Coding_Benchmark

Leo Deng

4 sections

1. Error/Edge Cases analysis
2. Task Planning
3. Optimization
4. General Testing

3 levels of difficuties

1. Easy-Beginner
2. Medium-Intermediate
3. Hard-Expert

Error/Edge cases handling

- 30 questions
- 10 in each difficulty level
- Aim to test if the AI can handle all of the error inputs as well as edge cases

sample question

- Write a function that takes a user's input for their age and returns a message indicating if they are eligible to vote

Task Planning

- 30 questions
- 10 in each difficulty level
- Aim to test if the AI can plan a task efficiently

sample question

- Write a function that takes a list of orders with delivery times and returns a schedule that minimizes the total delivery time.

Optimization

- 10 questions
- 3 easy, 3 medium, 4 hard
- Given a correct but not optimal solution, can AI successfully optimize it?

sample question

- Write a Python function that takes a list of numbers and returns their sum.
- Solution:
- def sum_of_list(numbers):
- return sum(numbers)
- Optimization Question: Can you optimize the function to work efficiently with a list of a million numbers? Assessment: The grade will depend on the efficiency of the solution. A solution using a loop instead of the built-in sum function would be graded lower because it would be less efficient.

General Testing

- 30 questions
- 10 in each difficulty level
- Aim to test the general coding ability of AI

sample question

- Implement a function to solve the N-Queens problem

Assessment Standard

- The assessment standard vary between sections. E.g.
- For Error Handling, part of the points focus on if the AI successfully handles all the errors and edge cases
- For Task Planning, the time it takes to solve the problem and the algorithm effiency takes a large part of the points
- For optimization, ‘Is the solution optimized for speed and memory usage’ is what we are assessing

Extract problems and standards

```python
import openai
from docx import Document

# Initialize the OpenAI API (in a real-world application)
openai.api_key = 'YOUR_OPENAI_API_KEY'

def extract_problems_and_standards(file_path):
    """Extract problems and assessment standards from the DOCX file."""

    doc = Document(file_path)
    text = [p.text for p in doc.paragraphs if p.text.strip() != ""]

    sections = ["Error/edge case handling", "Task planning", "Optimization", "General testing"]
    problems = {}
    standards = {}
    current_section = None

    for line in text:
        if line in sections:
            current_section = line
            problems[current_section] = []
        elif "assessment standard" in line.lower():
            standards[current_section] = problems[current_section]
            problems[current_section] = []
            current_section = None
        elif current_section:
            problems[current_section].append(line)

    return problems, standards
```

RESULT

```
{'Error/edge case handling': ['Easy-Beginner',
  "Problem 1: Write a function that takes a user's input for their age an
  'Problem 2: Write a function that takes a filename and returns the numb
  'Problem 3: Write a function that takes a URL and returns the content o
  'Problem 4: Write a function that takes a list of prices (as strings) a
  'Problem 5: Write a function that takes a dictionary of usernames and p
  'Problem 6: Write a function that takes a dictionary of usernames and a
  'Problem 7: Write a function that takes a filename and a list of string
  'Problem 8: Write a function that takes a list of integers and returns
  'Problem 9: Write a function that takes a string and returns the number
  'Problem 10: Write a function that takes a list of integers and returns
```

Send problems to GPT/other AI

```
def send_problem_to_chatgpt(prompt):
    """Send the prompt to ChatGPT and get the response."""

    response = openai.Completion.create(
      engine="davinci-codex",
      prompt=prompt,
      max_tokens=500
    )

    return response.choices[0].text.strip()
```

Do the problems and assess it

```
# Main logic
file_path = "/path/to/your/docx/file.docx"
problems, standards = extract_problems_and_standards(file_path)

for section, section_problems in problems.items():
    for problem in section_problems:
        solution = send_problem_to_chatgpt(problem)  # This would interact with ChatGPT in a real-world application

        # Assessment
        standard = standards[section]
        assessment_prompt = f"Here is the solution for the problem:\n{solution}\n\nPlease assess the solution based on the following criteria:\n{standard}"
        feedback = send_problem_to_chatgpt(assessment_prompt)

        print(f"Feedback for {problem}: {feedback}")
```

Sample assessments

1. Eligibility to Vote Function:

- **Correctness (70 points)**: The function seems to correctly handle tl cases. It checks for valid integer input and determines voting eligil
  - Score: 70/70
- **Efficiency (15 points)**: The function's operations are simple checks to a time complexity of $O(1)$.
  - Score: 15/15
- **Readability (15 points)**: The funct comments explaining its behavio
  - Score: 10/15
- **Total**: 95/100

2. Count Lines in File

- **Correctness (70 points)**: T case where the file doesn't
  - Score: 70/70
- **Efficiency (15 points)**: The function reads the entire file to count lines, which could be improved for very large files. Its time complexity is $O(n)$, where n is the number of lines.
  - Score: 12/15
- **Readability (15 points)**: The function is straightforward but lacks comments.
  - Score: 10/15
- **Total**: 92/100

3. Fetch URL Content Function:

- **Correctness (70 points)**: As a mock function, it doesn't fetch real content, but in a real-world scenario, it would need to handle various edge cases.
  - Score: 40/70 (due to being a mock function)
- **Efficiency (15 points)**: The mock function doesn't perform real operations, so we can't evaluate its efficiency.
  - Score: 0/15
- **Readability (15 points)**: The function clearly states its mock nature.
  - Score: 15/15
- **Total**: 55/100