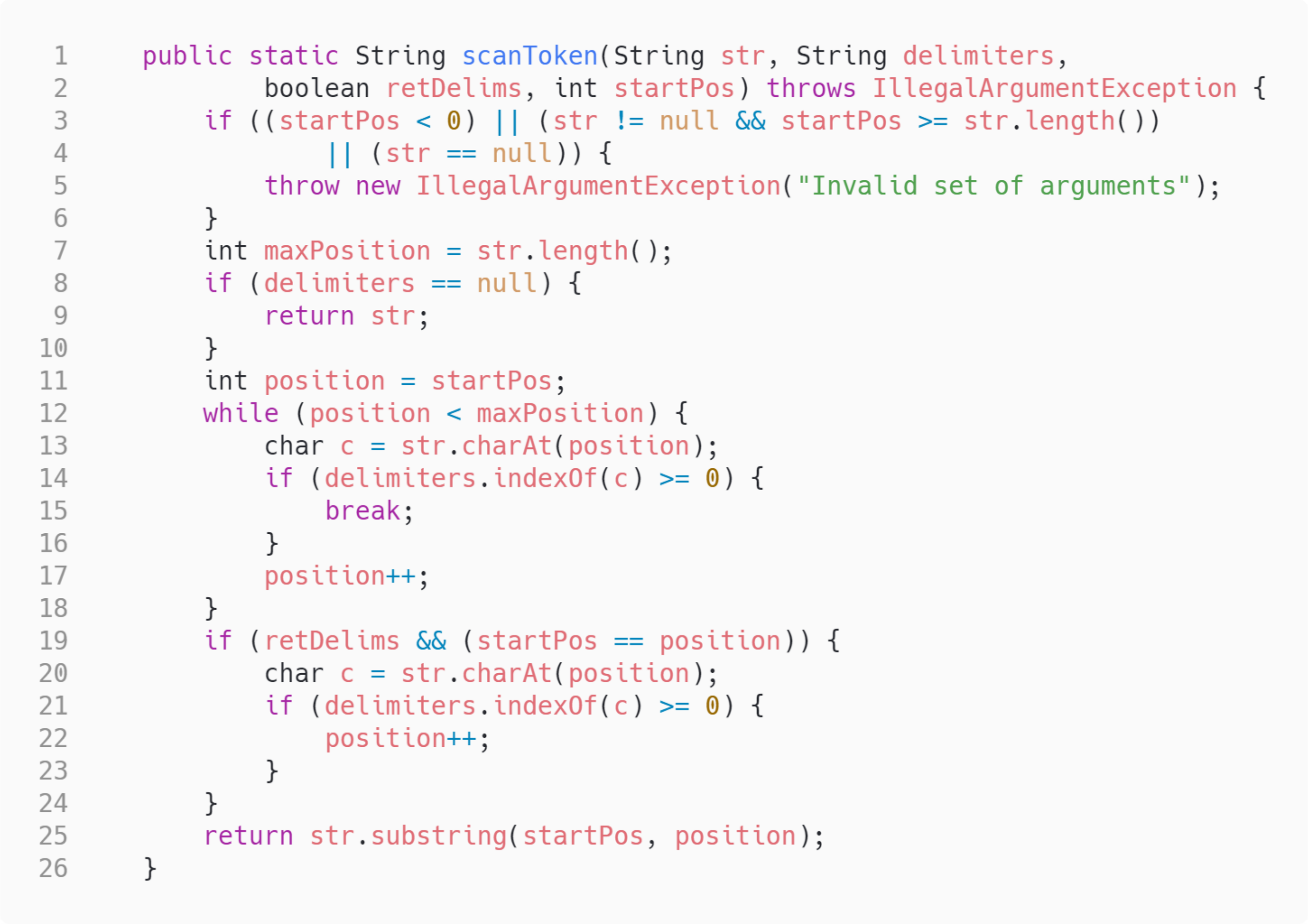
Name: Devansh Amin

For the following 3 coverage criteria questions use the following code:

```
/**
 * Skips ahead from startPos and returns the next token specified by the next
 * delimiter character encountered, or entire string if no such delimiter is
 * found. (Inspired by a simplified version of the actual code of
 * java.util.StringTokenizer)
 *
 * @param str String to be searched
 * @param delimiters String of character delimiters
 * @param retDelims Should delimiters be returned
 * @param startPos Index in the string to start search
 * @return Returns the next token
 * @throws IllegalArgumentException if str is null or startPos is invalid
 */
```



```
public static String scanToken(String str, String delimiters,
        boolean retDelims, int startPos) throws IllegalArgumentException {
    if ((startPos < 0) || (str != null && startPos >= str.length())
            || (str == null)) {
        throw new IllegalArgumentException("Invalid set of arguments");
    }
    int maxPosition = str.length();
    if (delimiters == null) {
        return str;
    }
    int position = startPos;
    while (position < maxPosition) {
        char c = str.charAt(position);
        if (delimiters.indexOf(c) >= 0) {
            break;
        }
        position++;
    }
    if (retDelims && (startPos == position)) {
        char c = str.charAt(position);
        if (delimiters.indexOf(c) >= 0) {
            position++;
        }
    }
    return str.substring(startPos, position);
}
```

1. Input space partitioning [20 pts]

Using the interface description above for the **scanToken** function, apply ISP and identify BCC coverage requirements. Be sure to clearly label your base choice. **For your ISP analysis use a Interface-Based Approach, *not* a Functionality-Based approach.** In other words your analysis should be based on not needing to fully understand the inner workings of the code beyond what is described in the method comments. You do not need to identify the data for test cases, just the BCC coverage requirements. You may use the table below or the Excel spreadsheet attached to the final on Blackboard.

Using the code on the previous page complete the following 3 tables.

Solution:

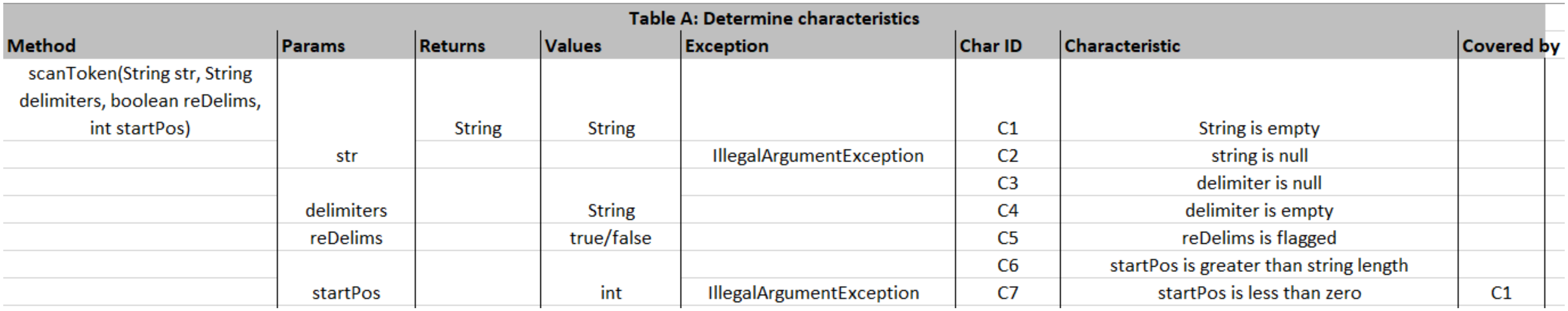


Table A: Determine characteristics

Method	Params	Returns	Values	Exception	Char ID	Characteristic	Covered by
scanToken(String str, String delimiters, boolean reDelims, int startPos)		String	String		C1	String is empty	
	str			IllegalArgumentException	C2	string is null	
					C3	delimiter is null	
	delimiters		String		C4	delimiter is empty	
	reDelims		true/false		C5	reDelims is flagged	
					C6	startPos is greater than string length	
	startPos		int	IllegalArgumentException	C7	startPos is less than zero	C1

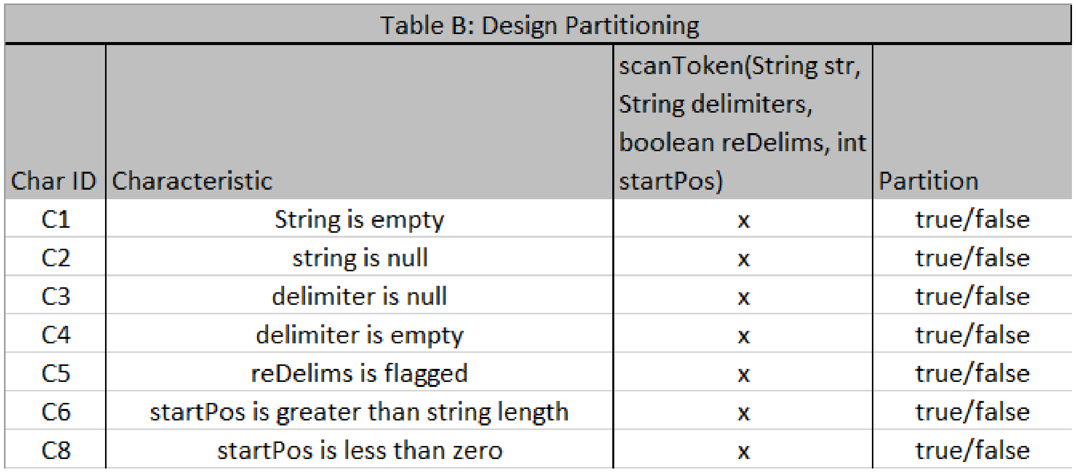


Table B: Design Partitioning			
Char ID	Characteristic	scanToken(String str, String delimiters, boolean reDelims, int startPos)	Partition
C1	String is empty	x	true/false
C2	string is null	x	true/false
C3	delimiter is null	x	true/false
C4	delimiter is empty	x	true/false
C5	reDelims is flagged	x	true/false
C6	startPos is greater than string length	x	true/false
C8	startPos is less than zero	x	true/false

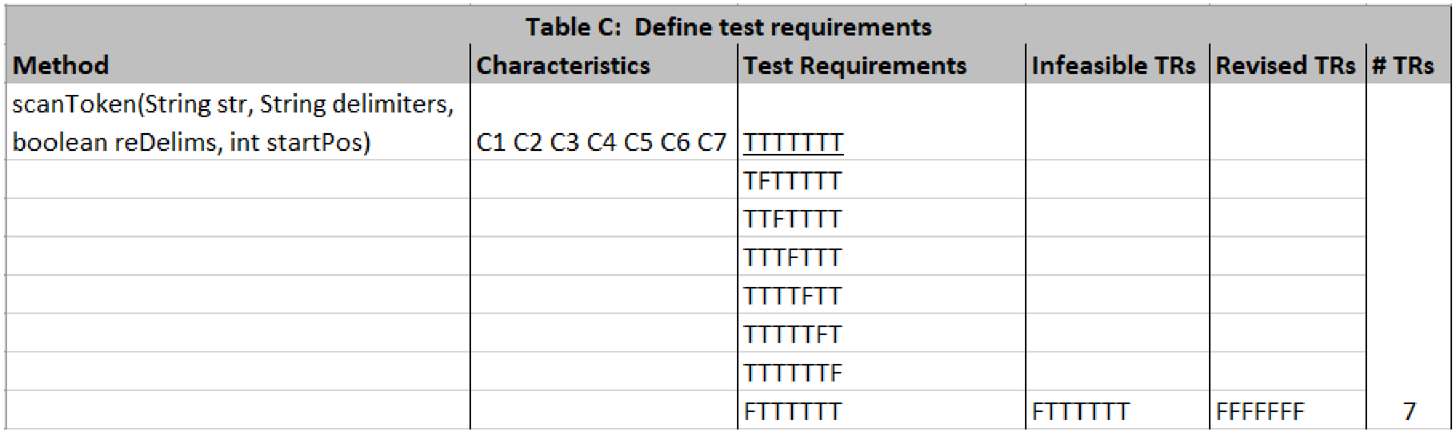


Table C: Define test requirements					
Method	**Characteristics**	**Test Requirements**	**Infeasible TRs**	**Revised TRs**	**# TRs**
scanToken(String str, String delimiters, boolean reDelims, int startPos)	C1 C2 C3 C4 C5 C6 C7	TTTTTTT			
		TFTTTTT			
		TTFTTTT			
		TTTFTTT			
		TTTTFTT			
		TTTTTFT			
		TTTTTTF			
		FTTTTTT	FTTTTTT	FFFFFFF	7

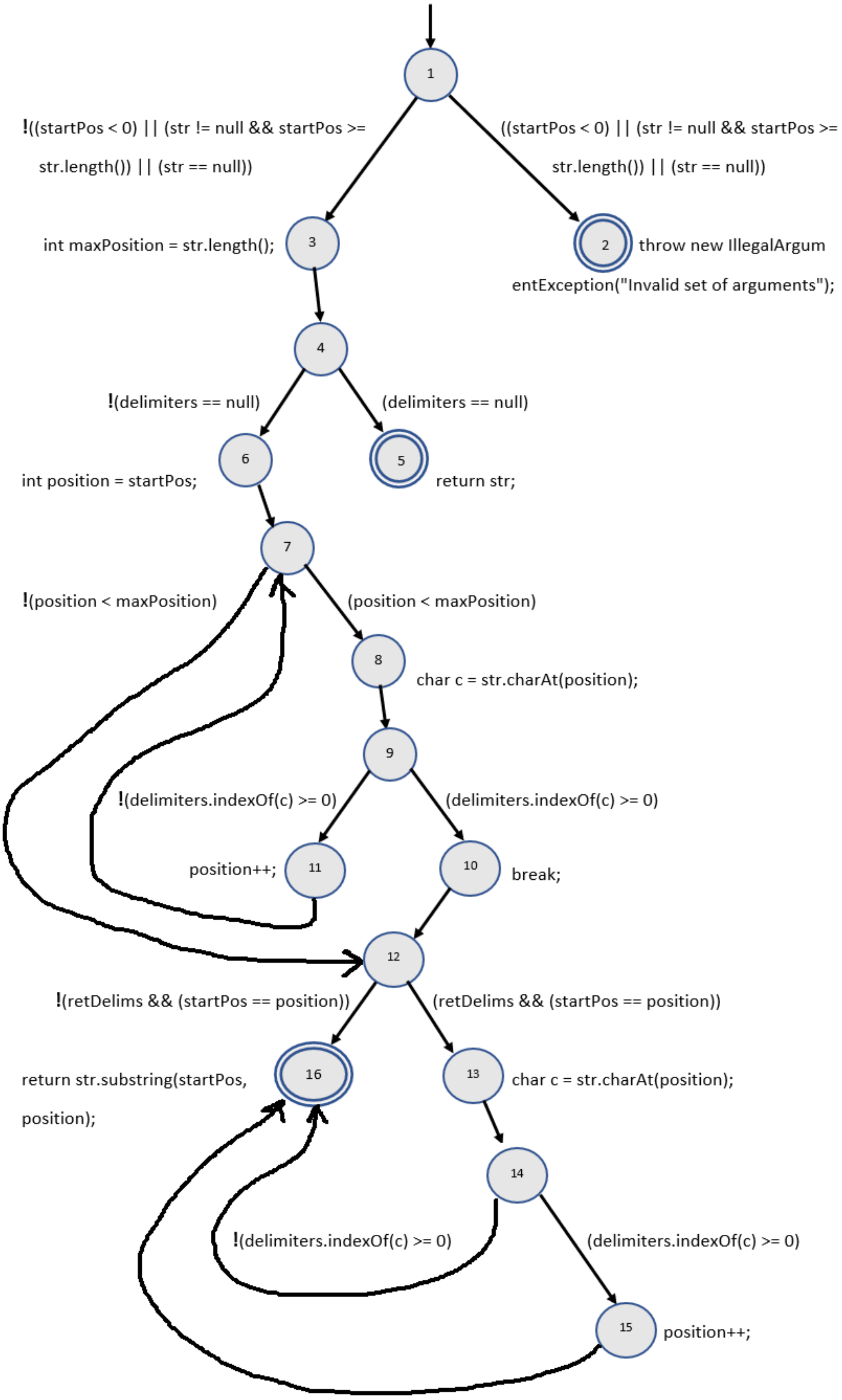
2. Graph Coverage Criteria [30 pts]

Using the **scanToken** code on page 1 complete the items below.

a. Draw the control flow graph that represents abstraction of its execution. (Label edges and nodes in the graph with the corresponding code fragments)
b. List the test requirements for Node Coverage
c. List the test requirements for Edge Coverage
d. List the test requirements for Edge-Pair Coverage
e. List all simple paths
f. List the prime paths
g. Extend the prime paths to create a set of test paths TR that provide Prime Path Coverage (PPC)

Solution:

a)


1
!((startPos < 0) || (str != null && startPos >= str.length()) || (str == null))
((startPos < 0) || (str != null && startPos >= str.length()) || (str == null))
3
int maxPosition = str.length();
2
throw new IllegalArgumentException("Invalid set of arguments");
4
!(delimiters == null)
(delimiters == null)
6
int position = startPos;
5
return str;
7
!(position < maxPosition)
(position < maxPosition)
8
char c = str.charAt(position);
9
!(delimiters.indexOf(c) >= 0)
(delimiters.indexOf(c) >= 0)
11
position++;
10
break;
12
!(retDelims && (startPos == position))
(retDelims && (startPos == position))
16
return str.substring(startPos, position);
13
char c = str.charAt(position);
14
!(delimiters.indexOf(c) >= 0)
(delimiters.indexOf(c) >= 0)
15
position++;

b) Node Coverage

[1]
[2]!
[3]
[4]
[5]!

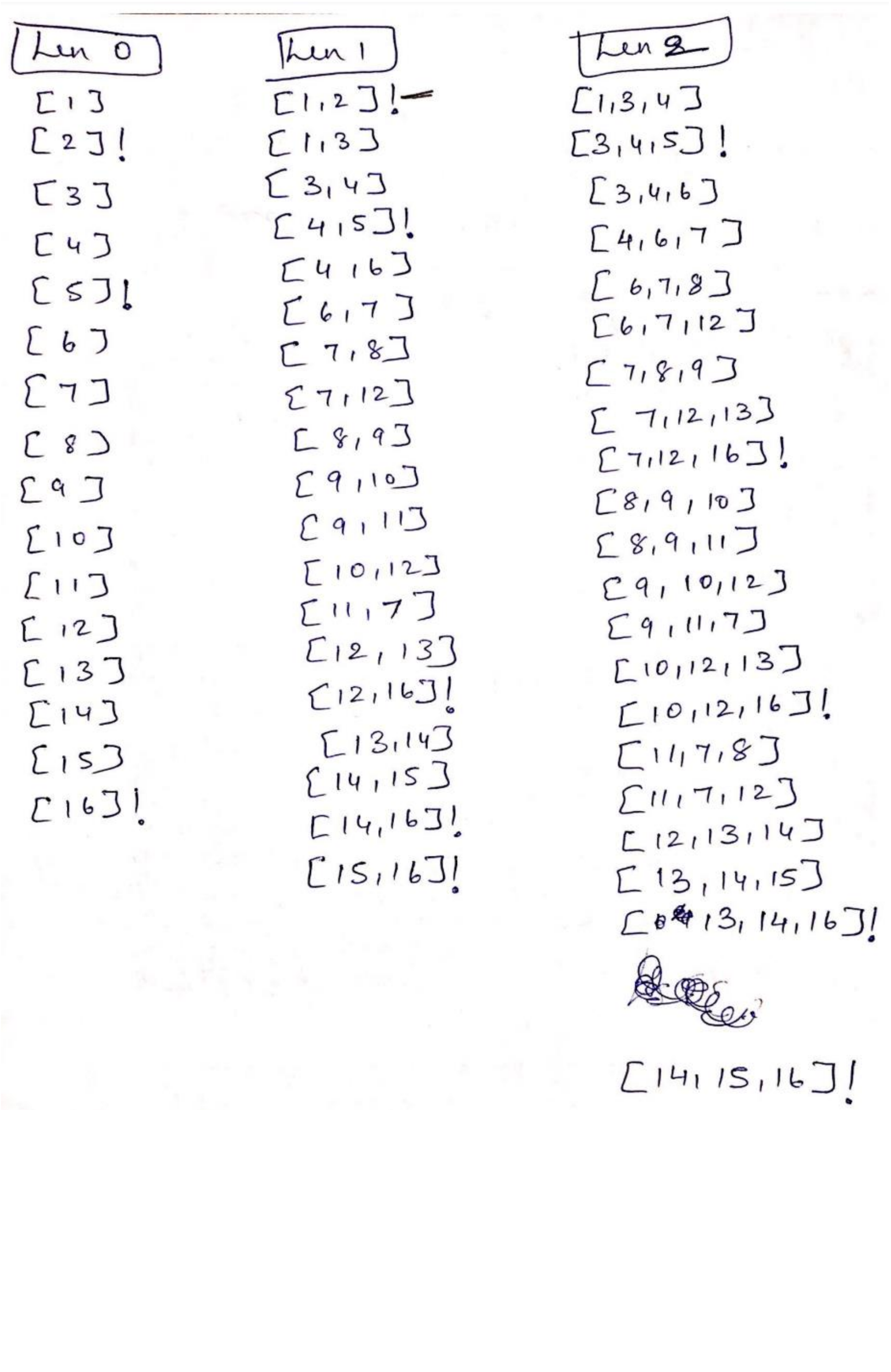
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
[16]!

b) Edge Coverage
[1, 2]!
[1, 3]
[3, 4]
[4, 5]!
[4, 6]
[6, 7]
[7, 8]
[7, 12]
[8, 9]
[9, 10]
[9, 11]
[10, 12]
[11, 7]
[12, 13]
[12, 16]!
[13, 14]
[14, 15]
[14, 16]!
[15, 16]!

c) Edge Pair Coverage
[1, 3, 4]
[3, 4, 5]!
[3, 4, 6]
[4, 6, 7]
[6, 7, 8]
[6, 7, 12]
[7, 8, 9]
[7, 12, 13]
[7, 12, 16]!
[8, 9, 10]
[8, 9, 11]
[9, 10, 12]
[9, 11, 7]
[10, 12, 13]
[10, 12, 16]!
[11, 7, 8]

[11, 7, 12]
[12, 13, 14]
[13, 14, 15]
[13, 14, 16]!
[14, 15, 16]!

d) Simple path



Len 0

[1]
[2]!
[3]
[4]
[5]!
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
[16]!

Len 1

[1,2]!
[1,3]
[3,4]
[4,5]!
[4,6]
[6,7]
[7,8]
[7,12]
[8,9]
[9,10]
[9,11]
[10,12]
[11,7]
[12,13]
[12,16]!
[13,14]
[14,15]
[14,16]!
[15,16]!

Len 2

[1,3,4]
[3,4,5]!
[3,4,6]
[4,6,7]
[6,7,8]
[6,7,12]
[7,8,9]
[7,12,13]
[7,12,16]!
[8,9,10]
[8,9,11]
[9,10,12]
[9,11,7]
[10,12,13]
[10,12,16]!
[11,7,8]
[11,7,12]
[12,13,14]
[13,14,15]
[13,14,16]!
[14,15,16]!

Len 3

[1,3,4,5]!—

[1,3,4,6]

[3,4,6,7]

[4,6,7,8]

[4,6,7,12]

[6,7,8,9]

[6,7,12,13]

[6,7,12,16]!

[7,8,9,10]

[7,8,9,11]

[7,12,13,14]

[8,9,10,12]

[8,9,11,7]

[9,10,12,13]

[9,10,12,16]!

[9,11,7,8]

[9,11,7,12]

[10,12,13,14]

[11,7,8,9]

[11,7,12,13]

[11,7,12,16]!

[12,13,14,15]

[12,13,14,16]!

[13,14,15,16]!

Len 4

[1, 3, 4, 6, 7]

[3, 4, 6, 7, 8]

[3, 4, 6, 7, 12]

[4, 6, 7, 8, 9]

[4, 6, 7, 12, 13]

[4, 6, 7, 12, 16]!

[6, 7, 8, 9, 10]

[6, 7, 8, 9, 11]

[6, 7, 12, 13, 14]

[7, 8, 9, 10, 12]

[7, 8, 9, 11, 7]*—

[7, 12, 13, 14, 15]

[8, 9, 10, 12, 13]

[8, 9, 10, 12, 16]!

[8, 9, 11, 7, 8]*—

[8, 9, 11, 7, 12]

[9, 10, 12, 13, 14]

[9, 11, 7, 8, 9]*—

[9, 11, 7, 12, 13]

[9, 11, 7, 12, 16]!*

[10, 12, 13, 14, 15]

[10, 12, 13, 14, 16]!

[11, 7, 8, 9, 11]*—

[11, 7, 8, 9, 10]

[11, 7, 12, 13, 14]

[12, 13, 14, 15, 16]!

Len 5

[1,3,4,6,7,8]

[1,3,4,6,7,12]

[3,4,6,7,8,9]

[3,4,6,7,12,13]

[3,4,6,7,12,16]!

[4,6,7,8,9,10]

[4,6,7,8,9,11]

[4,6,7,12,13,14]

[6,7,8,9,10,12]

~~[6,7,8,9,11,7]~~

[6,7,12,13,14,15]

[7,8,9,10,12,13]

[7,8,9,10,12,16]!

[7,12,13,14,15,16]!

[8,9,10,12,13,14]

[8,9,11,7,12,13]

[8,9,11,7,12,16]!

[9,10,12,13,14,15]

[9,11,7,12,13,14]

[10,12,13,14,15,16]!

[11,7,8,9,10,12]

[11,7,12,13,14,15]

[9,10,12,13,14,16]!

[11,7,12,13,14,16]!

[6,7,12,13,14,16]!

Len 6

[1,3,4,6,7,8,9]

[1,3,4,6,7,12,13]

[1,3,4,6,7,12,16]!—

[3,4,6,7,8,9,10]

[3,4,6,7,8,9,11]

[3,4,6,7,12,13,14]

[4,6,7,8,9,10,12]

~~[4,6,7,8,9,11,7]~~

[4,6,7,12,13,14,15]

[4,6,7,12,13,14,16]!

[6,7,8,9,10,12,13]

[6,7,8,9,10,12,16]!

[6,7,12,13,14,15,16]!

[7,8,9,10,12,13,14]

[8,9,10,12,13,14,16]!

[8,9,10,12,13,14,15]

[8,9,11,7,12,13,14]

[9,10,12,13,14,15,16]!

[9,11,7,12,13,14,15]

[9,11,7,12,13,14,16]!

[11,7,8,9,10,12,13]

[11,7,8,9,10,12,16]!—

[11,7,12,13,14,15;16]!

Len 7

[1,3,4,6,7,8,9,10]

[1,3,4,6,7,8,9,11]—

[1,3,4,6,7,12,13,14]

[3,4,6,7,8,9,10,12]

~~[3,4,6,7,8,9,11,~~

[3,4,6,7,12,13,14,15]

[3,4,6,7,12,13;14,16]!

[4,6,7,8,9,10,12,13]

[4,6,7,8,9,10,12,16]!

[4,6,7,12,13,14,15,16]!

[6,7,8,9,10,12,13,14]

[7,8,9,10,12,13,14,15]

[7,8,9,10,12,13,14,16]!

[8,9,10,12,13,14,15,16]!

[8,9,11,7,12,13,14,15]

[8,9,11,7,12,13,14,16]!—

[9,11,7,12,13,14,15,16]!

[11,7,8,9,10,12,13,14]

Len 8

[1, 3, 4, 6, 7, 8, 9, 10, 12]

~~[1, 3, 4, 6, 7, 8, 9, 11~~

[1, 3, 4, 6, 7, 12, 13, 14, 15]

[1, 3, 4, 6, 7, 12, 13, 14, 16]! —

[3, 4, 6, 7, 8, 9, 10, 12, 13]

[3, 4, 6, 7, 8, 9, 10, 12, 16]!

[3, 4, 6, 7, 12, 13, 14, 15, 16]!

[4, 6, 7, 8, 9, 10, 12, 13, 14]

[6, 7, 8, 9, 10, 12, 13, 14, 15]

[6, 7, 8, 9, 10, 12, 13, 14, 16]!

[7, 8, 9, 10, 12, 13, 14, 15, 16]!

[8, 9, 11, 7, 12, 13, 14, 15, 16]! —

[11, 7, 8, 9, 10, 12, 13, 14, 15]

[11, 7, 8, 9, 10, 12, 13, 14, 16]! —

Len 9

[1, 3, 4, 6, 7, 8, 9, 10, 12, 13]

[1, 3, 4, 6, 7, 8, 9, 10, 12, 16]! —

[1, 3, 4, 6, 7, 12, 13, 14, 15, 16]! —

[3, 4, 6, 7, 8, 9, 10, 12, 13, 14]

[4, 6, 7, 8, 9, 10, 12, 13, 14, 15]

[4, 6, 7, 8, 9, 10, 12, 13, 14, 16]!

[6, 7, 8, 9, 10, 12, 13, 14, 15, 16]!

[11, 7, 8, 9, 10, 12, 13, 14, 15, 16]! —

Len 10

[1,3,4,6,7,8,9,10,12,13,14]

[3,4,6,7,8,9,10,12,13,14,15]

[3,4,6,7,8,9,10,12,13,14,16]!

[4,6,7,8,9,10,12,13,14,15,16]!

Len 11

[1,3,4,6,7,8,9,10,12,13,14,15]

[1,3,4,6,7,8,9,10,12,13,14,16]! -

[3,4,6,7,8,9,10,12,13,14,15,16]!

Len 12

[1,3,4,6,7,8,9,10,12,13,14,15,16]! —

f) Prime paths

[1,3,4,6,7,8,9,10,12,13,14,15,16]!

[1,3,4,6,7,8,9,10,12,13,14,16]!

[1,3,4,6,7,8,9,10,12,16]!

[1,3,4,6,7,12,13,14,15,16]!

[11,7,8,9,10,12,13,14,15,16]!

[1,3,4,6,7,12,13,14,16]!

[8,9,11,7,12,13,14,15,16]!

[11,7,8,9,10,12,13,14,16]!

[1,3,4,6,7,8,9,11]

[8,9,11,7,12,13,14,16]!

[1,3,4,6,7,12,16]!

[11,7,8,9,10,12,16]!

[8,9,11,7,12,16]!

[7,8,9,11,7]*

[8,9,11,7,8]*

[9,11,7,8,9]*

~~[9,11,7,12,16~~

[11,7,8,9,11]*

[1,3,4,5]!

[1,2]!

9) PPC

	TP	TR
t_1	[1, 2]	[1, 2]
t_2	[1, 3, 4, 5]	[1, 3, 4, 5]
t_3	[1, 3, 4, 6, 7, 8, 9, 11, 7, 8, 9, 11, 7, 12, 16]	[7, 8, 9, 11, 7], [8, 9, 11, 7, 8], [9, 11, 7, 8, 9], [11, 7, 8, 9, 11], [8, 9, 11, 7, 12, 16], [1, 3, 4, 6, 7, 8, 9, 11]
t_4	[1, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16]	[1, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16]
t_5	[1, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 16]	[1, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 16]
t_6	[1, 3, 4, 6, 7, 8, 9, 10, 12, 16]	[1, 3, 4, 6, 7, 8, 9, 10, 12, 16]
t_7	[1, 3, 4, 6, 7, 12, 13, 14, 15, 16]	[1, 3, 4, 6, 7, 12, 13, 14, 15, 16]
t_8	[1, 3, 4, 6, 7, 12, 13, 14, 16]	[1, 3, 4, 6, 7, 12, 13, 14, 16]
t_9	[1, 3, 4, 6, 7, 8, 9, 11, 7, 8, 9, 10, 12, 13, 14, 15, 16]	[11, 7, 8, 9, 10, 12, 13, 14, 15, 16]
t_{10}	[1, 3, 4, 6, 7, 8, 9, 11, 7, 12, 13, 14, 15, 16]	[8, 9, 11, 7, 12, 13, 14, 15, 16]
t_{11}	[1, 3, 4, 6, 7, 8, 9, 11, 7, 8, 9, 10, 12, 13, 14, 16]	[11, 7, 8, 9, 10, 12, 13, 14, 16]
t_{12}	[1, 3, 4, 6, 7, 8, 9, 11, 7, 12, 13, 14, 16]	[8, 9, 11, 7, 12, 13, 14, 16]
t_{13}	[1, 3, 4, 6, 7, 12, 16]	[1, 3, 4, 6, 7, 12, 16]
t_{14}	[1, 3, 4, 6, 7, 8, 9, 11, 7, 8, 9, 10, 12, 16]	[11, 7, 8, 9, 10, 12, 16]

3. Logic Coverage Criteria [25 pts]

a. Identify all predicates from the **scanToken** function and clearly label each clause and its corresponding code

b. **For each predicate you identified in the previous answer**, complete the items below:

   i. Compute (and simplify) the conditions under which each clause determines the predicate. Be sure to include details or steps showing how you compute and simplify.

   ii. Write the complete truth table for each clause. Label your rows starting from 1.

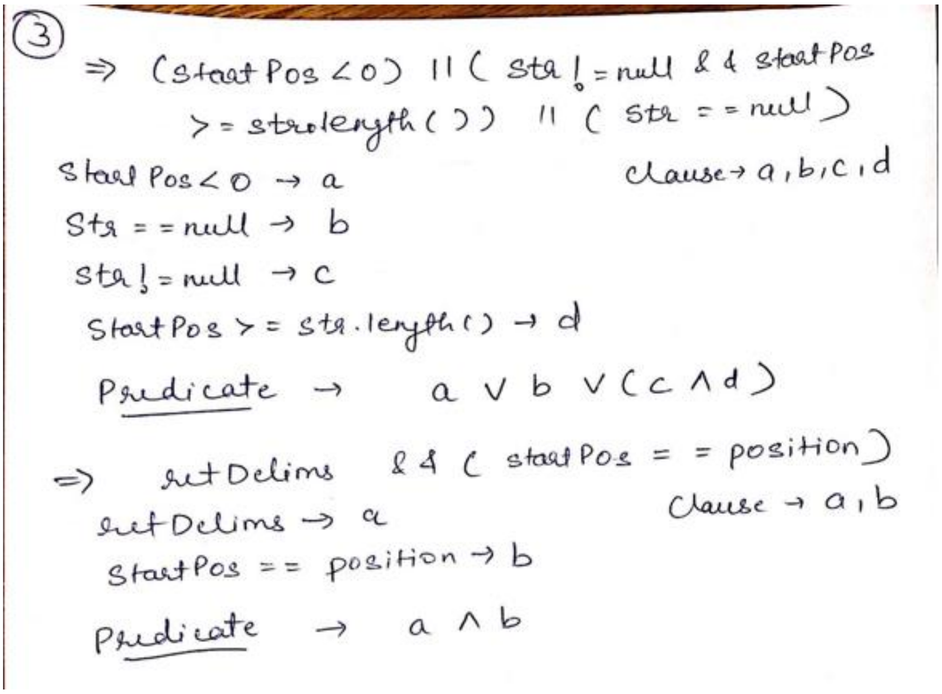
   iii. Use the format in the examples we covered in lecture. That is, row 1 should be all clauses true. You should include columns for the conditions under which each clause determines the predicate, and also a column for the value of the predicate itself.

   iv. Give a list of pairs of rows from your table that satisfy Clause Coverage (CC) but does not satisfy Predicate Coverage (PC).

   v. List all pairs of rows from your table that satisfy Correlated Active Clause Coverage (CACC) with respect to each clause.

Solution:

a)



③

⇒ (start Pos < 0) || (str != null && start Pos >= str.length( )) || (str == null)

Start Pos < 0 → a

clause → a, b, c, d

Str == null → b

str != null → c

Start Pos >= str.length( ) → d

Predicate → $a \vee b \vee (c \wedge d)$

⇒ retDelims && (start Pos == position)

Clause → a, b

retDelims → a

StartPos == position → b

Predicate → $a \wedge b$

Predicate 1

b) Predicate $a \vee b \vee (c \wedge d)$

i) $\rightarrow P_a \neq P_{a=true} \oplus P_{a=false}$

$= (true \vee b \vee (c \wedge d)) \oplus (false \vee b \vee (c \wedge d))$

$= (true \vee (c \wedge d)) \oplus (b \vee (c \wedge d))$ (∵ Using OR identity laws)

$= true \oplus (b \vee (c \wedge d))$ (∵ Using OR identity laws)

$= \neg(b \vee (c \wedge d))$ (∵ Using XOR identity laws)

$= \neg b \wedge \neg(c \wedge d)$ (∵ Using Demorgan's law)

∴ $\boxed{P_a = \neg b \wedge (\neg c \vee \neg d)}$ (∵ using demorgan's law)

$\rightarrow P_b = P_{b=true} \oplus P_{b=false}$

$= (a \vee true \vee (c \wedge d)) \oplus (a \vee false \vee (c \wedge d))$

$= (true \vee (c \wedge d)) \oplus (a \vee (c \wedge d))$ (∵ OR identity ~~rules~~ laws)

$= true \oplus (a \vee (c \wedge d))$ (∵ OR identity ~~rules~~ laws)

$= \neg(a \vee (c \wedge d))$ (∵ XOR identity laws)

$= \neg a \wedge \neg(c \wedge d)$ (∵ Demorgan's law)

$\therefore \boxed{P_b = \neg a \wedge (\neg c \vee \neg d)}$

(∵ Demorgan's law)

$\rightarrow P_c = P_c = \text{true} \oplus P_c = \text{false}$

$= (a \vee b \vee (\text{true} \wedge d)) \oplus (a \vee b \vee (\text{false} \wedge d))$

$= (a \vee b \vee d) \oplus (a \vee b \vee \text{false})$

(∵ AND identity laws)

$= ((a \vee b) \vee d) \oplus (a \vee b)$

(∵ OR identity laws)

$= \neg (a \vee b) \wedge d$ (∵ XOR identity laws)

$\therefore \boxed{P_c = \neg a \wedge \neg b \wedge d}$ (∵ Demorgan's law)

$\rightarrow P_d = P_d = \text{true} \oplus P_d = \text{false}$

$= (a \vee b \vee (c \wedge \text{true})) \oplus (a \vee b \vee (c \wedge \text{false}))$

$= (a \vee b \vee c) \oplus (a \vee b \vee \text{false})$

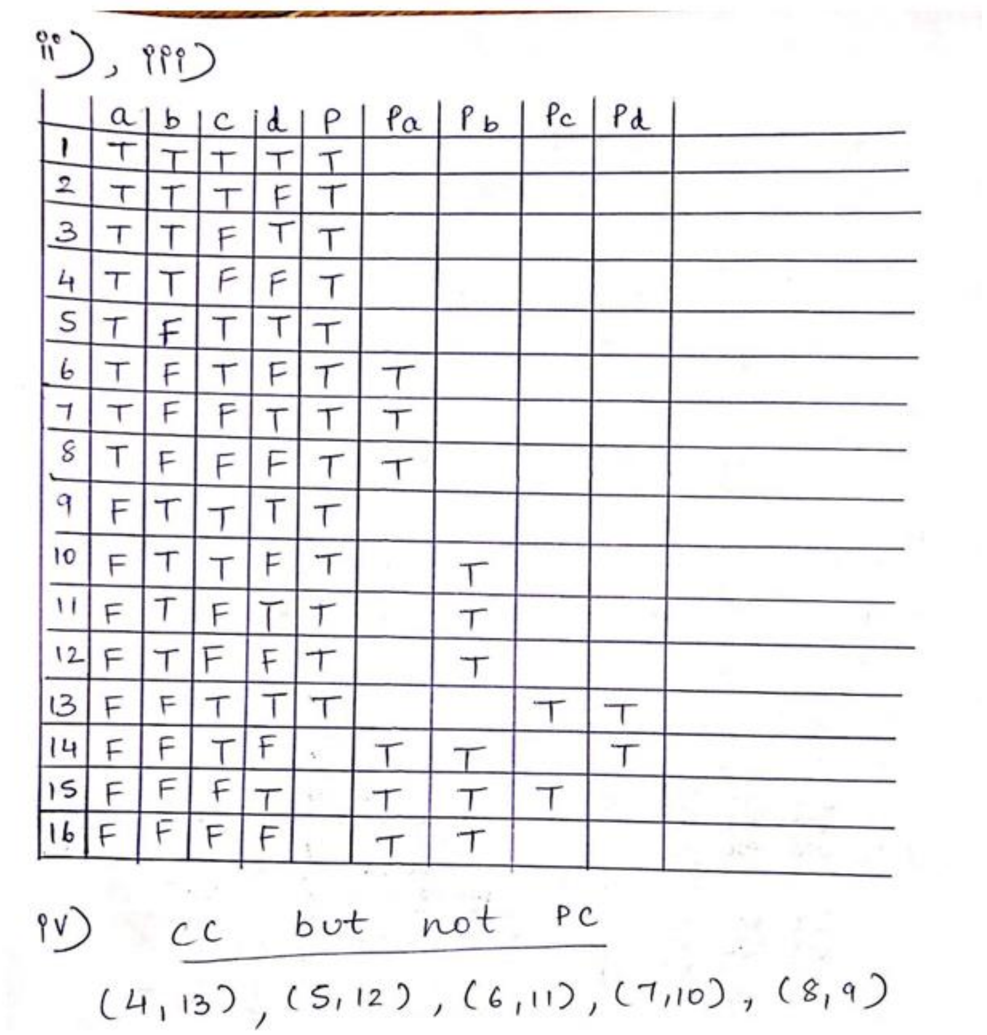
(∵ AND identity laws)

$= ((a \vee b) \vee c) \oplus (a \vee b)$

(∵ OR identity laws)

$= \neg (a \vee b) \wedge c$ (∵ XOR identity laws)

$\therefore \boxed{P_d = \neg a \wedge \neg b \wedge c}$ (∵ demorgan's law)

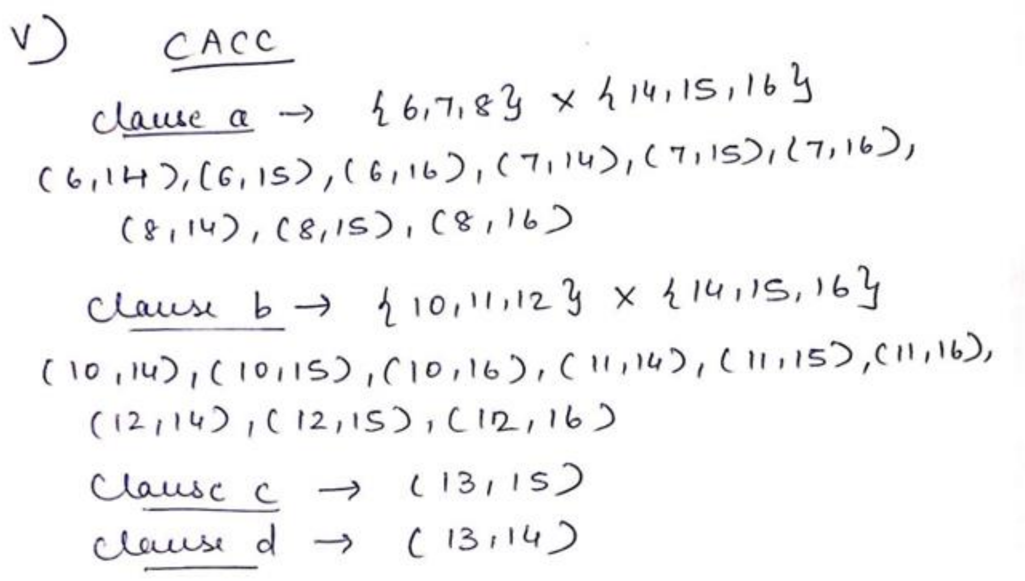


ii), iii)

	a	b	c	d	P	Pa	Pb	Pc	Pd
1	T	T	T	T	T				
2	T	T	T	F	T				
3	T	T	F	T	T				
4	T	T	F	F	T				
5	T	F	T	T	T				
6	T	F	T	F	T	T			
7	T	F	F	T	T	T			
8	T	F	F	F	T	T			
9	F	T	T	T	T				
10	F	T	T	F	T		T		
11	F	T	F	T	T		T		
12	F	T	F	F	T		T		
13	F	F	T	T	T			T	T
14	F	F	T	F		T	T		T
15	F	F	F	T		T	T	T	
16	F	F	F	F		T	T		

iv) CC but not PC

(4, 13), (5, 12), (6, 11), (7, 10), (8, 9)



v) CACC

clause a → {6, 7, 8} × {14, 15, 16}

(6, 14), (6, 15), (6, 16), (7, 14), (7, 15), (7, 16),
(8, 14), (8, 15), (8, 16)

clause b → {10, 11, 12} × {14, 15, 16}

(10, 14), (10, 15), (10, 16), (11, 14), (11, 15), (11, 16),
(12, 14), (12, 15), (12, 16)

Clause c → (13, 15)

clause d → (13, 14)

Predicate 2

b) Predicate $a \wedge b$

i) → $P_a = P_{a=true} \oplus P_{a=False}$

$= (true \wedge b) \oplus (false \wedge b)$

$= b \oplus false$ (∵ Using AND identity laws)

∴ $\boxed{P_a = b}$ (∵ XOR identity laws)

→ $P_b = P_{b=true} \oplus P_{b=false}$

$= (a \wedge true) \oplus (b \wedge false)$

$= a \oplus false$ (∵ AND identity laws)

∴ $\boxed{P_b = a}$ (∵ XOR identity laws)

ii), iii)

	a	b	P	Pa	Pb
1	T	T	T	T	T
2	T	F			T
3	F	T		T	
4	F	F			

iv) CC but not PC

(2, 3)

v) CACC

clause a → (1, 3)

clause b → (1, 2)

4. Syntactic Logic Coverage Criteria [15 pts]

Use the predicate f below to answer the following questions.

$$f = \bar{a}bd + abd + \bar{b}\bar{c}\bar{d} + \bar{b}c\bar{d}$$

a) Draw Karnaugh maps for f and $\bar{f}$

b) Find the DNF non-redundant prime implicant representation for f and $\bar{f}$

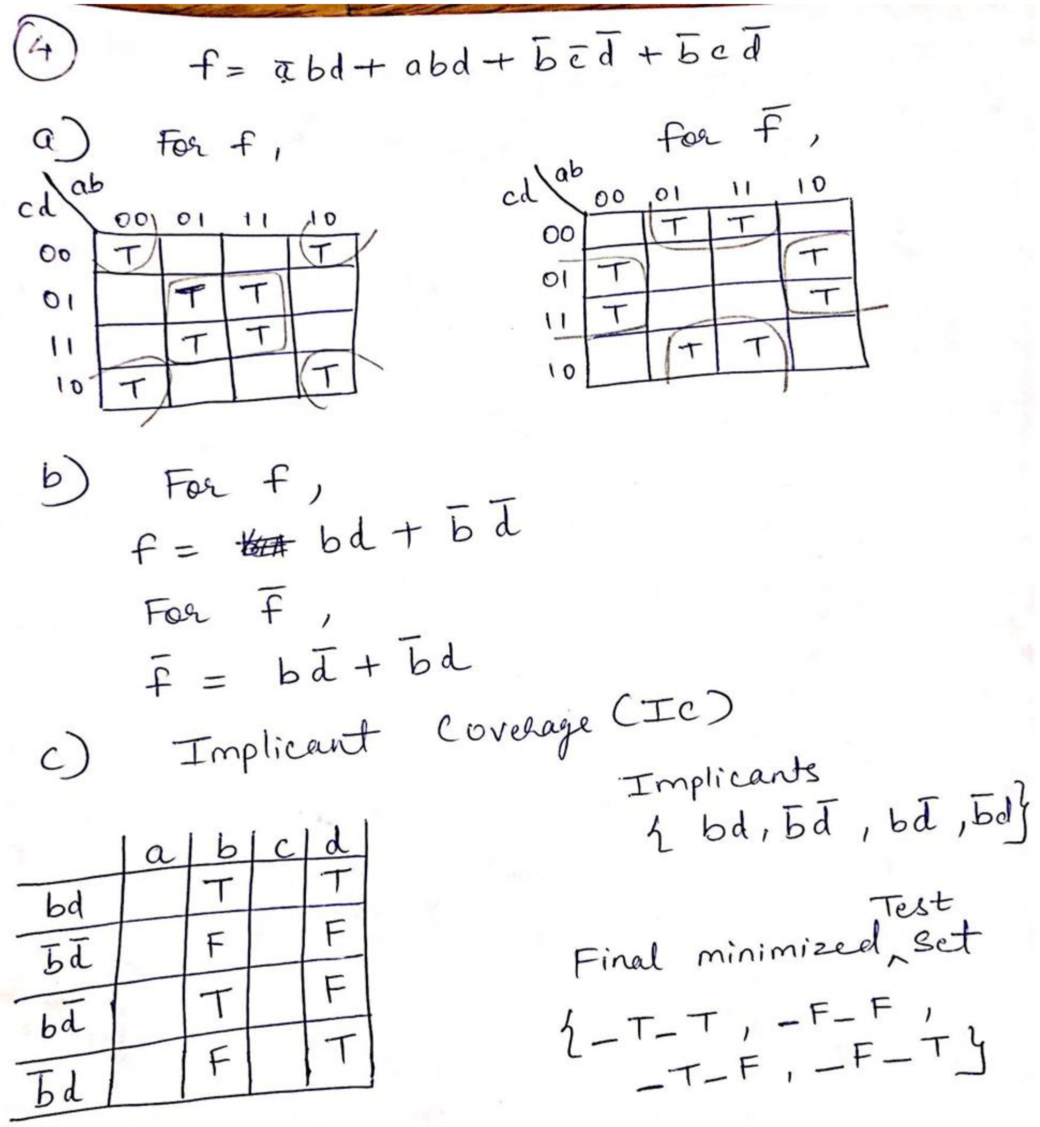
c) Give a test set that satisfies Implicant Coverage (IC) for f

d) Give a test set that satisfies Multiple Unique True Point (MUTP) for f

e) Give a test set that satisfies Corresponding Unique True Point and Near False Point Pair Coverage (CUTPNFP) for f

f) Give a test set that satisfies Multiple Near False Points (MNFP) for f

Solution:



4

$f = \bar{a}bd + abd + \bar{b}\bar{c}\bar{d} + \bar{b}c\bar{d}$

a) For f,

cd \ ab	00	01	11	10
00	T			T
01		T	T	
11		T	T	
10	T			T

for $\bar{f}$,

cd \ ab	00	01	11	10
00		T	T	
01	T			T
11	T			T
10		T	T	

b) For f,

$f = bd + \bar{b}\bar{d}$

For $\bar{f}$,

$\bar{f} = b\bar{d} + \bar{b}d$

c) Implicant Coverage (IC)

Implicants

$\{bd, \bar{b}\bar{d}, b\bar{d}, \bar{b}d\}$

	a	b	c	d
bd		T		T
$\bar{b}\bar{d}$		F		F
$b\bar{d}$		T		F
$\bar{b}d$		F		T

Final minimized Test set

{_T_T , _F_F , _T_F , _F_T}

d) MUTP

Test set: { TTFT , FTTT , TFFF , FFTF }

e) CUTPNFP

For implicant bd ,

	UTP	NFP
b	FTFT	FFFT
d	FTFT	FTFF

For implicant $\bar{b}\bar{d}$,

	UTP	NFP
$\bar{b}$	TFFF	TTFF
$\bar{d}$	TFFF	TFFT

Possible CUTPNFP Test Set

{ FTFT, TFFF } → UTPs

{ FFFT, FTFF, TTFF, TFFT } → NFPs

f) MNFP

For implicant bd ,

NFPs
- b → FFFT , TFTT
- d → FTFF , TTTF

For implicant $\bar{b}\bar{d}$,

NFPs
- $\bar{b}$ → TTFF, FTTF
- $\bar{d}$ → TFFT , FFTT

Test set

{ TTFF, TFFT, FTTF, FFTT , FFFT , FTFF, TFTT, TTTF }

5. Syntax Coverage [10 pts]

Using the following BNF grammar:

```
EXPRESSION ::=  NUMERAL  |  "(" EXPRESSION OPERATOR EXPRESSION ")"
OPERATOR ::=  " + "  |  " - "
NUMERAL ::=  DIGIT  |  DIGIT NUMERAL
DIGIT ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

a. Give a single test case that satisfies Production Coverage (PDC) for this grammar
b. Give an example of a mutant production operator and give an example of an invalid mutant that is produced
c. Give an example of a mutant terminal operator and give an example of an invalid mutant that is produced
d. Give an example of a mutant operator that can produce both a valid mutant and an invalid mutant and show both resulting mutants (can be same operator from part “b” or “c”)

Solution:

a) `( 7 + 34 )`

b) `( 91 – 4 )`

```
    Invalid Mutant
    ( - 91 4 )
```

c) `5`

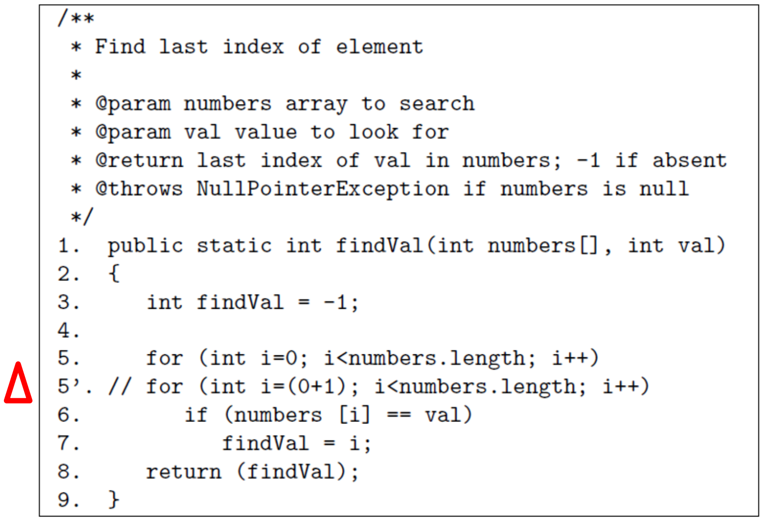
```
    Invalid Mutant
     0+
```

d) `( 91 – 4 )`

```
    Valid mutant
    ( 91 + 4 )
    Invalid mutant
    ( 91 % 4 )
```

6. Syntax Coverage [5 pts] (Extra Credit)

Answer questions (a) through (d) for the mutant on line 5 in the method **`findVal()`**.



```
/**
 * Find last index of element
 *
 * @param numbers array to search
 * @param val value to look for
 * @return last index of val in numbers; -1 if absent
 * @throws NullPointerException if numbers is null
 */
1.  public static int findVal(int numbers[], int val)
2.  {
3.     int findVal = -1;
4.
5.     for (int i=0; i<numbers.length; i++)
Δ 5'. // for (int i=(0+1); i<numbers.length; i++)
6.        if (numbers [i] == val)
7.           findVal = i;
8.     return (findVal);
9.  }
```

a) If possible find test inputs that do not reach the mutant.
b) If possible, find test inputs that satisfy reachability but not infection for the mutant.
c) If possible, find test inputs that satisfy reachability and infection, but not propagation for the mutant.
d) If possible, find test inputs that **strongly** kill the mutants.

Solution:

a) The mutant is always reached, even if x = null.

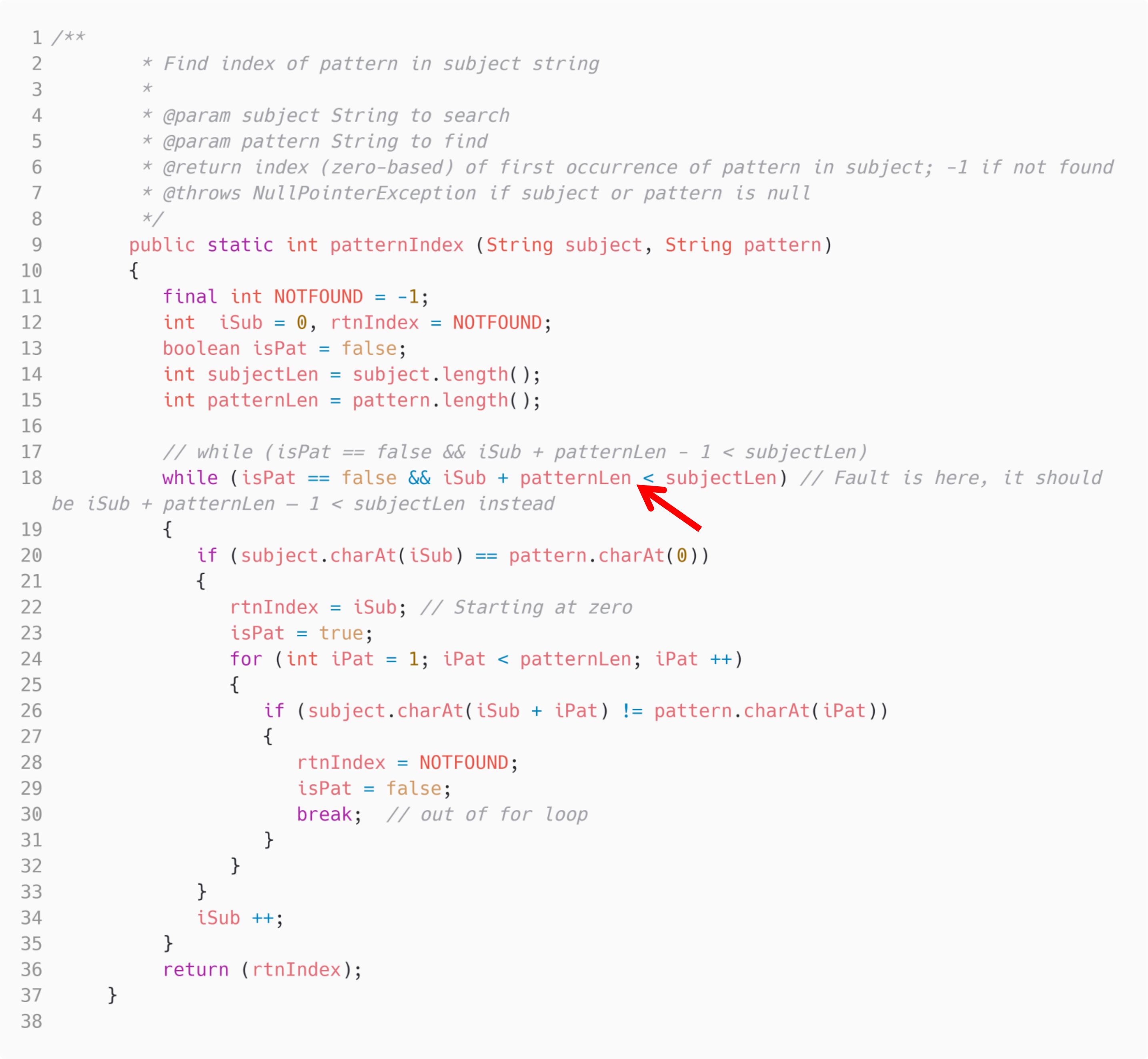
b) Infection always occurs, even if x = null, because i always has the wrong value after initialization in the loop.

c) As long as the last occurrence of val isn't at numbers[0], the correct output is returned. i.e. (numbers, val) = ([1, 1], 1) or ([-1,1], 1) or (null,0)

d) Any input with val only in numbers[0] works. i.e. (numbers, val) = ([1, 0], 1)

7. RIPR [10 pts] (Extra Credit)

By considering the method below, answer the following questions `a` through `d`. Also, if such a test is not possible for any of the answers, explain why?



```java
/**
     * Find index of pattern in subject string
     *
     * @param subject String to search
     * @param pattern String to find
     * @return index (zero-based) of first occurrence of pattern in subject; -1 if not found
     * @throws NullPointerException if subject or pattern is null
     */
    public static int patternIndex (String subject, String pattern)
    {
      final int NOTFOUND = -1;
      int  iSub = 0, rtnIndex = NOTFOUND;
      boolean isPat = false;
      int subjectLen = subject.length();
      int patternLen = pattern.length();

      // while (isPat == false && iSub + patternLen - 1 < subjectLen)
      while (isPat == false && iSub + patternLen < subjectLen) // Fault is here, it should
be iSub + patternLen – 1 < subjectLen instead
      {
        if (subject.charAt(iSub) == pattern.charAt(0))
        {
          rtnIndex = iSub; // Starting at zero
          isPat = true;
          for (int iPat = 1; iPat < patternLen; iPat ++)
          {
            if (subject.charAt(iSub + iPat) != pattern.charAt(iPat))
            {
              rtnIndex = NOTFOUND;
              isPat = false;
              break;  // out of for loop
            }
          }
        }
        iSub ++;
      }
      return (rtnIndex);
  }
```

a) If possible, give a test case that does not reach the fault. Explain your answer.
b) If possible, give a test case that reaches the fault, but does not infect. Explain your answer.
c) If possible, give a test case that infects the state, but does not propagate. Explain your answer.
d) If possible, give a test case that propagates. Explain your answer.

Solution:

a) subject == null or pattern == null.

Example(subject, pattern): (“A", null)

b) The pattern needs to be longer than the subject.

Example (subject, pattern): (“x”, “xyz”)

c) The subject must be as long as or longer than the pattern, and the pattern is not in the subject.

Example (subject, pattern): (“cat”, “car”)

d) subject == pattern

Example (subject, pattern): (“B”, “B”)