TASK 1

Analysis of Problems on the Sed Puzzle Website

Introduction

The Sed Puzzle is a game where we are given an initial string and must apply transitions to transform it, similar to the `sed` tool. In this game, the first match of a substring in the input string is replaced by the corresponding transition rule. The goal is to apply these transitions in such a way that the initial string eventually becomes empty.

Problem Levels and Characteristics

Level 0

- The problems contain only a few characters.
- The number of transitions is minimal.
- Either the source, target, or both are single characters.

Level 1

- The problems still have a small number of characters.
- The transitions are limited, but the number of steps required is slightly more than in Level 0.
- The source or target (or both) consist of 2 to 4 characters.

Level 2

- Some problems contain a very large number of transitions.
- Others have a few transitions but require many steps to reach the solution.
- Most problems involve 4 to 6 character-long strings.
- Some problems begin with large initial strings, while others start with small ones.
- The final transition always results in an empty string (`""`).

Level 3

- Most problems require a large number of steps, ending in an empty string (`""`).
- Some problems have a small character set and a small number of transitions.
- Others have a large character set with a large number of transitions.
- Some problems feature a small character set with a large number of transitions, while others have a large character set with a large number of transitions.
- Two specific problems (9 and 10) have a small character set and a small number of transitions.

- Some problems use only two characters.
- A few problems involve multiple string-to-empty transitions.

Level 4

- Some problems contain 10 or more characters with a large number of transitions.
- Others have only a few characters and a few transitions.
- Some problems consist of just two characters but require a very large number of transitions.
- Certain problems feature 2 or 3 characters with only 2 or 3 transitions.
- A unique case involves transitions that move from an empty string to a non-empty state, creating a distinct challenge.

Category: Impossible

(This category is custom-created and does not exist on the official Sed Puzzle website.)

Characteristics:

- **Huge character set**: Uses a wide range of characters (e.g., `a-z`, `A-Z`, `0-9`, special symbols).
- **Large number of transitions**: Includes hundreds or thousands of possible transformations.
- **High number of unique transitions**: Ensures that almost all transitions are distinct, preventing repetition.
- **Empty-to-Non-Empty Allowed**: Strings are allowed to transition from an empty state (`""`) to a non-empty state, making cycles possible.
- **Frequent transitions to empty**: A high percentage of transitions result in an empty string, leading to unpredictable behavior.
- **Wide source and target range**: The length of `src` and `tgt` can vary significantly, from very short (1 character) to very long (full-string replacement).
- **Extreme difficulty**: Due to the vast number of possible states, finding a solution is nearly infeasible within a reasonable number of steps.

Conclusion

The Sed Puzzle problems exhibit increasing complexity across levels. While Level 0 focuses on minimal character sets and transitions, higher levels introduce greater complexity in terms of the number of steps, transition count, and string length. Level 4 introduces unique challenges, such as transformations that lead from an empty string to a non-empty one. Understanding these characteristics is crucial for developing efficient strategies to solve the puzzles optimally.

The code is a generator for **Sed Puzzle** test cases. It creates a set of string transformation rules (**transitions**) and an initial string, then applies the transitions to solve the puzzle while ensuring constraints such as transition uniqueness and limits on transition usage.

How the Code Works

Step 1: Generate an Initial String

- The function `generate_random_string(int length)` creates a random string using the characters `"abcdef"`.
- Initially, the string is empty (`""`).
- The first randomly generated string is stored as the first transition (`src -> ""`) in the transition map.

Step 2: Generate Transitions

The function `generate_transitions(...)` is responsible for generating source-to-target transitions while ensuring:

- **Uniqueness**: No duplicate transitions are created.
- **Usage Limit**: Each transition is used at most a set number of times (`max_usage_per_transition`).
- **Empty Transitions Control**: Limits how many times a transition can turn a string into an empty string.

To manage transitions, two maps are used:

- `transition_map`: Stores already created transitions along with their indices.
- `transition_usage`: Tracks how many times each transition has been used.

Transition Creation Process:

1. A random substring of the current string is selected as the **source (`src`)**.
2. Another random string is generated as the **target (`tgt`)**.
3. The transition (`src -> tgt`) is applied to the first occurrence of `src` in the string.
4. The transition is stored in the transition map, and its count is tracked in `transition_usage`.

Step 3: Validate Transitions

To mimic the behavior of the **Sed Puzzle website**, each transition is tested by:

- **Applying the transition forward** (replace `src` with `tgt`).
- **Applying it in reverse** (replace `tgt` back with `src`).

If the resulting string is the same as the original after both operations, the transition is valid. Otherwise, the transition is skipped.

Step 4: Apply Transitions to Transform the String

- Transitions are applied iteratively to ensure the final string becomes empty (`""`).
- The order in which transitions are applied is stored in the `solution` vector.

Step 5: Write to JSON Files

Two JSON files are created for each test case:

1. **Puzzle File (`puzzles/problem_id.json`)**
   - Stores the **initial string** and the list of **transition rules**.
2. **Solution File (`solutions/problem_id.json`)**
   - Stores the **sequence of transition indices** that lead to an empty string.

Problems Encountered While Making the Generator

1. Incorrect Backtracking Due to `tgt` Position

- If, while backtracking, the **`tgt` string is found before `src`** in the current string, the transition would reverse incorrectly.
- This caused wrong transitions when applied in reverse order.
- **Fix**: Ensure that `src` occurs **before** `tgt` when reversing transitions.

2. String Corruption in Edge Cases

Example: If the string is `"aba"` and the transition `b -> a` is applied:

- `"aba"` → `"aaa"` (after applying `b -> a`).
- If the reverse transition (`a -> b`) is applied:
  - It could incorrectly generate `"baa"`, instead of `"aba"`.
- **Fix**: Each transition is **verified** by applying it **forward and then backward** to ensure it produces the **original string**.

This approach ensures the generated transitions closely mimic the **Sed Puzzle website's** mechanics while avoiding incorrect or non-reversible transitions.

Fields You Can Edit and Their Meaning

Field Name	Description	Default Value
`min_src_length`	Minimum length of source substring in transitions	`1`
`max_src_length`	Maximum length of source substring	`3`
`min_tgt_length`	Minimum length of target substring	`2`
`max_tgt_length`	Maximum length of target substring	`4`
`num_transitions`	Total number of transformation steps	`18`
`num_unique_transitions`	Number of unique transition rules allowed	`11`
`max_usage_per_transition`	Maximum times a transition can be used	`2`
`max_empty_transitions`	Maximum transitions that result in an empty string	`1`
`empty_to_non_empty`	Whether an empty string can transition to a non-empty string	`false`

Output Format

1. Problem JSON (Puzzle Definition)

Saved as:
📁 `../sample-data/puzzles/100.json`

```
{
  "problem_id": "100",
  "initial_string": "abc",
  "transitions": [
    {"src": "ab", "tgt": "c"},
    {"src": "c", "tgt": ""}
  ]
```

}

2. Solution JSON

Saved as:
📁 `../sample-data/solutions/100.json`

```
{
   "problem_id": "100",
   "solution": [1, 0]  // Transition sequence to solve the puzzle
}
```

How to Modify the Behavior

- **Increase Difficulty** → Raise `num_transitions` or `num_unique_transitions`.
- **Control String Complexity** → Adjust `min_src_length`, `max_src_length`, `min_tgt_length`, `max_tgt_length`.
- **Allow More Empty String Transitions** → Increase `max_empty_transitions`.
- **Enable Non-Empty to Empty Transitions** → Set `empty_to_non_empty = true`.