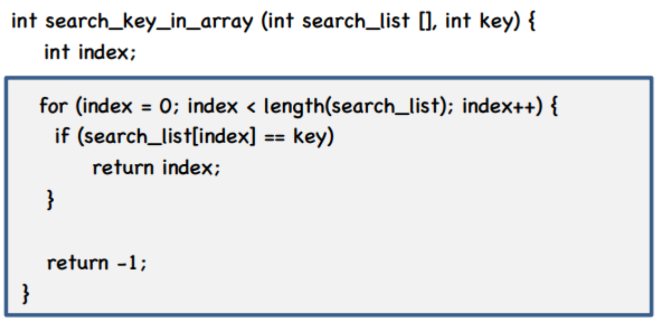
SUTD SHARP Term 3 Honours sessions AI Safety and Bias Testing Homework

Goh Jet Wei 1007655

Q1. Large language model (LLM) based code completion systems aim to provide an “AI pair programmer”. Concretely, the programmer may provide a code prompt as an input and the pair programmer completes the code. Following is an example of such code completion system. The code outside the box is the code prompt provided by the human programmer and the code inside the box is generated by the AI pair programmer. As shown, the AI pair programmer correctly synthesizes code that searches the key inside the list array.



```
int search_key_in_array (int search_list [], int key) {
    int index;

    for (index = 0; index < length(search_list); index++) {
      if (search_list[index] == key)
           return index;
    }

    return -1;
}
```

a. *We intend to automatically test/validate the reliability of the AI code completion system. Design a metamorphic test generation framework to test the functional correctness of the code completion system. Concretely, design five metamorphic transformations and their corresponding metamorphic relations. Show your metamorphic transformations with clear examples. For the metamorphic relations, clearly describe how you compare the outputs of the original and the metamorphically transformed inputs.*
(Note: Each of your metamorphic transformations should be conceptually different. Additionally, your design should be general, not specifically tuned for the example shown above.)

- **Transformation 1: Reversing the Input**
  Reverse the order of the input list.

  Example 1:
  Original input: [1, 2, 3, 4, 5] with key 3

Transformed input: [5, 4, 3, 2, 1] with key 3
Explanation: In the original input, if the key was 3, it is at position 2 (index 2). After reversing the list, the key 3 should now be at position 2 (because len(list) - 1 - 2 = 2).

Example 2:
Original input: [10, 20, 30, 40, 50] with key 20
Transformed input: [50, 40, 30, 20, 10] with key 20
Explanation: The key 20 is found at position 1 in the original list, and after reversing, it should still be found at position 3 (because len(list) - 1 - 1 = 3).

Relation: The position of the key in the reversed list should match len(list) - 1 - original position.

- **Transformation 2: Adding Irrelevant Elements**
  Add elements to the input list that are not equal to the key.

  Example 1:
  Original input: [1, 2, 3, 4, 5] with key 3
  Transformed input: [1, 2, 3, 4, 5, 99, 100]
  Explanation: In the original input, key 3 is at position 2. After adding irrelevant elements (99 and 100), the key 3 should still be at position 2 in the transformed list.

  Example 2:
  Original input: [3, 6, 9, 12] with key 6
  Transformed input: [3, 6, 9, 12, 99, 100, 101]
  Explanation: In the original input, key 12 is at position 4. After adding irrelevant elements, it should still be at index 4.

  Relation: The position of the key in the original list should remain unchanged in the transformed list.

- **Transformation 3: Duplicate an Existing Element**
  Duplicate one or more elements in the list.

  Example 1:

Original input: [1, 2, 3, 4, 5] with key 3
Transformed input: [1, 2, 3, 3, 4, 5] with key 3
Explanation: If the key 3 is duplicated, the AI should still return the position of the first occurrence of the key 3 at position 3.

Example 2:
Original input: [5, 10, 15, 20, 25] with key 5
Transformed input: [5, 10, 15, 5, 20, 25] with key 5
Explanation: If the key 5 is duplicated, the AI should still return the first occurrence of the key 5 at position 1.

Relation: If the key is duplicated, the AI should still return the index of the first occurrence of the key.

- **Transformation 4: Changing the Key Value**
  Use a different key value that does not exist in the list.

  Example 1:
  Original input: [1, 2, 3, 4, 5] with key 3
  Transformed input: [1, 2, 3, 4, 5] with key 10
  Explanation: The AI should return an indication that the key 10 is not found (e.g., -1 or None).

  Example 2:
  Original input: [2, 4, 6, 8, 10] with key 10
  Transformed input: [2, 4, 6, 8, 10] with key 12
  Explanation: The AI should return -1 (or None) because the key 12 does not exist in the list.

  Relation: The system should correctly return an indication that the key is not found (e.g., -1 or None).

- **Transformation 5: Scaling the Input**
  Multiply all elements in the list by a constant factor (e.g., 2).

  Example 1:
  Original input: [1, 2, 3, 4, 5] with key 4
  Transformed input: [2, 4, 6, 8, 10] with key 8

Explanation: The AI should find the key 8 at position 8, which corresponds to the original position of key 4 (position 4).

Example 2:
Original input: [1, 4, 7, 10] with key 7
Transformed input: [2, 8, 14, 20] with key 14
Explanation: The AI should find the key 14 at position 2, which corresponds to the original position of key 7 (position 3).

Relation: The AI should still find the scaled key at the corresponding position.

b. *Assume that you have a set of seed inputs to test the AI code completion system. Based on this assumption, write pseudocode for a randomized algorithm to metamorphically test the functional correctness of the code completion system. Discuss what seed inputs are needed and your pseudocode should be capable to generate unbounded number of test inputs despite starting with a small finite set of seed inputs.*

```
Algorithm: Randomized Metamorphic Testing
Input: Seed inputs (list of test cases), AI code completion system
Output: Number of successful and failed tests

1. Initialize success_count = 0 and failure_count = 0
2. For each seed input in the seed_inputs:
    a. Generate multiple transformed inputs by applying random metamorphic
transformations.
       Example: Reverse the input, scale the input, etc.
    b. For each transformed input:
        i. Run the AI code completion system on both the original and
transformed inputs.
        ii. Compare the outputs using the corresponding metamorphic relation:
            If outputs satisfy the relation:
                Increment success_count
            Else:
                Increment failure_count and log the failure.
3. Return success_count and failure_count
```

Example seed Inputs:

- Lists of integers: [1, 2, 3], [4, 5, 6, 7]
- Keys to search: 1, 5, 10

c. *Assume that the AI code completion system mostly satisfies robustness i.e., small changes to its input do not affect its output. Exploit this to make your metamorphic testing more effective. Modify the pseudocode of the previous question in this respect. Your effective metamorphic testing should be targeted to find more bugs (as compared to the vanilla randomized algorithm designed for the previous question) within a given time.*

   Example of small changes:
   - Minor noise to numerical values (e.g., [1.0, 2.0, 3.0] → [1.001, 2.0001, 3.0]).
   - Shuffling of elements slightly if order doesn't matter

   To exploit this mostly-robustness assumption, we can modify the pseudocode with the following ideas:

   - Combining Multiple Transformations: Instead of applying a single transformation at a time, combine two or more transformations to create more complex scenarios.
     - Example:
       - Original Input: [1, 2, 3, 4, 5] with key 2
       - Modification of pseudo code of Combining Multiple Transformations:
         - Step 1: Apply Reversing the Input → [5, 4, 3, 2, 1]
         - Step 2: Apply Adding Irrelevant Elements → [5, 4, 3, 2, 1, 99, 100]
       - Expected Output: The key 2 should be found at position 4 in the reversed list.
     - Rationale: By combining transformations, we can test the AI's ability to handle multiple changes simultaneously, exposing more subtle bugs that may not appear with isolated transformations.

- Randomized Repetition of Transformations: Randomly apply the same transformation multiple times to create extreme test cases.
    - Example:
        - Original Input: [1, 2, 3, 4, 5] with key 3
        - Modification of pseudo code of Randomized Repetition of Transformations:
            - Step 1: Apply Duplicate an Existing Element five times → [1, 2, 3, 3, 3, 3, 3, 4, 5]
        - Expected Output: The AI should still correctly identify the first occurrence of the key 3 at position 2.
    - Rationale: Repeating a transformation tests the AI's limits. For instance, if duplication causes unexpected behavior or excessive processing, this will reveal such vulnerabilities.

- Prioritization Based on Edge Cases: Focus the metamorphic testing framework on boundary values or edge cases, which are known to reveal bugs.
    - Example (using Empty Lists: Test how the AI handles an empty list)
        - Original Input: [] with key 3
        - Expected Output: Return -1 or None.
    - Example (using Large Lists: Test the AI with very large inputs to check scalability)
        - Original Input: A list of 1,000,000 integers [1, 2, ..., 1,000,000] with key 999,999
        - Transformed Input: Reverse the list or duplicate elements.
        - Expected Output: Key should still be found at the correct index.
    - Example (using Near Duplicates: Create lists where the key appears multiple times but in slightly different positions)
        - Original Input: [3, 3, 3, 3, 3] with key 3
        - Transformed Input: Add irrelevant elements ([3, 3, 99, 3, 3, 3, 3]).
        - Expected Output: AI should find the first occurrence.
    - Rationale: Edge cases often reveal hidden weaknesses or assumptions in the AI model, such as how it handles extreme inputs or uncommon scenarios.

```

Algorithm: Smarter Random Metamorphic Testing with Robustness
Input: Seed inputs (list of test cases), AI code completion system
Output: Number of successful and failed tests
```

```
1. Initialize `success_count` = 0 and `failure_count` = 0.
2. Create a combined input set: `test_inputs` = seed_inputs + edge_cases.
3. For each input in `test_inputs`:
    a. Generate a queue of test cases:
        i. Randomly combine multiple transformations (2-3 transformations) on the input.
        ii. Repeat transformations multiple times (e.g., apply the same transformation 2-5 times).
        iii. Include small perturbations to introduce minor changes (e.g., add noise or shuffle elements).
        iv. Append edge cases test cases such as empty inputs, very large inputs, inputs with all duplicates, etc.
    b. For each transformed input in the queue:
        i. Run the `AI_system` on both the original input and the transformed input.
        ii. Compare the outputs using the corresponding metamorphic relation:
            If outputs satisfy the relation:
                Increment success_count
            Else:
                Increment failure_count and log the failure.
4. Return `success_count` and `failure_count`.
```