Spring 2025, Digital Computer Concepts and Practice
Term Project Specification

Language and Data Intelligence Lab., SNU

Task

- Implement an efficient Wordle puzzle solver that minimizes the number of queries needed to find the secret word.
- Implement a simple HTTP server that returns the next guess upon request.
- Create a problem instance (secret word + candidate word list).

Matching Rules

Given an answer (secret word) and a query, the result is computed as follows:

1. If the two words have the same letter at the i-th ($1 \leq i \leq 5$) position, the corresponding result is `"B"`.
2. If the query contains letters not present in the answer, they are marked as `"G"`.
3. If the query contains matching but misplaced letters, they are marked as either `"Y"` or `"G"`, such that:
   - The total number of `B`'s and `Y`'s assigned to a character cannot exceed its count in the answer.
   - All `G`'s, if any, must appear after all `Y`'s for that character.

For example:

```
ANSWER TWEET
 QUERY MELEE
------------
RESULT GYGBG
```

Note how the three `E`'s in the query are handled. The second `"E"` is marked as `"B"` since it matches the correct position. The other two `"E"`'s are assigned `"Y"` and `"G"` from left to right, reflecting that only two `"E"`'s exist in the answer.

Your solver will not see these B/Y/G codes directly. Instead, you will receive a natural language description of the feedback, which you must parse.

An example:

```
'M' and 'L' are not in the word. The second 'E' is in the correct
position; there is only one more misplaced 'E' in the word.
```

Problem Setup

Each problem has a unique problem ID. For each problem in the test suite, the grader will first provide the list of candidate words associated with that problem. All words (both secret word and candidates) will be lowercase English words of length 5.

The interaction between the grader and your server follows a simple HTTP-based protocol. Your server must handle two kinds of requests.

1. Start Problem Request

At the start of each problem, the grader will send:

```
POST /start_problem
Content-Type: application/json
{
  "problem_id": "<problem_id>",
  "candidate_words": ["word1", "word2", ..., "wordN"]
}
```

Your server must store both `problem_id` and the provided candidate words. Your guesses must only select words from this candidate list.

2. Next Guess Request

Then, the grader will repeatedly ask for your next guess, which includes the feedback for the previous guess:

```
POST /guess
Content-Type: application/json
{
  "problem_id": "<problem_id>",
  "verbal_feedback": "<feedback>",
  "turn": <turn_number>
}
```

- `verbal_feedback` contains natural language feedback describing the result of your previous guess.
- On the first turn, `verbal_feedback` will be `null` since no guesses have been made.
- `turn` is the current turn number (starting from 1).

Only the most recent feedback will be provided. Your server must maintain complete feedback history internally for each problem.

3. Guess Response

For each `/guess` request, your server must respond:

```
HTTP 200 OK
Content-Type: application/json
{
  "guess": "<your_guess>"
}
```

- Your guess must always be one of the candidate words provided.
- An invalid guess will result in immediate failure for that problem.

Additional Notes

- Only one problem will be active at a time; concurrent handling of multiple problems is not required.
- You are responsible for all internal state management.
- All necessary information is provided through these HTTP requests; no external files or prior knowledge may be used.
- The verbal feedback is generated according to the matching rules above.
- You are free to use LLMs (e.g., ChatGPT, Copilot, etc.) as part of your development process. If you do so, briefly mention how you utilized them (e.g., for code generation, debugging, or prompt engineering) as part of your presentation.
- Your server must read the port number from the environment variable `PORT`, defaulting to 8000 if not set. During grading, your server may be assigned a different port.

Starter Code

Starter code is provided on eTL:

- `grader.py`: Simplified grader for your own testing. Note: the provided version uses simple verbalization rules that may not handle repeated characters precisely. For evaluation, the grader may use LLMs to generate accurate feedback.
- `team00.py`: Starter code template for your solver and HTTP server. You must implement the core solver logic. Modify the team number to match yours.

You may restructure your code as needed, but your server must be runnable using:

```
python team00.py
```

Budget

- Your server must be fully ready to receive requests within 10 seconds after startup.
- You have 60 seconds to complete solving each problem.
- You must provide your own Snowflake credentials (via environment variables or config files). Ensure sufficient credits are available for evaluation.

Writing a Problem

Create one Wordle problem instance consisting of a candidate word list and a secret word. The candidate words must be selected from the provided global list (of valid 5-character English word, lowercased). Your solver will be tested on your own problem instance as well, so ensure it can solve it efficiently.

Submission

Submit the following on eTL:

- Presentation slides (replacing a separate written report).
- Full code (all files necessary to run your server).
- Your problem instance as `team00_problem.json`, containing a dictionary with `secret_word` and `candidate_words` keys.

Evaluation

Evaluation will consider:

- The number of guesses used per problem. Failures (timeouts or errors) will be counted as 100 guesses.
- If solved within 60 seconds but with more than 100 guesses, the count will be capped at 100.
- Presentation quality.

Presentation

Each team will present for 5 minutes (mandatory). Explain your method, design decisions, challenges, and how you addressed them.

Schedule

- Code and slides submission: Due 6/18 11:59 PM
- Presentations: 6/16, 6/18 (remote)

Final Note (Dependencies)

The main focus of this project is on problem-solving logic. In this sense, you are kindly asked to avoid using additional third-party libraries beyond `snowflake-ml-python` and `dotenv`.

If you believe that a particular package is essential for your solution, you must provide the TAs with the necessary means to reproduce your environment, such as a complete `requirements.txt` file. However, please note that any such submission will be handled on a *best-effort* basis — we cannot guarantee that we will be able to reproduce or resolve all external dependencies during grading.