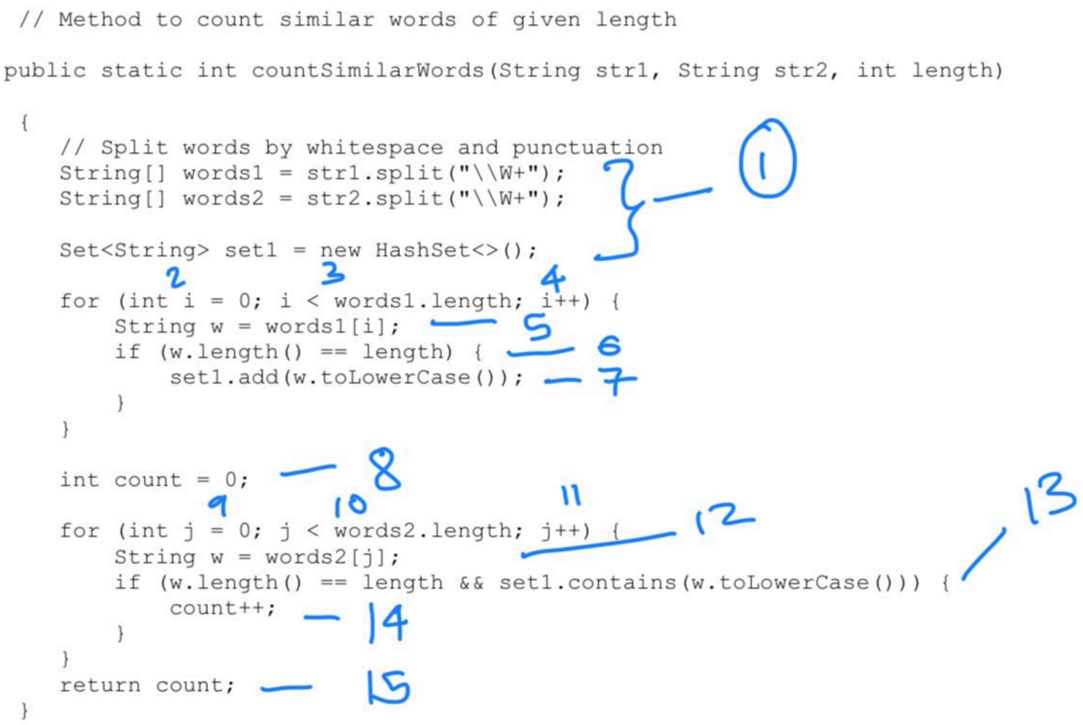
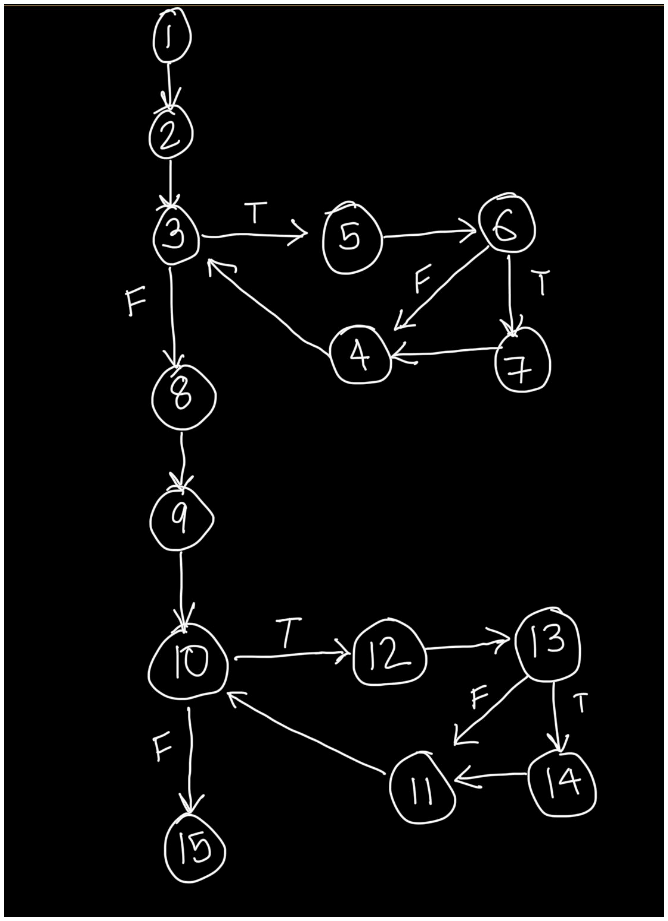
HW 4 - CSE 464
Bhavya Patel - ASU ID: 1225740997

Q1. a)



```
 // Method to count similar words of given length

public static int countSimilarWords(String str1, String str2, int length)

 {
    // Split words by whitespace and punctuation
    String[] words1 = str1.split("\\W+");
    String[] words2 = str2.split("\\W+");

    Set<String> set1 = new HashSet<>();

    for (int i = 0; i < words1.length; i++) {
        String w = words1[i];
        if (w.length() == length) {
            set1.add(w.toLowerCase());
        }
    }

    int count = 0;

    for (int j = 0; j < words2.length; j++) {
        String w = words2[j];
        if (w.length() == length && set1.contains(w.toLowerCase())) {
            count++;
        }
    }
    return count;
 }
```



Independent Paths:

- Path 1: 1 → 2 → 3 → 8 → 9 → 10 → 15
- Path 2: 1 → 2 → 3 → 5 → 4 → 3 → 8 → 9 → 10 → 15
- Path 3: 1 → 2 → 3 → 5 → 6 → 7 → 4 → 3 → 8 → 9 → 10 → 12 → 13 → 11 → 10 → 15
- Path 4: 1 → 2 → 3 → 5 → 6 → 7 → 4 → 3 → 8 → 9 → 10 → 12 → 13 → 14 → 10 → 15
- Path 5: 1 → 2 → 3 → 5 → 6 → 4 → 3 → 8 → 9 → 10 → 12 → 13 → 11 → 10 → 15

Test Cases:

Path	str1	str2	length	Expected
1	""	""	3	0
2	"abc,def"	""	4	0
3	"cat bat"	"dog bat"	3	1
4	"pen dog"	"dog pen dog"	3	3
5	"one two"	"three four"	3	0

b)

```
import static org.junit.Assert.*;
import org.junit.Test;
public class partb {
    @Test
    public void path1_bothLoopsSkipped() {
        assertEquals(0, countSimilarWords.countSimilarWords("", "",
3));
    }
    @Test
    public void path2_firstLoopRunsNoAdds() {
        assertEquals(0, countSimilarWords.countSimilarWords("abc,def",
"", 4));
    }
    @Test
    public void path3_someSetEntriesNoCount() {
        assertEquals(1, countSimilarWords.countSimilarWords("cat bat",
"dog bat", 3));
    }
    @Test
    public void path4_positiveFullCount() {
        assertEquals(3, countSimilarWords.countSimilarWords("pen dog",
"dog pen dog", 3));
    }
    @Test
    public void path5_secondLoopNoCompoundTrue() {
        assertEquals(0, countSimilarWords.countSimilarWords("one two",
"three four", 3));
    }
}
```

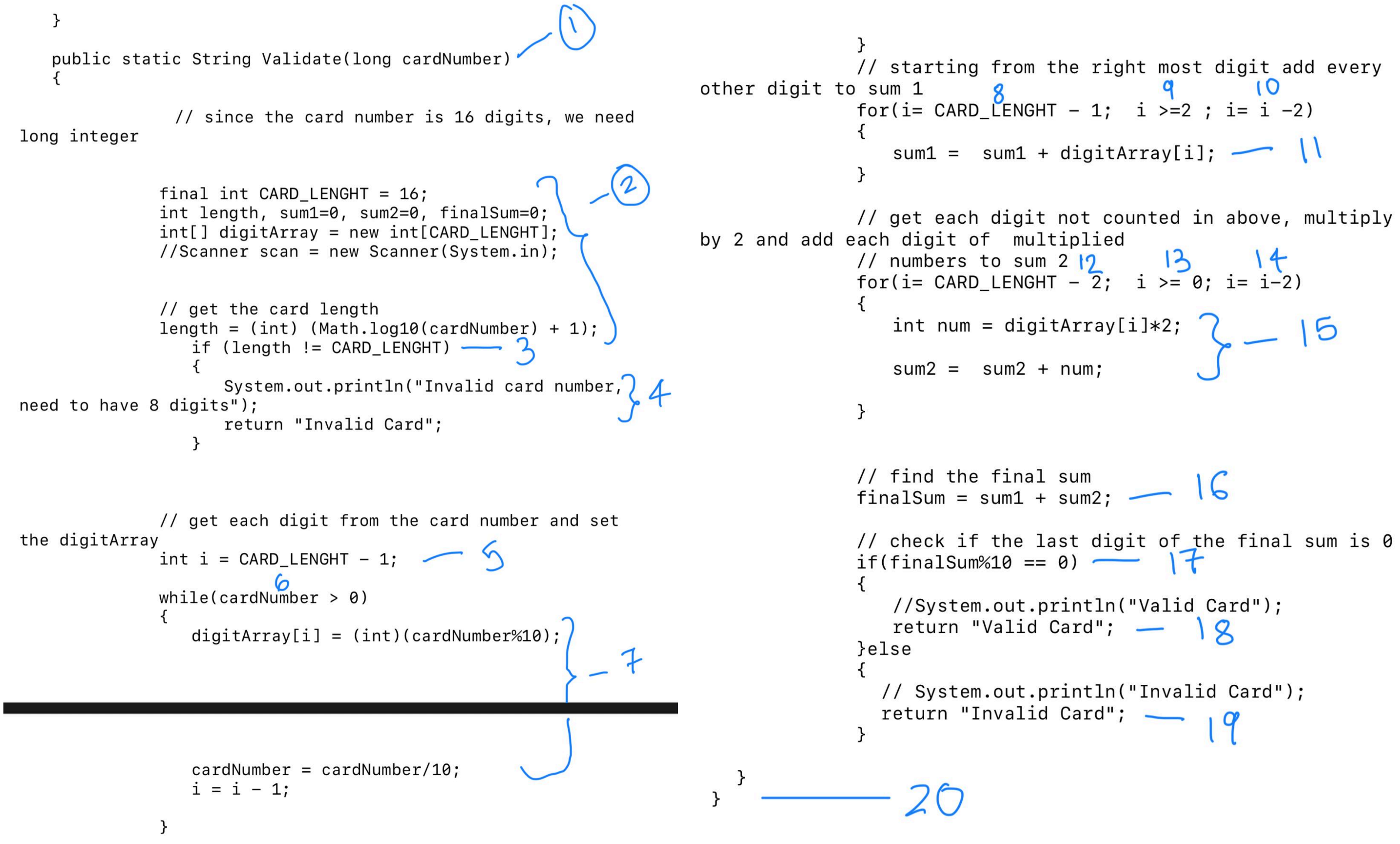
https://drive.google.com/file/d/1Enav5vFhqu1htBWX_jOqPqosQXGNNTLx/view?usp=drive_link

c)

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import java.util.stream.Stream;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.Arguments;
public class partc {
    static Stream<Arguments> cases() {
        return Stream.of(
                Arguments.of("", "", 3, 0),
                Arguments.of("abc,def", "", 4, 0),
                Arguments.of("cat bat", "dog bat", 3, 1),
                Arguments.of("pen dog", "dog pen dog", 3, 3),
                Arguments.of("one two", "three four", 3, 0)
        );
    }

    @ParameterizedTest
    @MethodSource("cases")

    void testAllPaths(String str1, String str2, int length, int
expected) {

        assertEquals(expected,
countSimilarWords.countSimilarWords(str1, str2, length));

    }

}
```

https://drive.google.com/file/d/1SvS6veIp2RZvLI9-A6AGSSWgb0RaAgsk/view?usp=drive_link

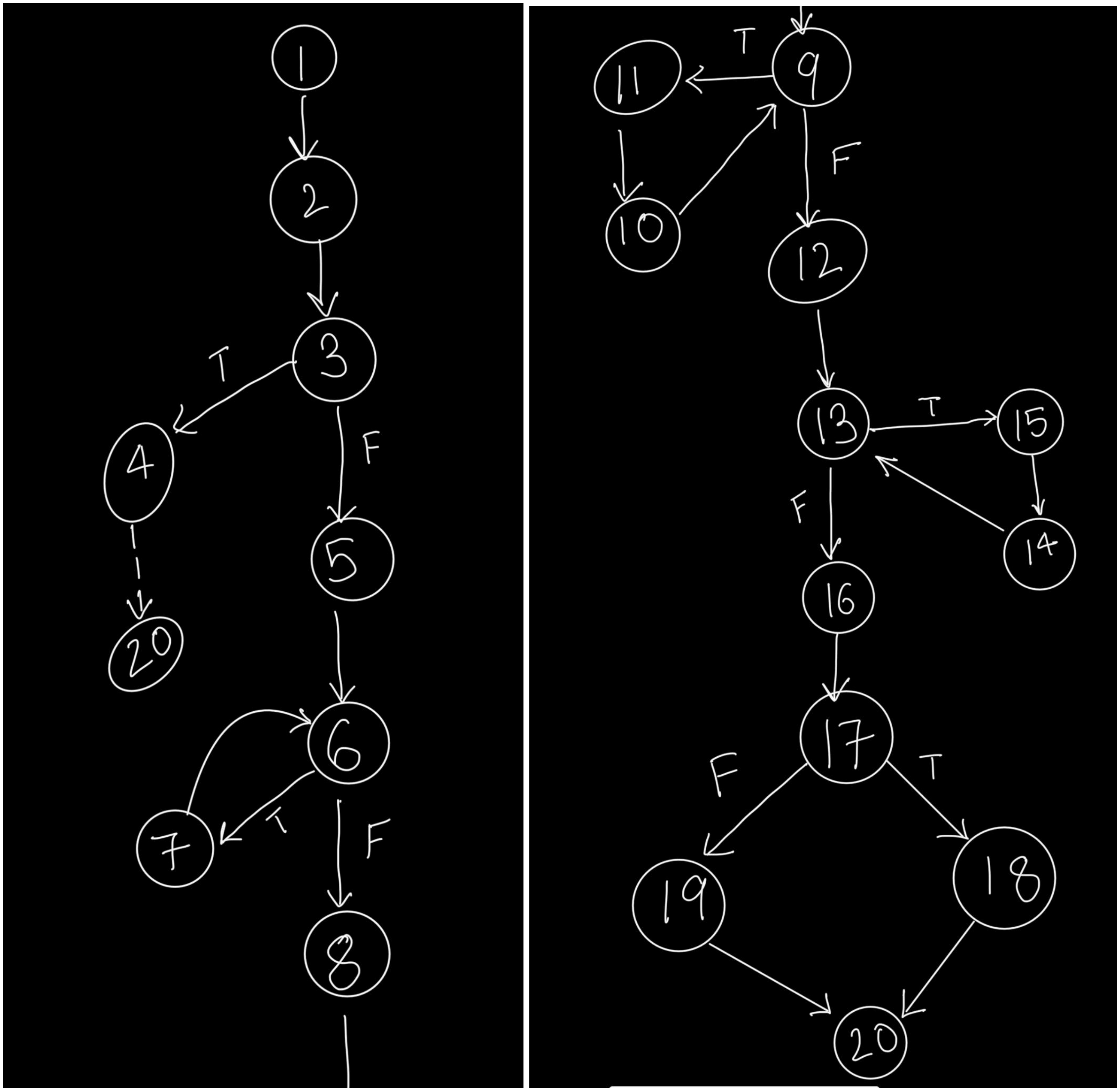
Q2 a)



```
    }

    public static String Validate(long cardNumber) - 1
    {

                // since the card number is 16 digits, we need
long integer


              final int CARD_LENGHT = 16;                          } - 2
              int length, sum1=0, sum2=0, finalSum=0;
              int[] digitArray = new int[CARD_LENGHT];
              //Scanner scan = new Scanner(System.in);


              // get the card length
              length = (int) (Math.log10(cardNumber) + 1);
                 if (length != CARD_LENGHT) - 3
                 {
                     System.out.println("Invalid card number,  } 4
need to have 8 digits");
                     return "Invalid Card";
                 }


              // get each digit from the card number and set
the digitArray
              int i = CARD_LENGHT - 1; - 5
                          6
              while(cardNumber > 0)
              {
                 digitArray[i] = (int)(cardNumber%10);    } - 7


                 cardNumber = cardNumber/10;
                 i = i - 1;

              }
```

```
              }
              // starting from the right most digit add every
other digit to sum 1   8                 9           10
              for(i= CARD_LENGHT - 1;  i >=2 ; i= i -2)
              {
                 sum1 =  sum1 + digitArray[i]; - 11
              }

              // get each digit not counted in above, multiply
by 2 and add each digit of  multiplied
              // numbers to sum 2 12     13       14
              for(i= CARD_LENGHT - 2;  i >= 0; i= i-2)
              {
                 int num = digitArray[i]*2;   } - 15

                 sum2 =  sum2 + num;

              }


              // find the final sum
              finalSum = sum1 + sum2; - 16

              // check if the last digit of the final sum is 0
              if(finalSum%10 == 0) - 17
              {
                 //System.out.println("Valid Card");
                 return "Valid Card"; - 18
              }else
              {
                // System.out.println("Invalid Card");
                return "Invalid Card"; - 19
              }

          }
      } - 20
```

Control Flow Graph:



Independent Paths:

- Path 1: 1→ 2→ 3→ 4→ 20
- Path 2: 1→ 2→ 3→ 5→ 6→ 8→ 9→ 12→ 13→ 16→ 17→ 19→ 20
- Path 3: 1→ 2→ 3→ 5→ 6→ 8→ 9→ 12→ 13→ 16→ 17→ 18→ 20
- Path 4: 1→ 2→ 3→ 5→ 6→ 8→ 9→ 12→ 13→ 15→ 14→ 13→ 16→ 17→ 19→ 20
- Path 5: 1→ 2→ 3→ 5→ 6→ 8→ 9→ 11→ 10→ 9→ 12→ 13→ 16→ 17→ 19→ 20
- Path 6: 1→ 2→ 3→ 5→ 6→ 7→ 6→ 8→ 9→ 12→ 13→ 16→ 17→ 19→ 20

Test Cases:

Path	cardNumber	Expected
1	1234567890	"Invalid Card"
2	1111222233334444	"Invalid Card"
3	1111222233334445	"Valid Card"

Paths 4, 5, and 6 are infeasible because they create a logical contradiction. To even start down these paths, a test case must use a 16-digit number to pass the first check (Node 3). However, these specific paths are defined by skipping the subsequent loops (Nodes 6, 9, or 13). This is impossible. Any 16-digit number must execute the while loop and both for loops because the logic is hard-coded for a 16-digit length. Since no 16-digit input can ever skip these loops, no test case can exist for these contradictory paths.

b)

```
import static org.junit.Assert.*;
import org.junit.Test;

public class partb {
    @Test
    public void p1() {
        assertEquals("Invalid Card",
CreditCardValidation.Validate(1234567890L));
    }
    @Test
    public void p2() {
        assertEquals("Invalid Card",
CreditCardValidation.Validate(1111222233334444L));
    }
    @Test
    public void p3() {
        assertEquals("Valid Card",
CreditCardValidation.Validate(1111222233334445L));
    }
}
```

https://drive.google.com/file/d/1hFukLuvqhz8mQCmOq8OpkBPX8OAfOQHQ/view?usp=sharing c)

- Bug 1: The for loop that calculates sum1 has an incorrect termination condition. The code i >= 2 causes the loop to stop at index 3, incorrectly omitting the digit at index 1 from the sum.
- Bug 2: The for loop for sum2 correctly doubles each digit, but it adds that resulting number (e.g., 18) directly to sum2. The algorithm requires adding the sum of the digits of the doubled number (e.g., $1 + 8 = 9$).

```
import java.util.Scanner;
public class CreditCardValidation {
    public static void main(String[] args) {
        System.out.println("Enter the 16 digit card number: ");
        Scanner scan = new Scanner(System.in);
        long cardNumber = scan.nextLong();

        String valid_invlaid = Validate(cardNumber);
        System.out.println(valid_invlaid);
    }
    public static String Validate(long cardNumber) {
        // since the card number is 16 digits, we need long integer
        final int CARD_LENGHT = 16;
        int length, sum1 = 0, sum2 = 0, finalSum = 0;
        int[] digitArray = new int[CARD_LENGHT];
        //Scanner scan = new Scanner(System.in);
        // get the card length
        length = (int) (Math.log10(cardNumber) + 1);
        if (length != CARD_LENGHT) {
            System.out.println("Invalid card number, need to have 8
digits");
            return "Invalid Card";
        }
        // get each digit from the card number and set the digitArray
        int i = CARD_LENGHT - 1;
        while (cardNumber > 0) {
            digitArray[i] = (int) (cardNumber % 10);

            cardNumber = cardNumber / 10;

            i = i - 1;

        }

        // starting from the right most digit add every other digit to 
sum 1

        // BUG 1 FIX:

        for (i = CARD_LENGHT - 1; i >= 0; i = i - 2) {

            sum1 = sum1 + digitArray[i];

        }

        // get each digit not counted in above, multiply by 2 and add 
each digit of  multiplied

        // numbers to sum 2

        for (i = CARD_LENGHT - 2; i >= 0; i = i - 2) {

            int num = digitArray[i] * 2;

            // BUG 2 FIX:

            sum2 = sum2 + (num / 10) + (num % 10);

        }

        // find the final sum

        finalSum = sum1 + sum2;

        // check if the last digit of the final sum is 0

        if (finalSum % 10 == 0) {

            //System.out.println("Valid Card");

            return "Valid Card";

        } else {

            // System.out.println("Invalid Card");

            return "Invalid Card";

        }

    }

}
```

https://drive.google.com/file/d/15siWkUC1AFGs18xz-ITjwSZKtQUzWcpZ/view?usp=drive_link