3. Mutant 1:

On line 11, return 0 instead of return 1;

Input: list = node1;
list->next = node1;

Output:
original: 1 mutant: 0

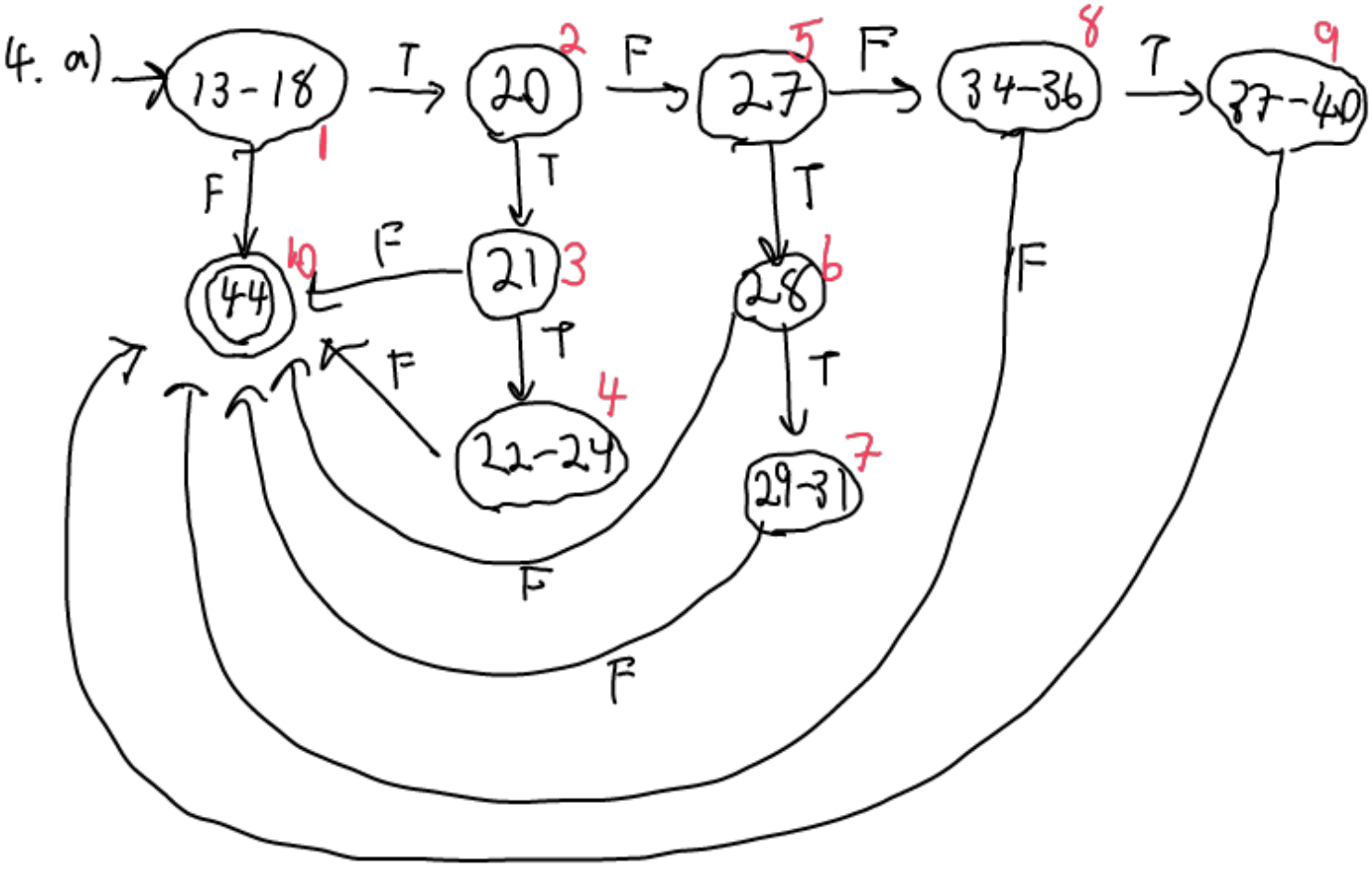
Mutant 2:

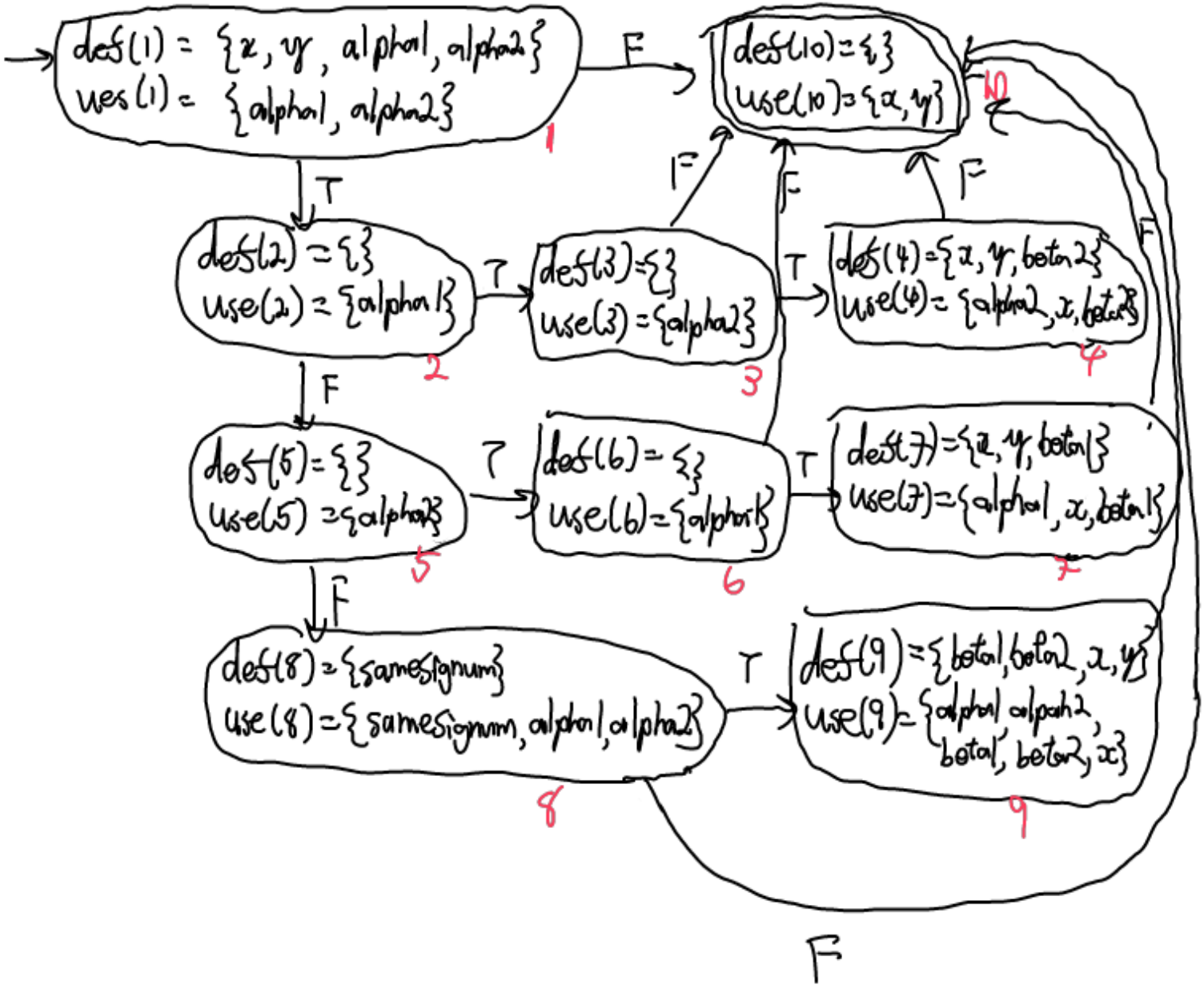
On line 9, change while(1) { to while(0) {.

Input: list = node1;
list->next = node1;

Output:
Original: 1 Mutant: 0

4. a)





b) $TR_{ppc} = \{[1, 10], [1, 2, 3, 10], [1, 2, 3, 4, 10], [1, 2, 5, 6, 10], [1, 2, 5, 6, 7, 10], [1, 2, 5, 8, 10], [1, 2, 5, 8, 9, 10]\}$

c) $TR_{ADUPC} = \{[1,2], [1,2,3], [1,2,3,4], [4,10], [1,2,5], [1,2,5,6], [1,2,5,6,7], [7,10], [1,2,5,8], [1,2,5,8,9], [9,10], [1,2,3,10], [1,2,5,6,10], [1,2,5,8,10], [1,10]\}$

d) Since this CFG is acyclic, as proved by Q1, we know that TR_{ppc} covers complete path coverage, which means it visits every possible subpaths. The strength of this is that it tests every single reachable code in the function. It gives a

comprehensive check over the whole program.
In the case of TR_{ADUPC}, it covers the definition and use of every variable. While it may not go through all of the possible paths, it checks every possible use of variables, whereas TR_{PPC} may not test the variables as thoroughly. In short, TR_{PPC} is good to roughly test every aspect of a program, while TR_{ADUPC} is good to thoroughly test every variable.

It is not worth it for TR_{ADUPC} to achieve TR_{APC}. This is because TR_{ADUPC} already tests every occurence of variables. Therefore giving it more test requirements will not give additional meaning. It is already serving its purpose well.

e) T1: computeIntersection([1,1], [2,2], [1,1], [2,2]);
$TR_1 = \{[1,10]\}$ alpha1 = 1, alpha2 = 1, sameSignum = true;
Output: [2,2]

T2: computeIntersection([1,1], [2,4002], [1,1], [2,4003]);
$TR_2 = \{[1,2,3,10]\}$ alpha1 = 4001, alpha2 = 4002, sameSignum = true;
Output: [2,4002]

T3: computeIntersection([1,1], [2,4002], [1,1], [2,2]);
$TR_3 = \{[1,2,3,4,10]\}$ alpha1 = 4001, alpha2 = 1, sameSignum = true;
Output: [1,1]

T4: computeIntersection([1,1], [2,4001], [1,1], [4002]);
$TR_4 = \{[1,2,5,6,10]\}$ alpha1 = 4000, alpha2 = 4001, sameSignum = true;
Output: [1,1]

T5: computeIntersection([1,1], [2,2], [1,1], [2,4002]);

TR_5 = {[1,2,5,6,7,10]} alpha1 = 1, alpha2 = 4001, sameSignum = true

output: [1, 1]

T6: computeIntersection([1,1], [2,2], [1,1], [2,2.00001]);

TR_6 = {[1,2,5,8,10]} alpha1 = 1, alpha2 = 1.00001, sameSignum = true

output: [1,1]

T7: computeIntersection([1,1], [2,2], [1,1], [2,3]);

TR_7 = {[1,2,5,8,9,10]} alpha1 = 1, alpha2 = 2, sameSignum = true

output: [-1, -1]