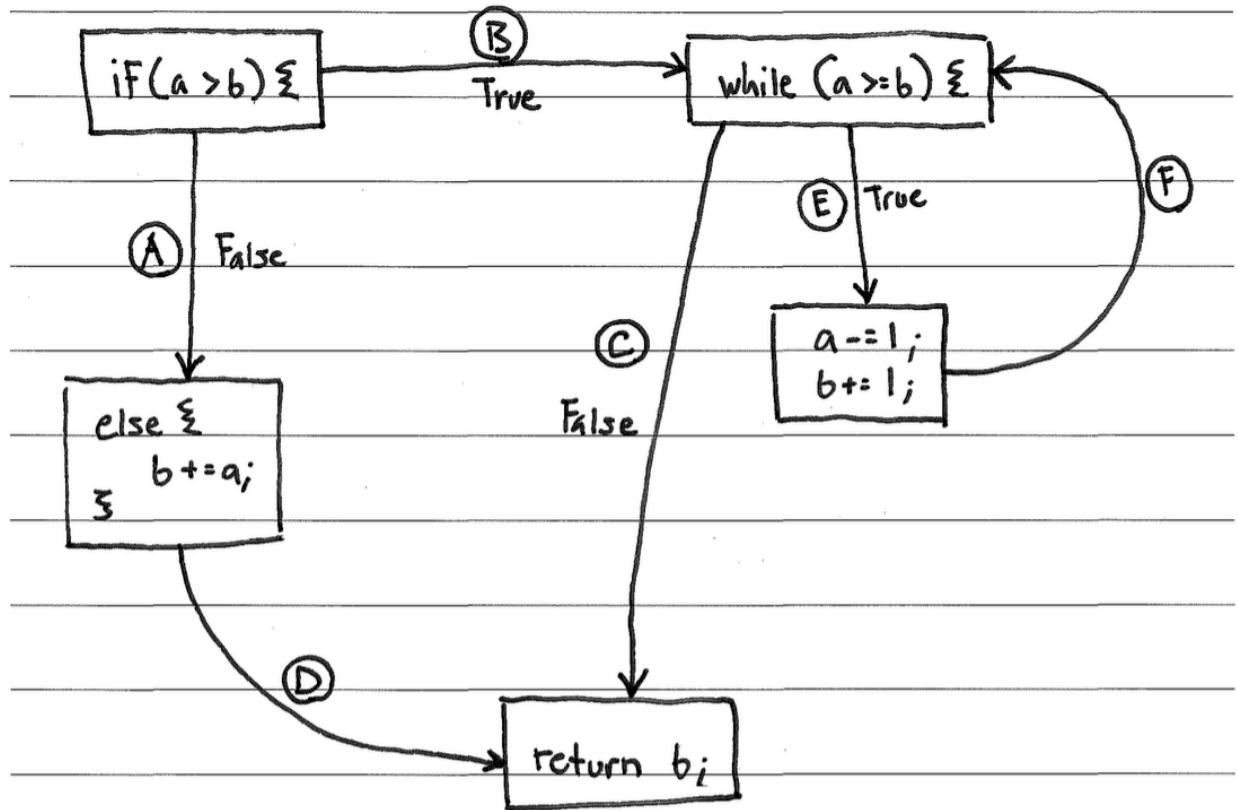


Solution:



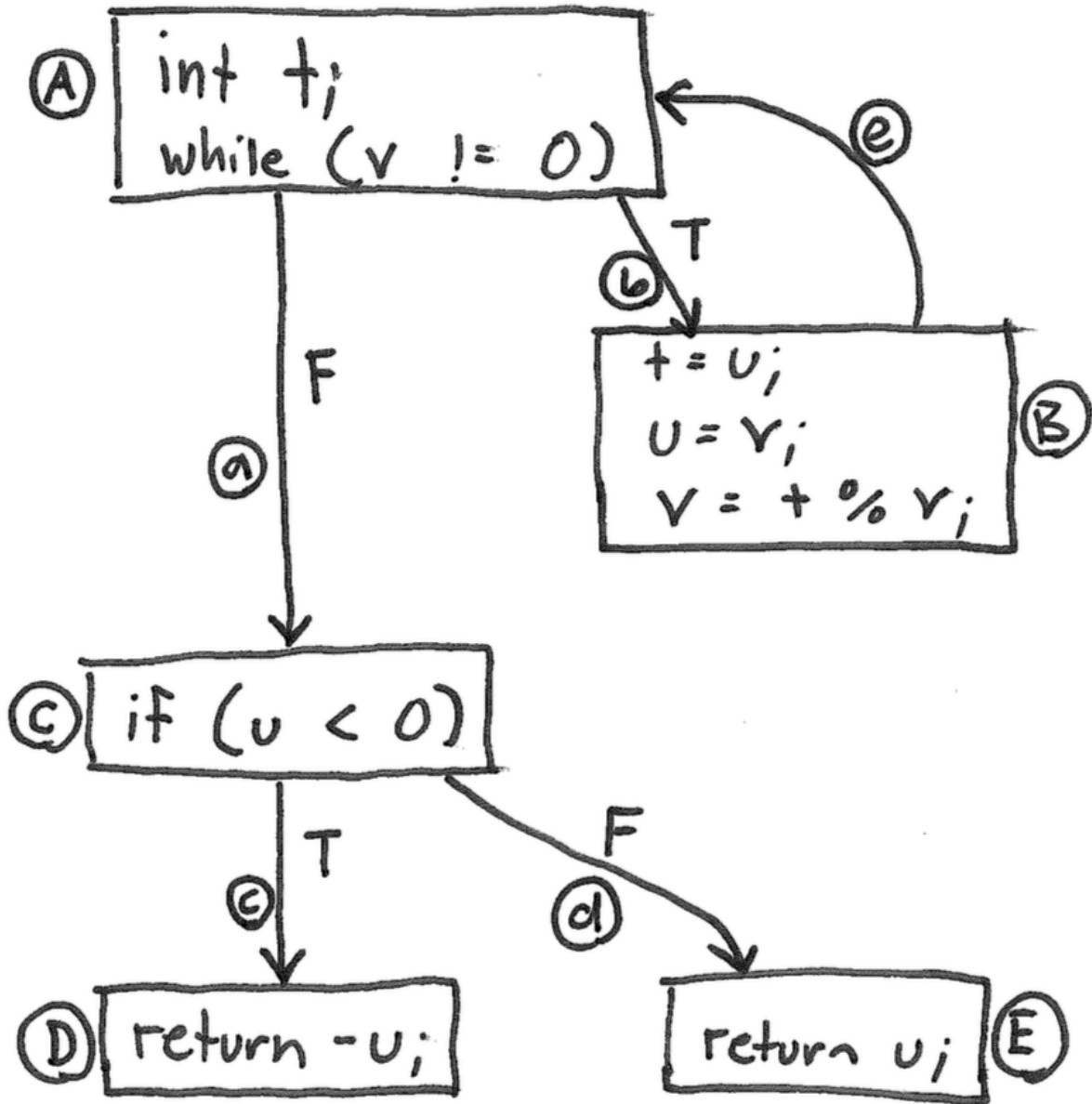
Solution:

Input		Covers
x	y	
1	2	AD
N/A	N/A	BC
1	0	BEFC
4	2	BEFEFC

Problem: Draw a control flow diagram for this function. Label each node in the graph with a capital letter, and label each edge with a lowercase letter.

```
int blammo(int u, int v) {  
    int t;  
    while (v != 0) {  
        t = u;  
        u = v;  
        v = t % v; // Recall that % computes remainder of t/v  
    }  
    if (u < 0) { return -u; }  
    return u;  
}
```

Solution:



Problems:

1. Fill in the table below with a test suite that provides statement coverage of the “blammo” code. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Covers
u	v	

2. Fill in the table below with a test suite that provides path coverage of the “blammo” code. Cover no more than 1 iteration of the loop. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Covers
u	v	

Solutions:

1.

Input		Covers
u	v	
2	2	A, B, C, E
-1	0	A, C, D

2.

Input		Covers
u	v	
-1	0	a, c
0	0	a, d
-2	-2	b, e, a, c
2	2	b, e, a, d

Paths:

a, c

a, d

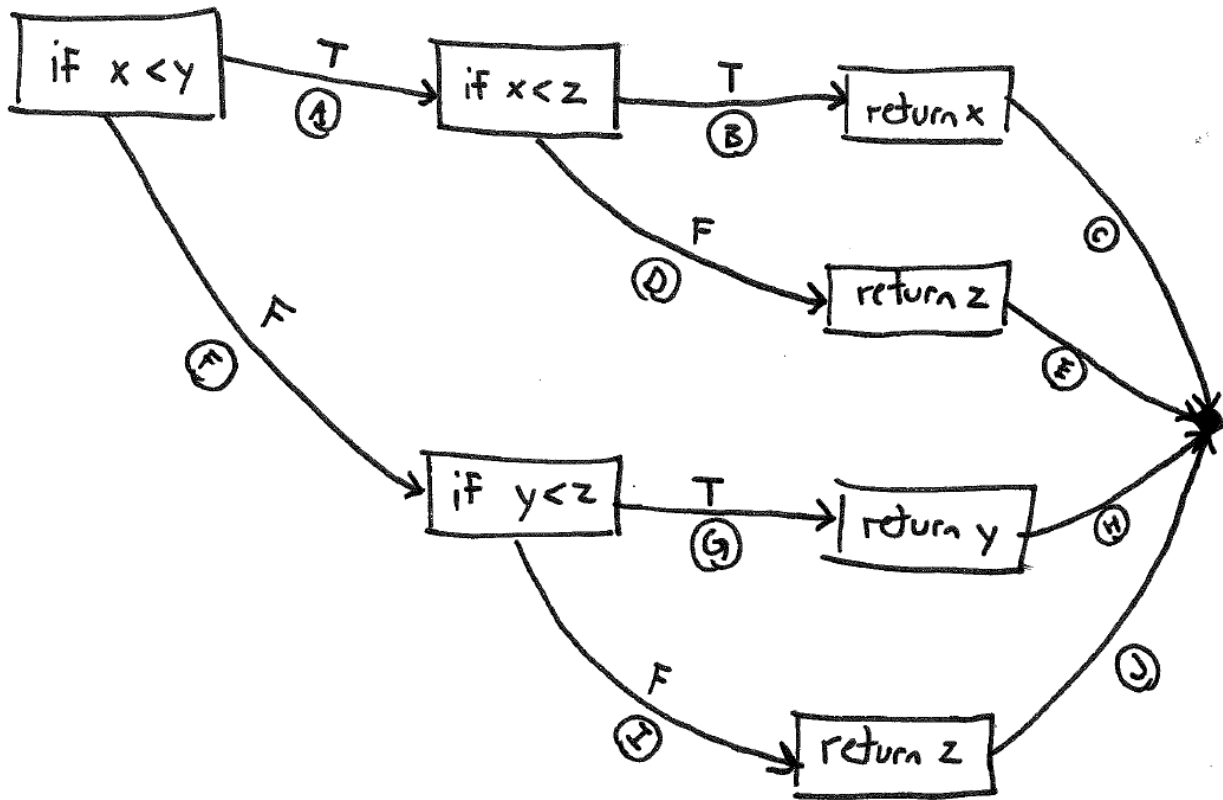
b, e, a, c

b, e, a, d

Problem: Draw a control-flow graph for the following function. Label each edge in the graph with an uppercase letter.

```
def min_of_three(x, y, z)
  if x < y then
    if x < z then
      return x
    else
      return z
    end
  else
    if y < z then
      return y
    else
      return z
    end
  end
end
```

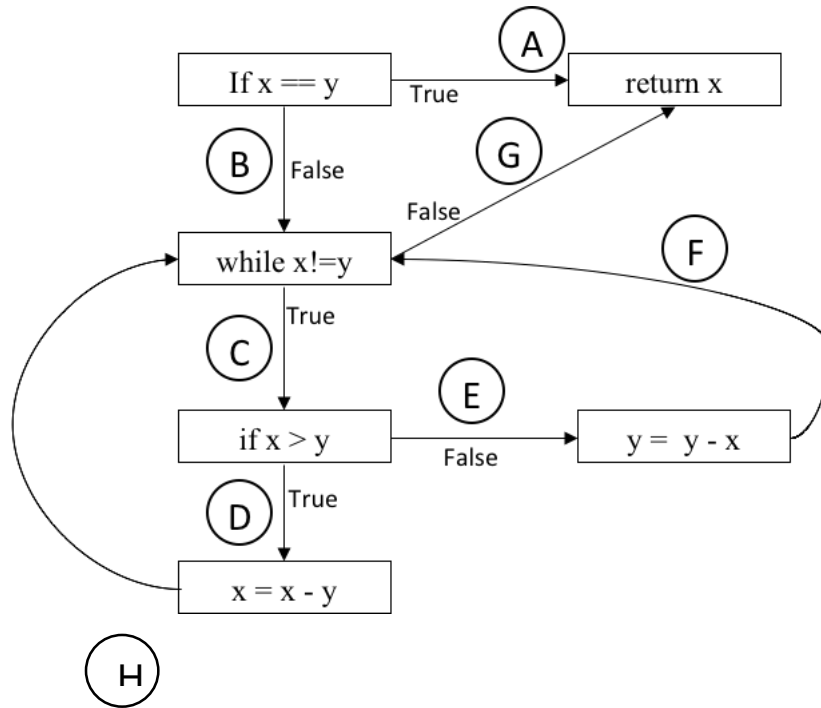
Solution:



Solution:

Input			Expected Output	Covers
x	y	z		
1	2	2	1	A, B, C
2	3	1	1	A, D, E
2	1	2	1	F, G, H
3	2	1	1	F, I, J

Consider the following control-flow graph for a gcd function in answering the questions below.



Solution: Condition Coverage

Input		Expected Output	Covers
x	y		
1	1	1	A
1	2	1	B, C, E, G
2	1	1	B, C, D, G
3	2		B, C, D, C, E, G

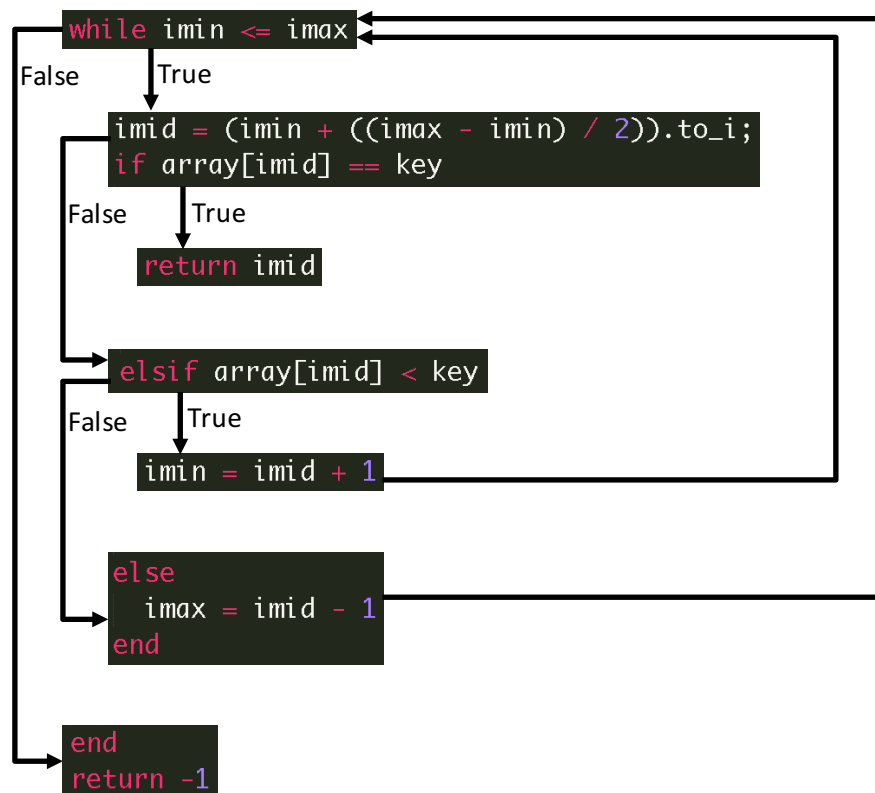
alternatively (with a bracket and arrow pointing to the second and third rows)

Solution: Path Coverage

Input		Expected Output	Covers
x	y		
1	1	1	A
2	1	1	B, C, D, H, G
1	2	1	B, C, E, F, G
			B, G ← not possible

Consider this binary-search function and its associated control-flow graph.

```
def binary_search(array, key, imin, imax)
  while imin <= imax
    imid = (imin + ((imax - imin) / 2)).to_i;
    if array[imid] == key
      return imid
    elsif array[imid] < key
      imin = imid + 1
    else
      imax = imid - 1
    end
  end
  return -1
end
```



Problems:

Consider the following test cases for the `binary_search` function.

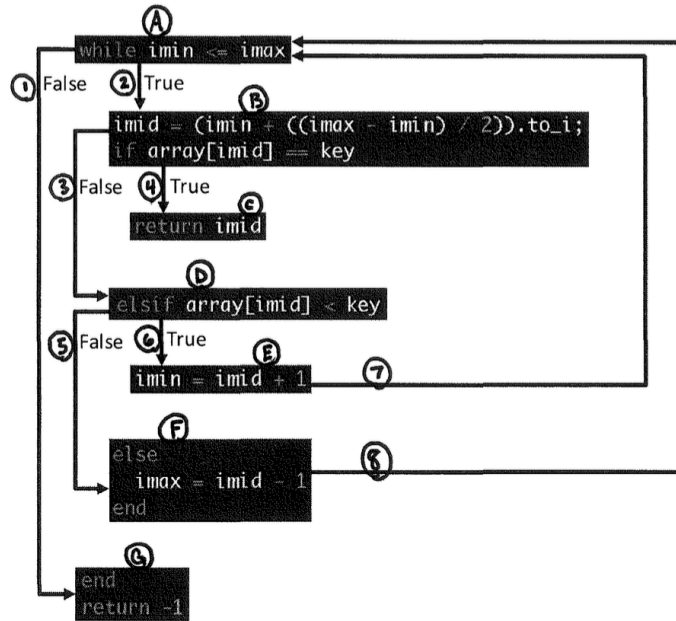
	array	key	imin	imax
a.	[1]	0	0	0
b.	[1]	1	0	0
c.	[1]	1	1	0
d.	[1, 2, 3]	1	0	2
e.	[1, 2, 3]	2	0	2
f.	[1, 2, 3]	3	0	2
g.	[1, 2, 3]	1	2	0
h.	[1, 2, 3]	2	2	0
i.	[1, 2, 3]	3	2	0

1. Select tests from the above to create a test suite that provides statement coverage of the `binary_search` function. Your suite should contain the minimum number of tests to provide the coverage.

2. Select tests from the above to create a test suite that provides condition coverage of the `binary_search` function. Your suite should contain the minimum number of tests to provide the coverage.

3. Select tests from the above to create a test suite that provides path coverage of the `binary_search` function. Cover only paths that contain one loop iteration or fewer (i.e., no path should enter the loop more than once). Your suite should contain the minimum number of tests to provide the coverage.

Solutions:



	array	key	imin	imax	Statements Covered	Edges ² Covered
a.	[1]	0	0	0	ABDFAG	23581
b.	[1]	1	0	0	ABC	24
c.	[1]	2	0	0	ABDEAG	23671
d.	[1, 2, 3]	1	0	2	ABDFABC	235824
e.	[1, 2, 3]	2	0	2	ABC	24
f.	[1, 2, 3]	3	0	2	ABDEABC	236724
g.	[1, 2, 3]	1	2	0	AG	1
h.	[1, 2, 3]	2	2	0	AG	1
i.	[1, 2, 3]	3	2	0	AG	1

1.

a, F or c, d
 (Need to cover statements A, B, C, D, E, F, G)

2.

a, F or c, d
 (Need to cover edges 1, 2, 3, 4, 5, 6)

3.

$(g|h|i)$, $(b|e)$, c, a
 Any 1 of these Any 1 of these

Possible paths	Tests that cover
1	g, h, i
24	b, e
23671	c
23581	a

Problems:

Consider the following test cases for the `binary_search` function.

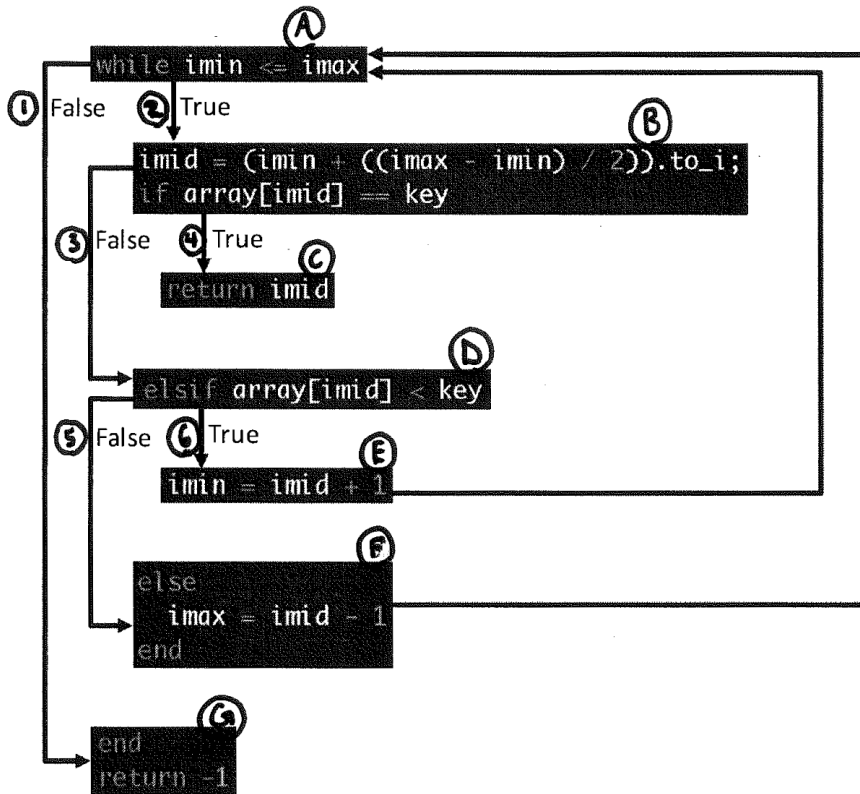
	<code>array</code>	<code>key</code>	<code>imin</code>	<code>imax</code>
a.	<code>[0]</code>	0	0	0
b.	<code>[0]</code>	1	0	0
c.	<code>[0]</code>	1	1	0
d.	<code>[0]</code>	-1	0	0

1. Select tests from the above to create a test suite that provides statement coverage of the `binary_search` function. Your suite should contain the minimum number of tests to provide the coverage.

2. Select tests from the above to create a test suite that provides condition coverage of the `binary_search` function. Your suite should contain the minimum number of tests to provide the coverage.

3. Select tests from the above to create a test suite that provides path coverage of the `binary_search` function. Cover only paths that contain one loop iteration or fewer (i.e., no path should enter the loop more than once). Your suite should contain the minimum number of tests to provide the coverage.

Solutions:



	array	key	imin	imax	Statements Covered	Conditions Covered
a.	[0]	0	0	0	A B C	2 4
b.	[0]	1	0	0	A B D E G	2 3 6 1
c.	[0]	1	1	0	A G	1
d.	[0]	-1	0	0	A B D F G	2 3 5 1

Path	Covered by
A G	c
A B C	a
A B D E A G	b
A B D F A G	d

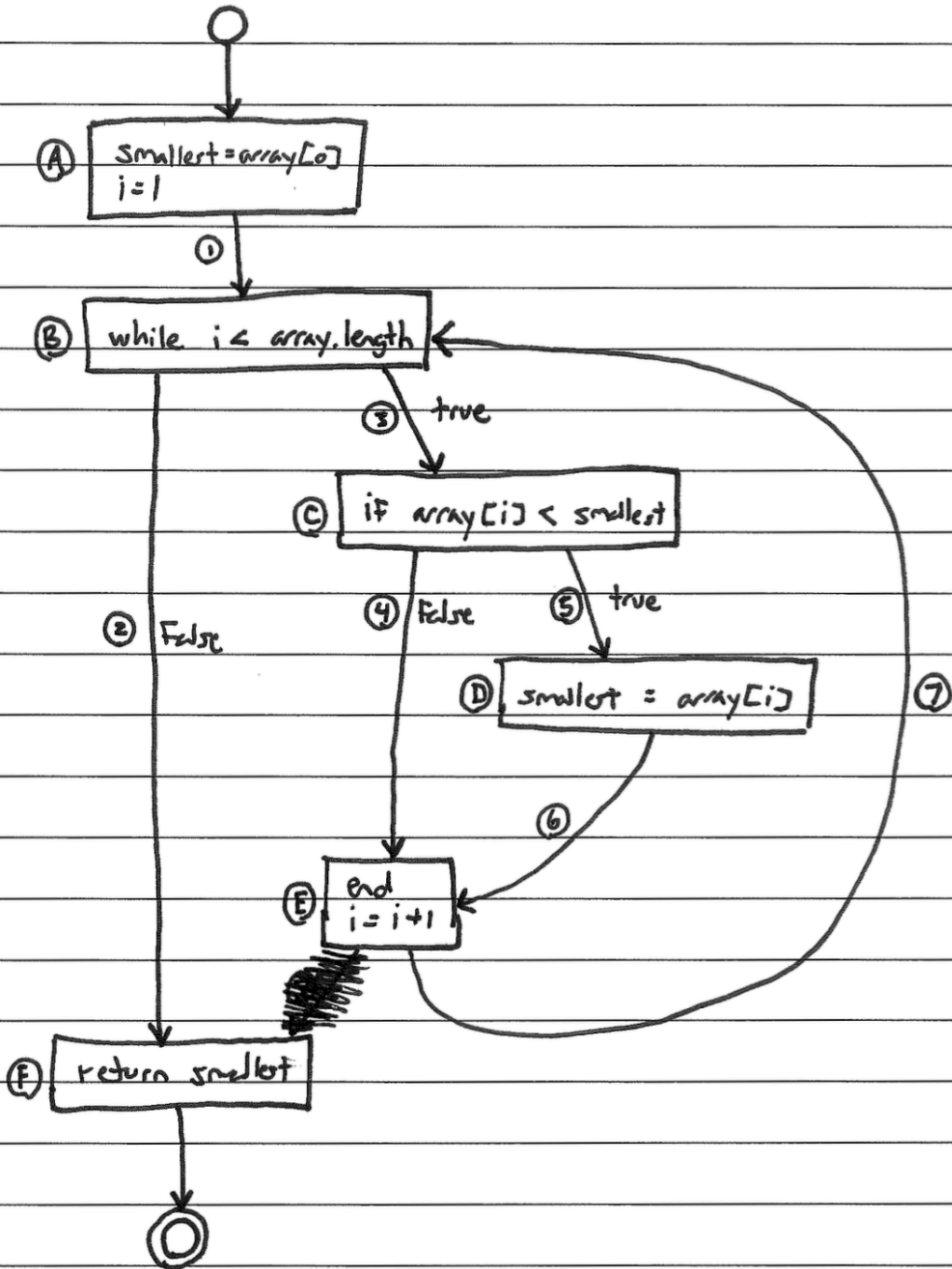
1. a, b, d
2. a, b, d
3. a, b, c, d

Consider this figure in answer the following questions.

```
def find_smallest(array)
  smallest = array[0]
  i = 1
  while i < array.length
    if array[i] < smallest
      smallest = array[i]
    end
    i = i + 1
  end
  return smallest
end
```

Figure 1. Function that finds the smallest value in an array.

Solution:



3. Fill in the table below with a test suite that provides path coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case. You need only cover executions that involve at most 1 iteration of each loop (if there are any). Before you fill in the table, list all the paths to be covered.

Paths:

Input array	Expected Output	Covers

Solutions:

Multiple solutions are possible. These are just examples of correct solutions.

1.

Input array	Expected Output	Covers
[1,0]	0	A, B, C, D, E, B, F

2.

Input array	Expected Output	Covers
[1,0,2]	0	3, 5, ^{3,} 4, 2

3.

- 1, 2

- 1, 3, 5, 6, 7, 2

- 1, 3, 4, 7, 2

Input array	Expected Output	Covers
[0]	0	1, 2
[1,0]	0	1, 3, 5, 6, 7, 2
[0,1]	0	1, 3, 4, 7, 2