Assignment 5: Black Box Testing: Lyell Read

Summary

I approached this with a programmatical check method. I chose to automate tests, so they are scalable, and apply ECP to the largest number of boundary cases possible. To that end, I wrote the python program located in this directory to perform these tests. A more manual approach is certainly valid, too, and is explained below.

Manual Case

In case the source code is not easily reproduced exactly in terms of functionality in the scripting language, then it becomes necessary to perform checks by hand. This will end up in the user creating a dictionary of input and corresponding correct output cases in the correct() function.

Cases Covered

This pseudocode test script covers a wide range of cases, and covers them randomly (within bounds (10% around edges)) to assure the best coverage. This allows this test to be repeated without editing the test to get more and more accurate results, time permitting. This test covers the following cases (semi-separately, to reduce time and exponential time scaling that would result if these were combined -- this would approach full code coverage or brute force):

- All boundaries where integer length increases or decreases (boundary example: 99, 100):
    - (len(INT_MAX)*-1, -1], (1, len(INT_MAX))
    - Note: -1 here represents the negative numbers with length 1
    - Tests check both top 10% and bottom 10% of each range respectively.
    - ‘X’ denotes checked, ‘[]’ denote equivalence class:
        - [0X--------X][10X--------X][100X...]
- First 20%, middle 20% and last 20% placement:
    - Hardcoded tests provide a matrix with the first nonzero placed at the boundary of the start of the row and at the end, and at the center, each an equivalence class that gets tested for the matrix of that size
        - Test runs for positives and negatives at that place.
        - X represents possible value placement for each start, middle, end class:
        - [X, X, 0, 0, X, X, 0, 0, X, X] (for example)
- All valid array sizes (passed as an argument) have their equivalence classes checked both low and high. This works the same as integer length testing, except in this case, the integer is the height and width of the 2D matrix
    - Tested on both sides of each boundary where integers increase or decrease, including all these lengths:
    - [1, len(ARRAY_MAX))
    - Tests check both top 10% and bottom 10% of each class

Discussion

This test is essentially an improvement on black box testing - it combines smart exhaustive and black box. Therefore, it more than completes the assignment goals. Given the current length of C's INT_MAX == 10, this gives the following equivalence classes (red) and boundaries (blue) (inclusive), for example *just for the integer length tests (1/3).

Integer Length

Negative Length 10
All lengths 10 > length >= 9, negative (top and bottom checked)
Negative Length 9
All lengths 9 > length >= 8, negative (top and bottom checked)

. . .

Negative Length 2
All lengths 2 > length >= 1, negative (top and bottom checked)
Negative Length 1

Positive Length 1
All lengths 2 > length >= 1, positive (top and bottom checked)
Positive Length 2

. . .

Positive Length 8
All lengths 9 > length >= 8, positive (top and bottom checked)
Positive Length 9
All lengths 10 > length >= 9, positive (top and bottom checked)
Positive Length 10