1 EXEMPLARY CRITIQUES

We provide 2 out of 3 exemplary critiques for few-shot prompting in Table 1, with one addressing the solution code and the other focusing on the problem code.

Table 1: Exemplary Critiques for Few-shot Prompting

Exemplary critique for solution code
Let's evaluate the code based on the provided criteria: **1. Is there any compilation error in the code?** No, the code does not have any compilation errors. **2. Is the code functionally correct?** From a preliminary review, it seems the code is functionally correct. It attempts to swap two characters in the string 's' such that the resulting string is lexicographically smaller than 'c'. **3. Is there an algorithm that is more efficient than the one used by the code?** The algorithm provided is relatively efficient for the problem constraints. The code essentially goes through the string 's' and checks if there's a lexicographically smaller character available later in the string. The double loop structure gives it an O(n^2) complexity, which is reasonable given the constraints. **4. Is the code too long or not concise enough?** The code is concise for the given problem. It sticks to the problem's requirements without unnecessary additions. **5. Can the code logic structure be simplified?** The structure seems clear, with the outer loop iterating over the test cases and the inner loops focusing on finding a potential swap. **6. What is the cyclomatic complexity of the code? Is it too high?** For solve(): V(G) = 1(for) + 1(for) + 1(if) + 1(elif) + 1(if) + 1(if) + 1 = 7, which is less than 10, so the cyclomatic complexity is small. **7. What is the cognitive complexity of the code? Is it too high?** Cognitive complexity deals with how hard the code is to read and understand. The code here is straightforward, and the variable names ('s', 'c', 't', 'n', etc.) are indicative of their use. However, for further clarity, better naming and some comments could be added. Still, it's not too high in its current form. **8. Are there any bad smells in the code? If so, please point them out.** A few potential areas of improvement: - The use of 'sys.stdin' and the custom reader might be considered unconventional when 'input()' would suffice for this problem's input constraints. - Variable naming could be improved for clarity. - The logic within the 'solve' function could benefit from some comments or intermediary variables with descriptive names. In conclusion, the code seems functional and reasonably efficient. With minor adjustments and clearer variable naming, it can be made more readable.
Exemplary critique for problem code
Let's analyze the provided code according to the specified criteria: 1. **Compilation error**: The code has an indentation error at the line 'graph[int(v) - 1][int(u) - 1] = c', which would prevent it from compiling. Additionally, there is an issue with the undefined variable 'char_idx' that would lead to a runtime error. 2. **Functional correctness**: Aside from the indentation error and the undefined 'char_idx' variable, the overall structure seems to aim at solving the given problem. However, the correctness of the algorithm in all cases is not guaranteed due to the 'char_idx' issue. Also, the logic in the nested for loop and the recursive function 'get_winner' might not handle all edge cases correctly, especially in handling the ASCII comparison of characters. 3. **Algorithm efficiency**: The algorithm uses recursion with memoization (caching results in 'winner_table'). This approach is generally efficient for this type of problem, though there may be more optimized algorithms in terms of space complexity. 4. **Code length and conciseness**: The code is relatively concise for the problem it's solving. However, the function 'mat' for creating a multi-dimensional array is a bit overcomplicated for this specific use case. Python's list comprehension could be used more effectively here. 5. **Code logic structure simplification**: The logic structure in the 'main' function, especially the last for-loop, seems unnecessary and unrelated to the problem. This part could be simplified or removed. 6. **Cyclomatic complexity**: For mat(): V(G) = 1(if) + 1(for) + 1 = 3; For main.get_winner(): V(G) = 1(if) + 1(for) + 2(if, and) + 1 = 5; For main(): V(G) = 1(for) + 1(for) + 1(for) + 1(for) + 1(for) + 1(for) + 3(if, and, or) + 2(elif, and) + 1 = 12; For __starting_point(): V(G) = 1. The cyclomatic complexity of main() is greater than 10, which is relatively large. 7. **Cognitive complexity**: The code is moderately complex to understand, primarily due to recursion and the use of a three-dimensional array for memoization. Better naming of variables and more comments could reduce cognitive complexity. 8. **Code smells**: The undefined 'char_idx' is a significant code smell. Also, the last for-loop in the 'main' function seems like dead or irrelevant code. Additionally, the use of a complex structure like a three-dimensional array for memoization could be considered over-engineering for this problem. **Recommendations for Improvement:** 1. **Fix Indentation and Undefined Variables**: Correct the indentation error and define or correctly use the 'char_idx' variable. 2. **Simplify Data Structures**: Consider simplifying the 'mat' function or using more straightforward Python list comprehensions for creating the multi-dimensional array. 3. **Remove Irrelevant Code**: The last for-loop in the 'main' function appears unnecessary for the problem and should be removed. 4. **Optimize Recursive Logic**: Review and possibly optimize the recursive function 'get_winner' to ensure it handles all edge cases correctly, particularly the ASCII character comparisons. 5. **Improve Readability**: Enhance readability by using more descriptive variable names and adding comments to explain complex parts, especially the recursive logic and memoization strategy.

2 OPTIMIZED CODES

Figure 1 shows the optimized code using the critiques generated by different models as feedback.

#Using feedback generated by CodeLlama:

```
def min_paint_needed(n, m, k, initial_colors, paint_amounts):
    min_paint = float('inf')
    # Iterate over all possible colorings
    for colors in range(1, m + 1):
        # Initialize colored trees
        colored_trees = initial_colors.copy()
        # Color uncolored trees
        for idx, c in enumerate(colored_trees):
            if c == 0:    colored_trees[idx] = colors
        # Calculate beauty of coloring
        beauty = 1
        for i in range(1, n):
            if colored_trees[i] != colored_trees[i - 1]:    beauty += 1
        # Check if beauty matches target
        if beauty == k:
            paint_needed = sum(paint_amounts[idx][color - 1] for idx, color in enumerate(colored_trees) if color != 0)
            min_paint = min(min_paint, paint_needed)
    # Return minimum paint needed or -1 if no valid coloring found
    return min_paint if min_paint != float('inf') else -1

def main():
    n, m, k = map(int, input().split())
    initial_colors = list(map(int, input().split()))
    paint_amounts = [list(map(int, input().split())) for _ in range(n)]
    result = min_paint_needed(n, m, k, initial_colors, paint_amounts)
    print(result)

if __name__ == "__main__":
    main()
```

#Using feedback generated by StarChat:

```
def min_paint_needed(n, m, k, initial_colors, paint_amounts):
    # Initialize the minimum paint needed as infinity
    min_paint = float('inf')
    # Iterate through all possible color combinations
    for color_combination in range(1, 2 ** m):
        # Initialize variables to track coloring and paint needed
        colored_trees = initial_colors.copy()
        paint_needed = 0
        # Iterate through each tree and color it if uncolored
        for i in range(n):
            if colored_trees[i] == 0:
                # Calculate the color for this tree based on the current combination
                color = (color_combination >> (i % m)) & 1
                # Update the color of the tree and add the paint needed
                colored_trees[i] = color + 1
                paint_needed += paint_amounts[i][color]
        # Check if the beauty of the coloring matches the target
        if len(set(colored_trees)) == k:
            # Update the minimum paint needed if this coloring is valid and requires less paint
            min_paint = min(min_paint, paint_needed)
    # Return the minimum paint needed or -1 if no valid coloring found
    return min_paint if min_paint != float('inf') else -1

def main():
    n, m, k = map(int, input().split())
    initial_colors = list(map(int, input().split()))
    paint_amounts = [list(map(int, input().split())) for _ in range(n)]
    result = min_paint_needed(n, m, k, initial_colors, paint_amounts)
    print(result)
if __name__ == "__main__":
    main()
```

#Using feedback generated by WizardCoder:

```
def minimum_paint(n, m, k, initial_colors, paint_amounts):
    # Count the number of groups needed to achieve beauty k
    groups_needed = len(set(initial_colors)) - (1 if 0 in initial_colors else 0)
    if groups_needed > k:
        return -1  # Impossible to achieve beauty k

    # Calculate the minimum paint needed
    min_paint = float('inf')
    for color in range(1, m + 1):
        paint_needed = sum(paint_amounts[i][color - 1] for i, c in enumerate(initial_colors) if c == 0 or c == color)
        min_paint = min(min_paint, paint_needed)

    return min_paint

def main():
    # Input
    n, m, k = map(int, input().split())
    initial_colors = list(map(int, input().split()))
    paint_amounts = [list(map(int, input().split())) for _ in range(n)]
    # Calculate and print result
    result = minimum_paint(n, m, k, initial_colors, paint_amounts)
    print(result)
```

#Using feedback generated by CoQuaLlama:

```
def min_paint_needed_dp(n, m, k, initial_colors, paint_amounts):
    # Initialize dp array with infinity
    dp = [[float('inf')] * (k + 1) for _ in range(n + 1)]
    dp[0][0] = 0

    # Iterate over all trees
    for i in range(1, n + 1):
        # Calculate number of contiguous groups for each color
        for color in range(1, m + 1):
            # Initialize group length and paint amount
            group_length = 0
            paint_needed = 0
            for j in range(i, 0, -1):
                # Update group length and paint amount
                if initial_colors[j - 1] == color:
                    group_length += 1
                else:
                    paint_needed += paint_amounts[j - 1][color - 1]
                # Update dp array based on group length and paint amount
                for b in range(1, min(k, i) + 1):
                    if dp[j - 1][b - 1] != float('inf'):
                        dp[i][b] = min(dp[i][b], dp[j - 1][b - 1] + paint_needed)
                    if group_length == b:
                        break
    # Find minimum paint needed for beauty k
    min_paint = min(dp[n][k], default=-1)

    return min_paint
```

Figure 1: The optimized code using feedback from different models.

3 PROMPTS

We provide the prompts for GPT-4 scoring and pairwise comparison, respectively, in Table 2 and 3.

Table 2: Prompt for GPT-4 Scoring

Role	Content
system	You are an expert critique scorer and will be provided with a code snippet and a critique for the code. The critique is based on the evaluation criteria. Your task is to score the critique based on the following scoring criteria consisting of 8 dimensions. Each dimension has a scoring range of 1 to 3 points. <Evaluation Criteria>{evaluation_criteria}</Evaluation Criteria> <Scoring Criteria>{scoring_criteria}</Scoring Criteria>
user	<Code>{code}</Code> <Critique>{critique}</Critique> The critique should explicitly itemize each aspect of analysis across 8 dimensions. If the critique lacks an analysis corresponding to a specific dimension, it receives a score of 0 points for that dimension. If you notice a lack of detailed analysis in a specific dimension, consider deducting one point accordingly. You need to assign scores to the critique for each of the 8 dimensions, totaling 8 scores. Please provide your scores for each dimension in the format: [0, 1, 2, 3, 3, 2, 1, 0].

Table 3: Prompt for Pairwise Comparison

Role	Content
system	You are an expert in comparing critiques and will be provided with a code snippet, a reference critique and two critiques for the code. Your task is to vote for the better critique from the two provided based on the reference critique.
user	<Code>{code}</Code> <Reference Critique>{reference_critique}</Reference Critique> <Critique A>{critique_A}</Critique A> <Critique B>{critique_B}</Critique B> Please cast your vote directly: A for Critique A, B for Critique B, or C for no significant difference between the two. Vote format: [A].