HW2

Aung Kyaw Tun

April 2024

1 Problem 2.1

a) For functional coverage, at least one test is necessary to ensure that the function is called and it returns a value. As such a single test asserting the correct output of the function with appropriate inputs as below is sufficient.

```rust
#[test]
fn lved_func_coverage_tests() {
    let distance = lved("hello", "world");
    assert_eq!(distance, 4);
}
```

b) To achieve statement coverage, no additional test is necessary, as the test case above covers all the statements within the function.

c) To achieve branch coverage, every branch that occurs within the function need to be executed. In this case, a branch occurs where there is an "if else". That can be achieved with the following test cases which include a test case where characters are equal and a case where characters are not equal.

```rust
#[test]
fn lved_branch_coverage_tests() {
    assert_eq!(lved("hello", "hello"), 0);
    assert_eq!(lved("hello", "world"), 4);

}
```

d) For condition coverage, each boolean sub-expression within the conditions of branch statements evaluates to both true and false. In the `lved` function, the condition `s1[i-1] == s2[j-1]` is the primary condition. To achieve condition coverage, we need at least two tests where one tests where the condition is true and another that tests where the condition is false.

```rust
#[test]
fn lved_condition_coverage_tests() {
    assert_eq!(lved("hello", "hello"), 0);
```

```
    assert_eq!(lved("hello", "world"), 5);
}
```

e) To achieve boundary interior path coverage, each possible path from the start to end must be executed. Therefore multiple test cases are required such as with empty string, strings of various lengths, strings of similar characters but in different positions and other test cases with different scenarios. It will require multiple test cases but the example below covers some aspects of it.

```
#[test]
fn lved_boundary_interior_path_coverage_tests() {
    // Testing empty strings
    assert_eq!(lved("", ""), 0);
    assert_eq!(lved("", "world"), 5);
    assert_eq!(lved("hello", ""), 5);

    // Testing strings of different lengths
    assert_eq!(lved("hello", "hell"), 1);
    assert_eq!(lved("hello", "helloworld"), 5);

    // Testing strings with similar characters but different positions
    assert_eq!(lved("hello", "ehllo"), 2);
    assert_eq!(lved("hello", "hlelo"), 2);
}
```

2 Problem 2.2

a) While fuzzing the program given, there is a potential issue with memcmp() that could lead to a crash or unexpected behavior. The issue is that the function doesn't check the size of the memory blocks ‘s1‘ and ‘s2‘. If ‘n‘ is larger than the size of either of the memory block, the function will read beyond allocated memory which can lead to undefined behavior in the program.