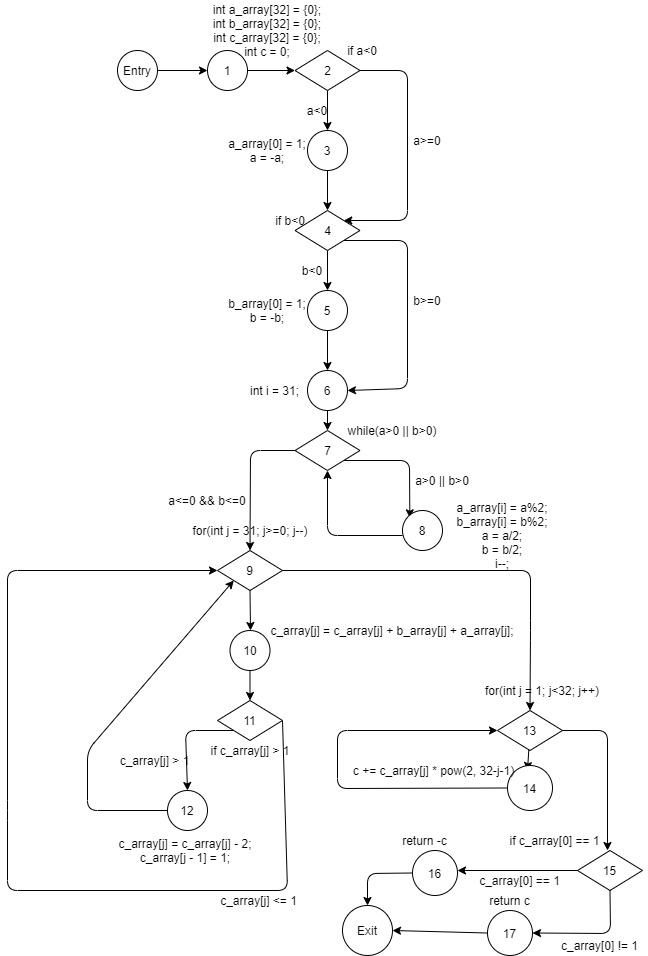
Q1

(1)



(2)

a) # nodes: 19

b) # edges: 25

c) # predicate nodes: 7

d) # regions: 8

e) V(G): 8

(3) V(G) = E – N + 2 = R = P + 1

Where E is the numbers of edges, N is the numbers of nodes, R is the numbers of regions, P is the numbers of predicate nodes + 1.

(4)

Entry, 1, 2, 4, 6, 7, 8, 7, 9, 13, 15, 17, Exit

Entry, 1, 2, 4, 6, 7, 8, 7, 9, 13, 15, 16, Exit

Entry, 1, 2, 4, 6, 7, 8, 7, 9, 13, 14, 13, 15, 16, Exit

Entry, 1, 2, 4, 6, 7, 8, 7, 9, 10, 11, 9, 13, 14, 13, 15, 16, Exit

Entry, 1, 2, 4, 6, 7, 8, 7, 9, 10, 11, 12, 9, 13, 14, 13, 15, 16, Exit

Entry, 1, 2, 4, 6, 7, 9, 10, 11, 12, 9, 13, 14, 13, 15, 16, Exit

Entry, 1, 2, 4, 5, 6, 7, 9, 10, 11, 12, 9, 13, 14, 13, 15, 16, Exit

Entry, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 9, 13, 14, 13, 15, 16, Exit

Q2

(1)

{<a=4, b=-4>, <a=-4, b=4>, <a=-4, b=-4>}

(2)

The numbers of predicate nodes are 7. They are:

Node 2 (correspond to line 8)

Node 4 (correspond to line 12)

Node 7 (correspond to line 17)

Node 9 (correspond to line 24)

Node 10 (correspond to line 26)

Node 13 (correspond to line 31)

Node 14 (correspond to line 33)

Consider the test set {<a=4, b=-4>, <a=-4, b=4>, <a=-4, b=-4>} which is the same as in (1).

When a=4 and b=-4, node 2 will be false, node 4 will be true, node 7 will be true, node 9 will be true, no bugs inside the for loop so we can escape the for loop, node 9 will become false, node 10 will be true since 100 + 100 = 200, c\_array[j] = 2, node 13 will be true, again we can escape the for loop, then node 13 will become false, node 14 will be false.

When a=-4 and b=4, node 2 will be true and node 4 will be false. The other is the same as above.

When a=-4 and b=-4, node 14 will be true.

However, the false situation of node 10 can never be reached. One more test case <a=2, b=4> is required since 10 + 100 = 110, c\_array[j] never greater than 1.

(3)

There is only one conditional statement contains two conditions. We need a test set with n + 1 = 2 + 1 = 3 test cases. (true + true, true + false, false + (true or false))

The test set required can be {<a=4, b=4>, <a=4, b=-4>, <a=-4, b=-4>}

(4)

No. Consider the path 7->9->13, this path is impossible. Since we cannot escape the for loop immediately without executing codes inside the for loop.

Q3

(1) One of the bugs will lead to wrong output for more than half of the inputs:

If one of the inputs is positive and another is negative. For example, a = -1, b = 1. Then, a\_array[0] = 1, b\_array[0] = 0, so c\_array[0] = 1. It comes out that c = -2. This is a wrong output.

Two of the bugs will make the program fail in some test cases:

Firstly, if a, b are so large such that a, b > 2^31, in the while loop 17, i may be negative. a\_array[-1] and b\_array[-1] are not defined.

Secondly, consider line 24, for j = 0, if c\_array[j] > 1, c\_array[j – 1] is not defined. This is array index problem.

The last bug will not affect the program in most cases:

The range of int is -2^31 to 2^31-1. If a or b is -2^31. After executing line 10 and 14, a or b becomes 2^31 which does not lie in the range of int. This is an integer overflow problem.

(2) The program wants to achieve addition of two numbers. The program first changes a and b to binary number which is represented by array. Then, it performs addition of array. Finally, the array will convert back to integer.

(3)

Testing strategy:

(1) Generate zeroes a and b; Test case: <a=0, b=0>

(2) Generate small a and b; Test case: <a=1, b=1>

(3) Generate large a and b; Test case: <a=2^33, b=2^33>

(4) Generate positive a and negative b; Test case: <a=1, b=-1>

(5) Generate negative a and positive b; Test case: <a=-1, b=1>

(6) Generate negative a and negative b; Test case: <a=-1, b=-1>

(4) For white-box testing, the answer is no. White-box testing can only test all program paths. It may find bugs only if all test sets are designed accordingly. Suppose we have run all paths, we still don’t get there is stack overflow problem. We can find the bug if a and b are too large by white box testing. For black-box testing, the answer is yes. Black-box testing can have infinity possibilities. If there is bug in some extreme case, the bug can be found.

Q4

(1) Driver Modules:

Simulated *account management module*: The module will be called by *income calculation module*.

Stub Modules:

Simulated *GUI module*: The module can randomly generate a valid user account and password and it will send the information to *account management module*.

Simulated *live streaming module*: The module can randomly generate number of customers and the gift value and then send to *data recording module*.

Simulated *data recording module*: The module receives the data from *live streaming module*. Then, it can be called by *income calculation module*.

Simulated *transaction module*: The module can randomly generate a bank account. Then, the module will be called by *income calculation module*.

Simulated *emailing module*: The module can randomly generate an email address. Then, the module will be called by *income calculation module*.

(2)

|  |  |  |
| --- | --- | --- |
| Number of customers | Gift value | Expected income |
| 8256 | 520 | 424 HKD |
| 12863 | 2753 | 1357.2 HKD |
| 125742 | 8641 | 5538.35 HKD |
| 1246732 | 12845 | 28787.5 HKD |

(3)

Assume every customer will contribute 0.11 HKD per live streaming.

Let x be the numbers of customers.

For 0 <= x <= 9999,

x \* 0.02 + x \* 0.11 \* 0.5 = 0.075x

8256 \* 0.075 = 619.2 > 424

For 10000 <= x <= 99999,

x \* 0.02 + x \* 0.11 \* 0.4 = 0.064x

12863 \* 0.064 = 823.232 < 1357.2

For 100000 <= x <= 999999,

x \* 0.02 + x \* 0.11 \* 0.35 = 0.0585x

125742 \* 0.0585 = 7355.907 > 5538.35

For x >= 1000000,

x \* 0.02 + x \* 0.11 \* 0.3 = 0.053x

1246732 \* 0.053 = 66076.796 > 28787.5

In the range, 10000 <= x <= 99999, the actual income is greater than the expected income by big data analysis. So I suggest the ranges of customer numbers be 10000~99999.