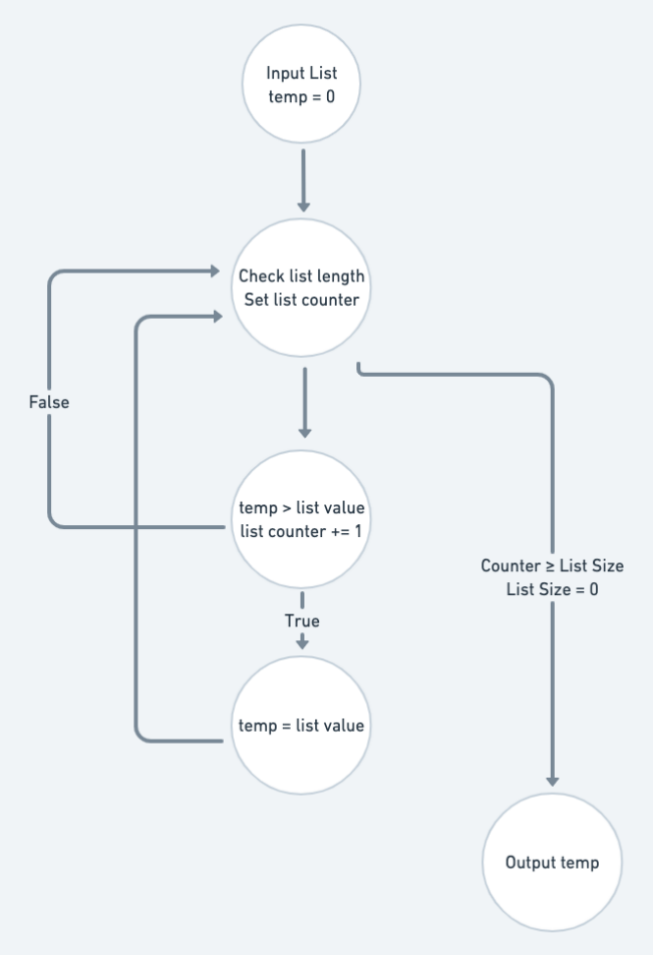
Problem 11.1

The program I'm considering is one that finds the largest number within an array of positive integers. The following diagram is a CFG for the program:



Extension 1: Determine number of paths and sensitize a few of your test cases.

There are infinite number of paths for the CFG since it contains a loop and we can go through the loop for as long as the array can be iterated over. Some test cases for the program:

- **Test 1:** List = [], an empty list should output 0, since there are no elements the program will take the input, check the length of the list, and return an output.
- **Test 2:** List = [1], a list with one element will go through the loop once, update the temp variable and list counter, compare list counter to list size and output the value of temp.
- **Test 3:** List = [1,2], the program will loop multiple times and produce an output of 2.

Extension 2: If you CFG contains no loops, try to modify to include at least one loop.

The CFG already has a loop.

Extension 3: What is your loop testing strategy, and the corresponding test cases for it?

Since testing loops can lead to testing infinite test cases, we can use upper and lower boundaries to help ensure that all the test cases are covered. The way to do so will be to create three test cases as follows:

Case 1: Bypass the loop

Case 2: Check if program can enter the loop for one iteration

Case 3: Check if program can stay in the loop for more than one iteration (2 iterations)

The test cases created in Extension 1, correspond to the above three cases. Test 1 corresponds to Case 1, Test 2 corresponds to Case 2, and Test 3 corresponds to Case 3.

Extension 4: What is your loop testing strategy when you introduce nested loops in your CFG?

To test nested loops, we need to set upper and lower bounds on our graph. We can use the following cases:

Case 1: Bypass the loop

Case 2: Check if program can loop through inner loop once for one outer loop iteration

Case 3: Reduce inner loop to one node and conduct multiple iterations of outer loop

We reduce the inner loop because with every iteration of the outer loop the number of paths grow exponentially, and this becomes very hard to keep track and test when both the inner and outer loop have several iterations. Thus, reduction helps ensure that we can test the program thoroughly while also making sure that the inner loop works via Case 2.

Problem 11.4e

Assuming that C2 is x >= 0 and C1 is x > 100.

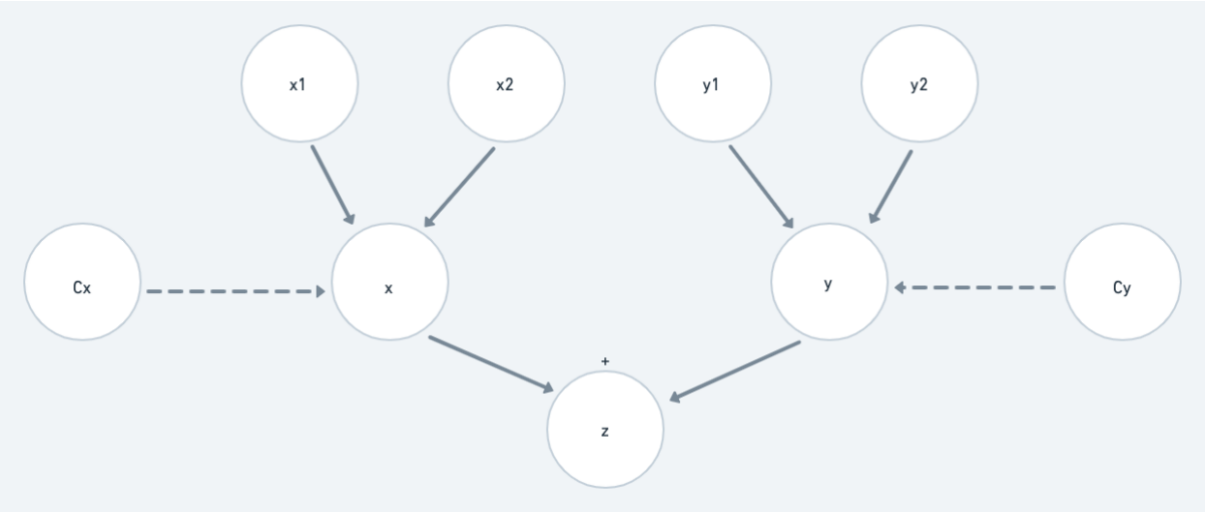
C2	C1	C2 $\Rightarrow$ C1
T	T	T
T	F	F
F	T	T
F	F	T

When a concatenation is implemented between two sub-graphs G1 and G2, the number of test cases will be the multiplication of both possible paths' numbers. Hence, because C1 and C2 are binary conditions, we should end up with 4 possible paths. However, that's not quite true when a relation like implication is present between our conditions.

The above truth table captures the results of the condition as stated within the CFG. We can see that when C2 is True and C1 is False our condition is violated. Thus, this case can't be sensitized. So, we can sensitize the TT, FT, and FF paths for the CFG but not the TF path (The first letter in TT, FT, FF, and TF corresponds to the C2 and the second corresponds to C1).

Problem 11.9

The program I'm considering finds the maximum sum of two numbers from two sets of numbers. The following is the DDG for this program:



Extension 1: Determine the number of slices and sensitize a few of your test cases.

There are 4 slices of the graph. The test cases would be based to ensure that all the pairs in [{x1,y1}, {x1,y2}, {x2,y1}, {x2,y2}] get tested. Some sample test cases are as follows:

Test 1: {x1, x2} = {1,2}, {y1, y2} = {3,4} => z = 6
Test 2: {x1, x2} = {2,1}, {y1, y2} = {3,4} => z = 6
Test 3: {x1, x2} = {1,2}, {y1, y2} = {4,3} => z = 6
Test 4: {x1, x2} = {2,1}, {y1, y2} = {4,3} => z = 6

Extension 2: If you have only a single data selector in your DDG, try to add at least one more.

I have multiple data selectors in the DDG.

Extension 3: For multiple data selectors in your DDG, how do you handle correlated control input conditions?

If control input conditions are correlated that means the values, they are using to make data selection decisions must originate from an earlier node. So, we would connect these control input conditions to the same relevant node to ensure that their data dependency is on the same node that makes them correlated. For instance, if we have two data selectors such as (x > 0) and (x <= 17), then these two nodes will be connected to the node that determines the value of x, thus ensuring that these correlated control input conditions are handled appropriately.

Extension 4: If you have loops in your program or repetitions in your specification, how would your DDG be different, and how would you perform DFT for it?

The key difference would be that the DDG would have multiple levels of value assignment to a variable. This is because a loop that is being run n times is basically an if condition that is being executed n times. So, for every if condition we would have a data

selector level. The way to perform DFT for loops within a DDG we use upper and lower bounds just like we did for CFG. We set the following cases:

Case 1: Bypass the loop
Case 2: Enter the loop for just one iteration
Case 3: Enter the loop for two iterations

We can also choose to perform testing of our loops using CFG instead of using DDG.