, 'value': 'b'}, {'t': 2, 'value': 'e'}], 3: [{'t': 3, 'value': 'd'}]}
Пример из задания: {1: [{'t': 1}, {'t': 1}], 2: [{'t': 2}]}

stderr:

Попытка #3

Задача: py-hard-2

Время выполнения: 41 ms

stdout: Путь между A и E (BFS): True Путь между A и G (BFS): False

stderr:

2. Решения кандидата по задачам

Задача py-easy-4 — Палиндром

Определите, является ли строка палиндромом.

```
def is_palindrome_pointers(s: str) -> bool:
    # Этот вариант не игнорирует пробелы и регистр.
    left_index = 0
    right_index = len(s) - 1
    while left_index < right_index:
        # Если символы не совпадают, это не палиндром
        if s[left_index] != s[right_index]:
            return False
        # Сдвигаем указатели к центру
        left_index += 1
        right_index -= 1
    return True
# Пример использования варианта с указателями (чувствителен к регистру и пробелам):
```

```
print(f"\n'level' (pointers) -> {is_palindrome_pointers('level')}") # True
print(f"'Level' (pointers) -> {is_palindrome_pointers('Level')}") # False (L != l)
```

Задача py-med-10 — Группировка по ключу

Сгруппируйте список словарей по значению ключа.

```
from collections import defaultdict
from typing import List, Dict, Any
def group_by(items: List[Dict[str, Any]], key: str) -> Dict[Any, List[Dict[str, Any]]]:
    """
    Группирует список словарей по значению указанного ключа.
    :param items: Список словарей.
    :param key: Ключ, по которому происходит группировка.
    :returns: Словарь сгруппированных элементов.
    """
    # defaultdict автоматически создает пустой список для нового ключа, если его нет
    grouped_data = defaultdict(list)
    for item in items:
        if key in item:
            group_value = item[key]
            grouped_data[group_value].append(item)
        # Опционально можно обрабатывать случаи, когда ключа нет в словаре
    # Преобразуем defaultdict обратно в обычный dict для стандартного вывода, если необходимо
    return dict(grouped_data)
# Пример использования:
data_list = [
    {'t': 1, 'value': 'a'},
    {'t': 2, 'value': 'b'},
    {'t': 1, 'value': 'c'},
    {'t': 3, 'value': 'd'},
    {'t': 2, 'value': 'e'}
]
grouped_result = group_by(data_list, 't')
print("Результат группировки по ключу 't':")
print(grouped_result)
# Вывод:
# {
#   1: [{'t': 1, 'value': 'a'}, {'t': 1, 'value': 'c'}],
#   2: [{'t': 2, 'value': 'b'}, {'t': 2, 'value': 'e'}],
#   3: [{'t': 3, 'value': 'd'}]
# }
# Пример из задания:
print("\nПример из задания:")
print(group_by([{'t':1}, {'t':2}, {'t':1}], 't'))
# Вывод: {1: [{'t': 1}], 2: [{'t': 2}]}
```

Задача py-hard-2 — Поиск пути в графе

Реализуйте DFS/BFS поиск пути между двумя вершинами.

```
from collections import deque
from typing import List, Dict, Any, Optional
def path_exists_bfs(graph: Dict[Any, List[Any]], start: Any, end: Any) -> bool:
    """
    Проверяет существование пути между start и end с использованием BFS.
    """
```

```
    if start not in graph or end not in graph:
        return False
    if start == end:
        return True
    # Очередь для хранения вершин, которые нужно посетить
    queue = deque([start])
    # Множество для отслеживания уже посещенных вершин
    visited = {start}
    while queue:
        current_node = queue.popleft()
        # Проходим по всем соседям текущей вершины
        for neighbor in graph[current_node]:
            if neighbor == end:
                return True
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
    # Если очередь опустела, а путь не найден
    return False
# Пример использования BFS:
graph_example = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
print(f"Путь между A и E (BFS): {path_exists_bfs(graph_example, 'A', 'E')}") # True
print(f"Путь между A и G (BFS): {path_exists_bfs(graph_example, 'A', 'G')}") # False
```

3. Анализ решений по задачам (LLM)

Задача py-easy-4 — оценка: 51.0/100

Комментарий: Итоговая оценка за задачу: 51.0/100 Оценка кода: 60/100 Код реализует базовую логику проверки палиндрома с использованием двух указателей, но не учитывает требования задачи: игнорировать пробелы и регистр символов. Это приводит к неправильной работе на тестах с пробелами или разным регистром. Оценка коммуникации: 30/100 Код не содержит проверок входных данных, что является серьезным недостатком. Отсутствие валидации может привести к ошибкам времени выполнения и уязвимостям. Также не указано, как именно должен обрабатываться случай, когда входные данные некорректны.

Замечаний не зафиксировано.

Задача py-med-10 — оценка: 86.0/100

Комментарий: Итоговая оценка за задачу: 86.0/100 Оценка кода: 95/100 Код корректно реализует группировку списка словарей по заданному ключу с использованием defaultdict. Логика работы понятна, код читаемый и эффективный. Единственное улучшение — можно добавить обработку случая, когда ключ отсутствует в словаре, чтобы избежать потенциальных ошибок при некорректных входных данных. Оценка коммуникации: 65/100 Кандидат предоставил корректное объяснение альтернативного подхода без использования defaultdict, что демонстрирует понимание работы со словарями. Однако код не представлен, и отсутствует практическая реализация. Также не указано, как именно будет работать алгоритм, что снижает

оценку.

Замечаний не зафиксировано.

Задача py-hard-2 — оценка: 72.0/100

Комментарий: Итоговая оценка за задачу: 72.0/100 Оценка кода: 90/100 Код корректно реализует алгоритм BFS для проверки существования пути между вершинами. Логика работы алгоритма верна, используются стандартные структуры данных (deque для очереди, set для посещенных вершин). Есть небольшая оптимизация: можно избежать проверки на равенство start == end, если добавить в очередь только соседей, но это не критично. Также стоит уточнить, что функция не обрабатывает случай, когда граф пустой или start и end не являются ключами в словаре graph, хотя в данном случае это не является ошибкой, так как проверка уже есть. Оценка коммуникации: 30/100 Кандидат не предоставил код для оценки, только теоретический анализ слабых мест алгоритмов BFS и DFS. Отсутствует работающий код, поэтому невозможно оценить корректность реализации. Также отсутствует конкретика в плане оптимизаций или решений проблем.

Замечаний не зафиксировано.

4. Итоговая оценка кандидата

Средняя оценка: 69.67/100. Сильные стороны: Хорошая оптимизация и эффективность решений, Хорошая читаемость и структура кода. Слабые стороны: Проблемы с корректностью/устойчивостью кода. Рекомендации: развивать алгоритмическое мышление, покрывать код тестами и уделять больше внимания структуре и читаемости решений.

5. Коммуникативные ответы кандидата

Задача: py-easy-4

Вопрос: Что является слабым местом твоего решения?

Ответ кандидата: Нет проверок на входные данные

Оценка коммуникации: 30/100

Код не содержит проверок входных данных, что является серьезным недостатком. Отсутствие валидации может привести к ошибкам времени выполнения и уязвимостям. Также не указано, как именно должен обрабатываться случай, когда входные данные некорректны.

Задача: py-med-10

Вопрос: Какие альтернативные способы решения возможны?

Ответ кандидата: Собственная реализация с использованием обычного словаря dict Вместо использования специализированного collections.defaultdict, можно написать ту же логику вручную, проверяя наличие ключа перед добавлением элемента. Этот способ более многословен, но не требует импорта дополнительных модулей.

Оценка коммуникации: 65/100

Кандидат предоставил корректное объяснение альтернативного подхода без использования defaultdict, что демонстрирует понимание работы со словарями. Однако код не представлен, и

отсутствует практическая реализация. Также не указано, как именно будет работать алгоритм, что снижает оценку.

Задача: py-hard-2

Вопрос: Что является слабым местом твоего решения?

Ответ кандидата: Общее слабое место (для BFS и DFS) Обработка графов с циклами (циклические графы): Самым критичным слабым местом является то, что без использования множества visited (посещенных вершин) алгоритмы DFS и BFS зациклились бы на графах, содержащих циклы. Графы в реальном мире почти всегда содержат циклы. К счастью, в моих реализациях я добавил множество visited, чтобы предотвратить бесконечный обход одних и тех же узлов. Слабые места конкретных реализаций 1. Итеративный BFS (с deque) Использование памяти при поиске: BFS хранит в очереди все узлы текущего "уровня" графа. Если граф очень широкий (много узлов на одном уровне), использование памяти может быть высоким. Гарантия кратчайшего пути: Хотя это обычно плюс, если вам нужен любой путь, а не кратчайший, BFS может быть менее эффективным, чем DFS, так как он исследует все пути равномерно. 2. Рекурсивный DFS Переполнение стека вызовов (Stack Overflow): DFS реализован с использованием рекурсии. В Python (и большинстве языков) глубина рекурсии ограничена. Если граф очень глубокий (длинный линейный путь) и этот путь не ведет к цели, функция может превысить максимальную глубину рекурсии и вызвать ошибку RecursionError. Не гарантирует кратчайший путь: DFS находит первый попавшийся путь, который может быть очень длинным и неоптимальным.

Оценка коммуникации: 30/100

Кандидат не предоставил код для оценки, только теоретический анализ слабых мест алгоритмов BFS и DFS. Отсутствует работающий код, поэтому невозможно оценить корректность реализации. Также отсутствует конкретика в плане оптимизаций или решений проблем.

6. Анализ античита

Общая статистика: вставок из буфера - 3, выходов из вкладки - 8, анализов кода - 14

Задача: py-easy-4

Вероятность списывания: 65% (уровень риска: medium)

Комментарий: Код содержит признаки списывания: одна вставка из буфера обмена, два выхода из вкладки, а также значительное увеличение размера кода на одном снимке (607 символов), что может указывать на вставку готового решения. Однако код написан самостоятельно, и нет явных признаков использования чужих решений из интернета.

Подозрительные события:

- clipboard_paste (medium): Одна вставка из буфера обмена

- tab_switch (low): Два выхода из вкладки

- code_growth (high): Резкое увеличение размера кода на одном снимке (+607 символов)

Задача: py-med-10

Вероятность списывания: 65% (уровень риска: medium)

Комментарий: Код был значительно увеличен (на 1353 символа), что указывает на возможное списывание или использование готового решения. Одна вставка из буфера обмена и три выхода из вкладки также повышают подозрительность. Однако структура кода соответствует ожидаемому решению задачи, что не является явным признаком читерства. Возможно, кандидат использовал шаблон из интернета или готовое решение.

Подозрительные события:

- code_growth (high): Резкое увеличение размера кода на 1353 символа
- clipboard_paste (medium): Одна вставка из буфера обмена
- tab_switch (medium): Три выхода из вкладки

Задача: py-hard-2

Вероятность списывания: 65% (уровень риска: medium)

Комментарий: Наблюдается умеренная вероятность списывания. Код значительно увеличился за один снимок (на 1241 символ), что может указывать на вставку готового решения. Также имеется одна вставка из буфера обмена и три выхода из вкладки, что повышает подозрительность. Однако структура кода соответствует стандартной реализации BFS, без явных признаков списывания из интернета.

Подозрительные события:

- code_growth (high): Резкое увеличение размера кода на 1241 символ за один снимок
- clipboard_paste (medium): Одна вставка из буфера обмена
- tab_switch (medium): Три выхода из вкладки