Software Testing Project Report

Meghna Dubey (MT2020017)

Chandrika Bhuyan (MT2020056)

Nupur Banerjee (MT2020103)

About Source Code

- The project consists of functions covering different topics, such as Maths, Searching, Sorting, etc.
- It is a Java based console application that is divided into 5 packages as following:
    - Maths
        - GCD
        - Area
        - PythagoreanTriple
        - MagicSquare
        - PrimeFactorization
    - Conversion
        - BinaryToDecimal
        - AnyToAny
        - HexToOct
    - Search
        - SearchBinary
        - SearchLinear
    - Sort
        - BubbleSort
        - CountingSort
        - InsertionSort
    - Misc
        - LeapYear
        - NthUglyNumber

Testing Strategy

Control Flow Graphs:

- Prime Path coverage
- Edge coverage
- Edge Pair coverage

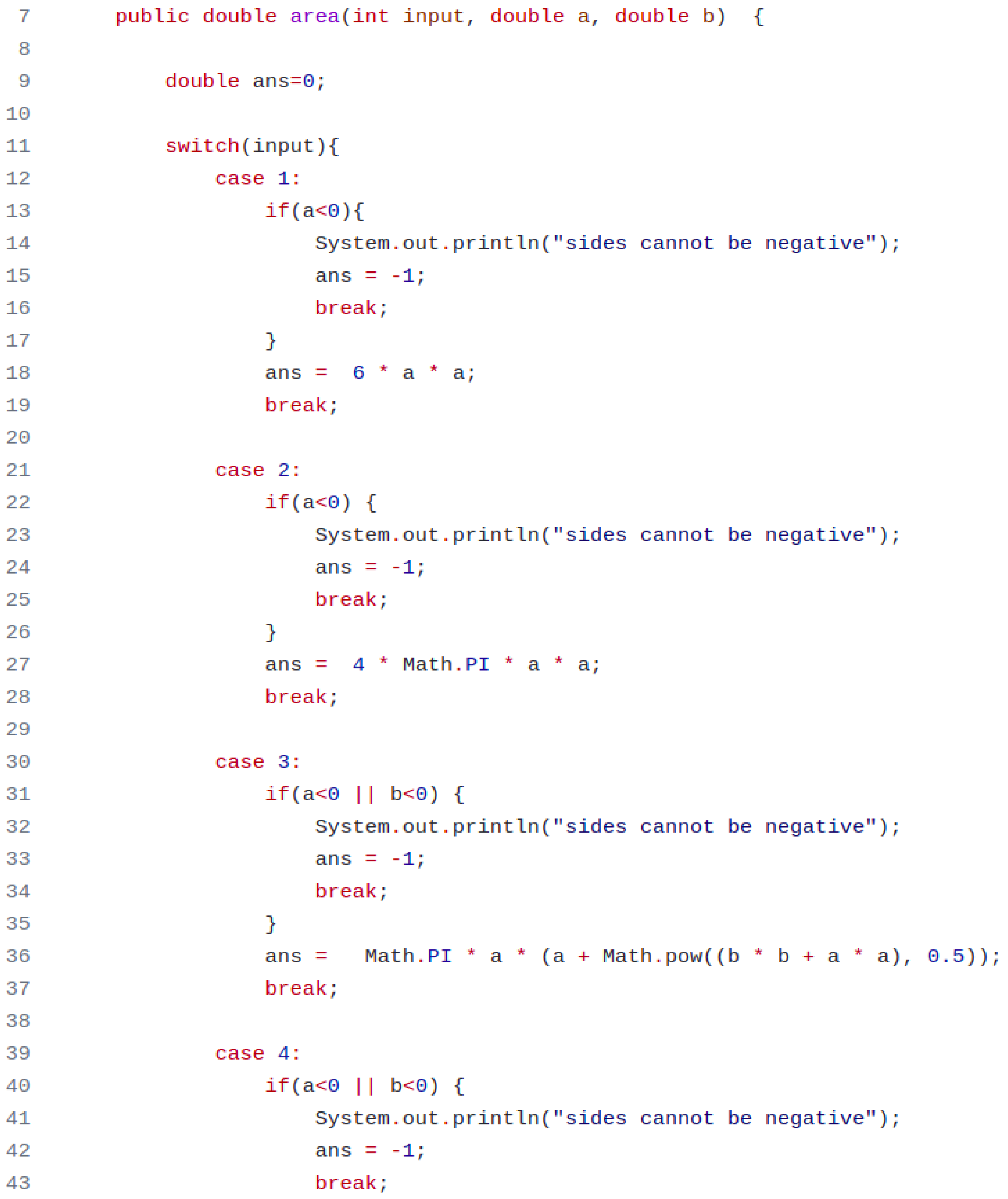
Tools Used

- JUnit5 for testing
- http://cs.gmu.edu/~offutt/softwaretest/ for TR generation

1. Math

Area.java

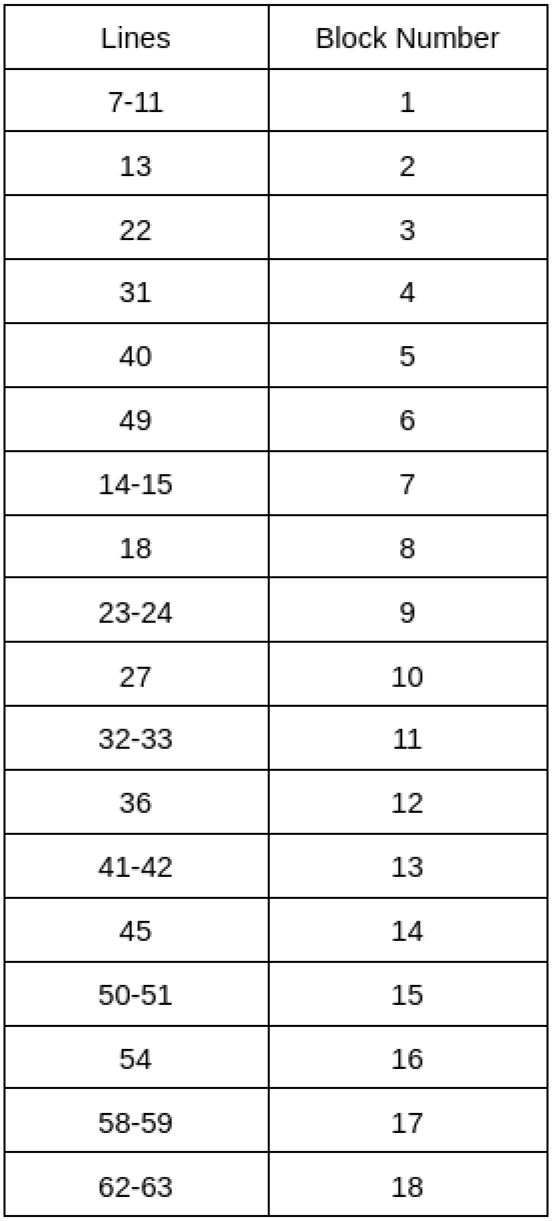
- Code



```
    public double area(int input, double a, double b)  {

        double ans=0;

        switch(input){
            case 1:
                if(a<0){
                    System.out.println("sides cannot be negative");
                    ans = -1;
                    break;
                }
                ans =  6 * a * a;
                break;

            case 2:
                if(a<0) {
                    System.out.println("sides cannot be negative");
                    ans = -1;
                    break;
                }
                ans =  4 * Math.PI * a * a;
                break;

            case 3:
                if(a<0 || b<0) {
                    System.out.println("sides cannot be negative");
                    ans = -1;
                    break;
                }
                ans =   Math.PI * a * (a + Math.pow((b * b + a * a), 0.5));
                break;

            case 4:
                if(a<0 || b<0) {
                    System.out.println("sides cannot be negative");
                    ans = -1;
                    break;
```

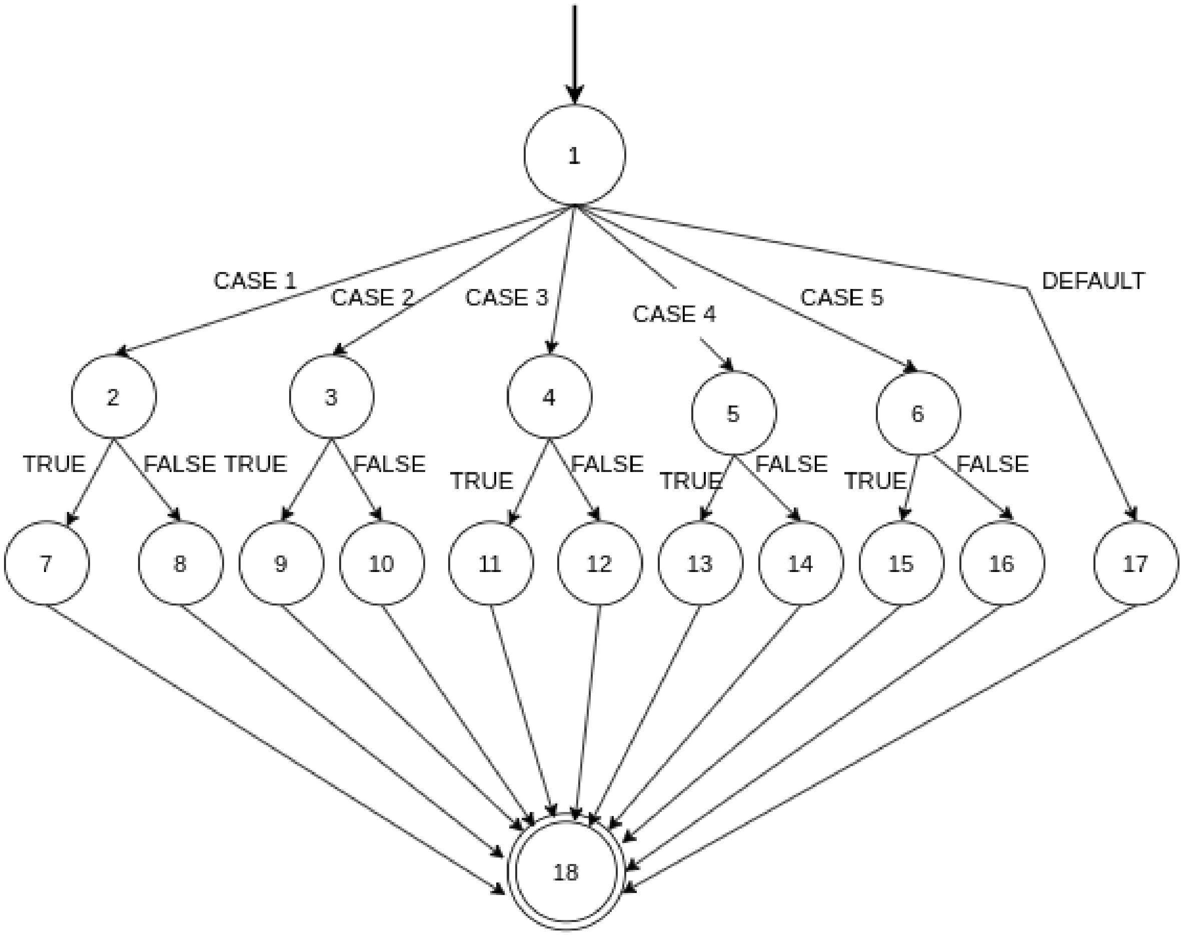
```java
                }
                ans =  3 * Math.PI * a * a;
                break;

            case 5:
                if(a<0 || b<0) {
                    System.out.println("sides cannot be negative");
                    ans = -1;
                    break;
                }
                ans = 2 * (Math.PI * a * a + Math.PI * a * b);
                break;

            default:
                System.out.println("invalid input");
                ans = -1;
        }

        System.out.println("Result: "+ans);
        return ans;

    }
```

- Basic Block



Lines	Block Number
7-11	1
13	2
22	3
31	4
40	5
49	6
14-15	7
18	8
23-24	9
27	10
32-33	11
36	12
41-42	13
45	14
50-51	15
54	16
58-59	17
62-63	18

- CFG

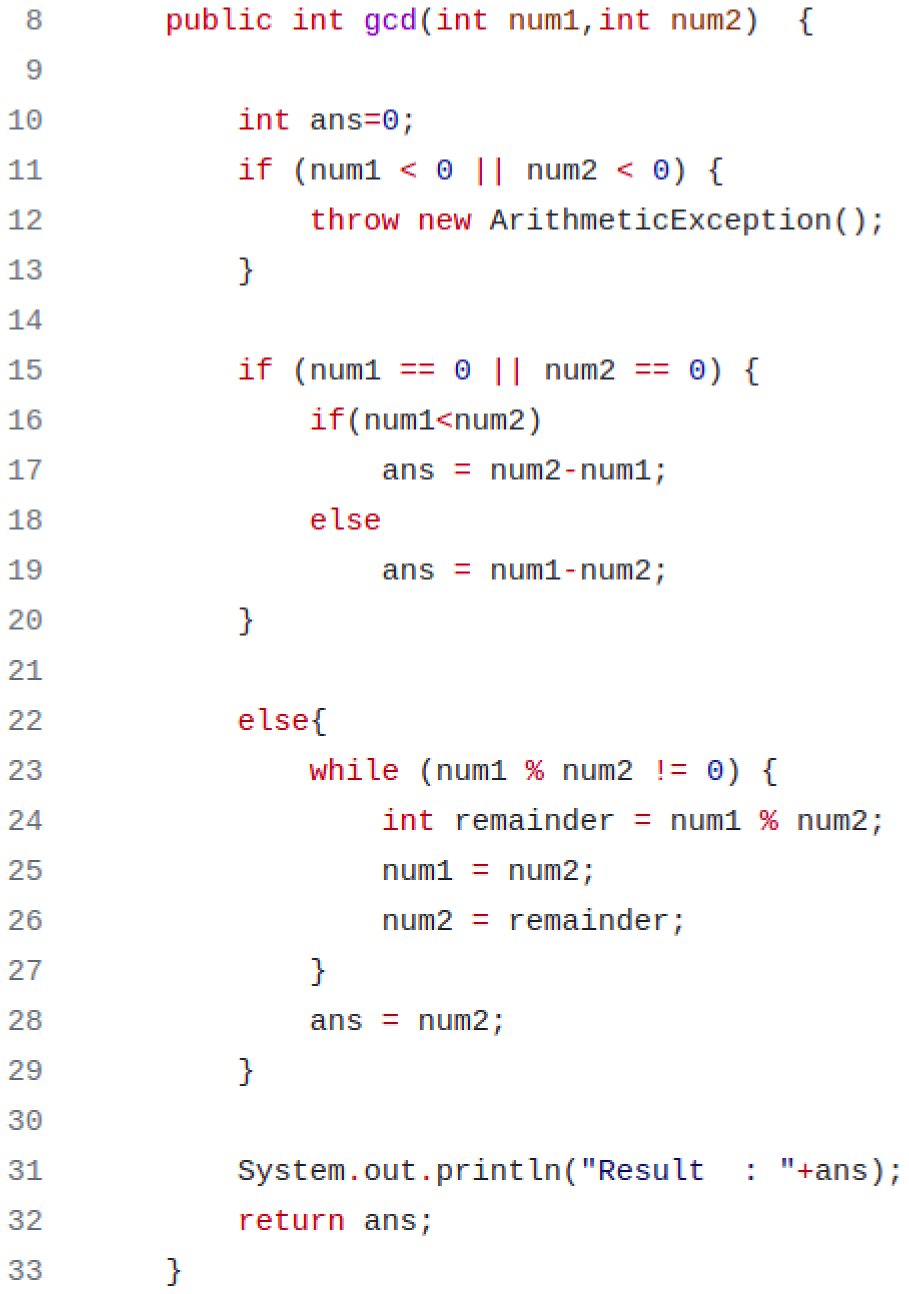


- Edge coverage

#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,2,7,18,End]	a = -4, b=0,input=1	-1
2	[Start,1,3,10,18,End]	a = 1, b=0,input=2	12.566370614359172
3	[Start,1,5,13,18,End]	a = -2, b=0,input=4	-1
4	[Start,1,4,12,18,End]	a = 4, b = 2, input=3	37.69911184307752

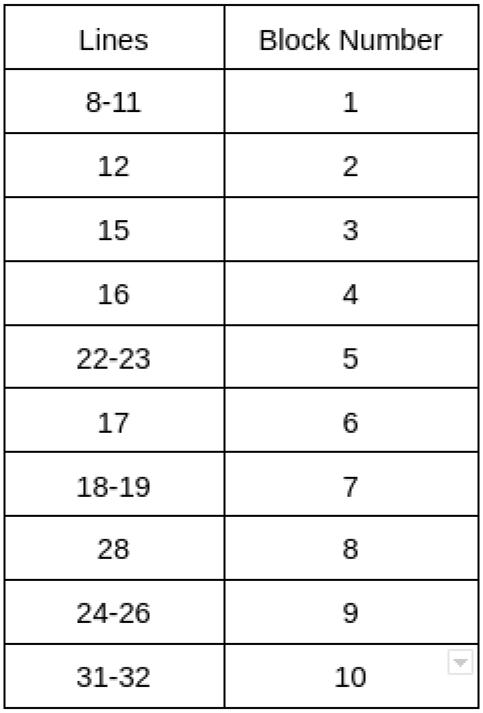
GCD.java

- Code



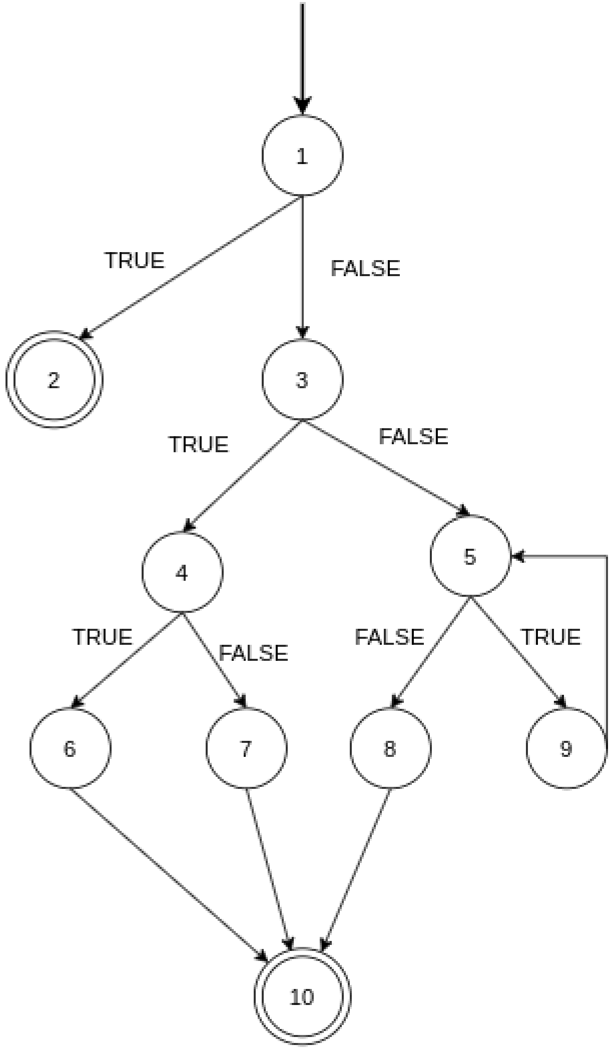
```java
    public int gcd(int num1,int num2)  {

        int ans=0;
        if (num1 < 0 || num2 < 0) {
            throw new ArithmeticException();
        }

        if (num1 == 0 || num2 == 0) {
            if(num1<num2)
                ans = num2-num1;
            else
                ans = num1-num2;
        }

        else{
            while (num1 % num2 != 0) {
                int remainder = num1 % num2;
                num1 = num2;
                num2 = remainder;
            }
            ans = num2;
        }

        System.out.println("Result  : "+ans);
        return ans;
    }
```

- Basic Block



Lines	Block Number
8-11	1
12	2
15	3
16	4
22-23	5
17	6
18-19	7
28	8
24-26	9
31-32	10

- CFG

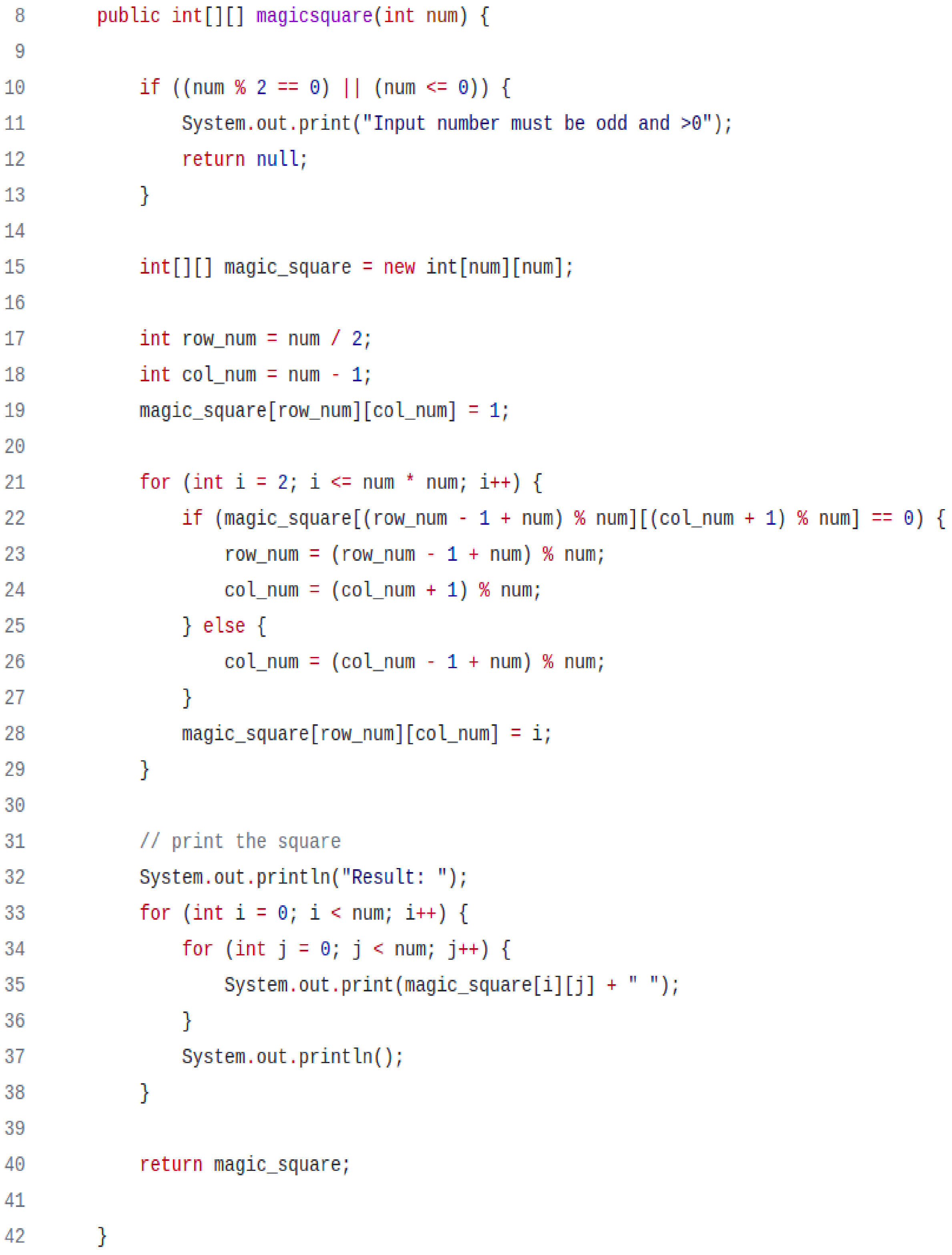


- Edge coverage

#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,2,End]	num1 = -2, num2 = 3	-1
2	[Start,1,3,4,6,10,End]	num1 = 0, num2 = 4	4
3	[Start,1,3,5,8,10,End]	num1 = 4, num2 = 2	2
4	[Start,1,3,5,9,5,8,10, End]	num1 = 4, num2 = 3	1

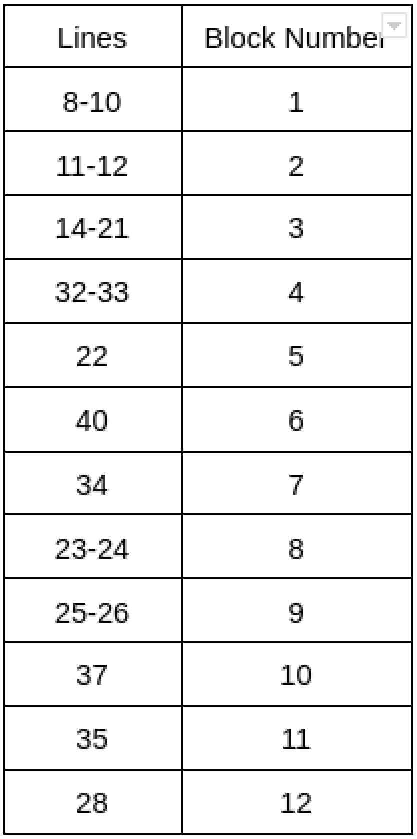
MagicSquare.java

- Code



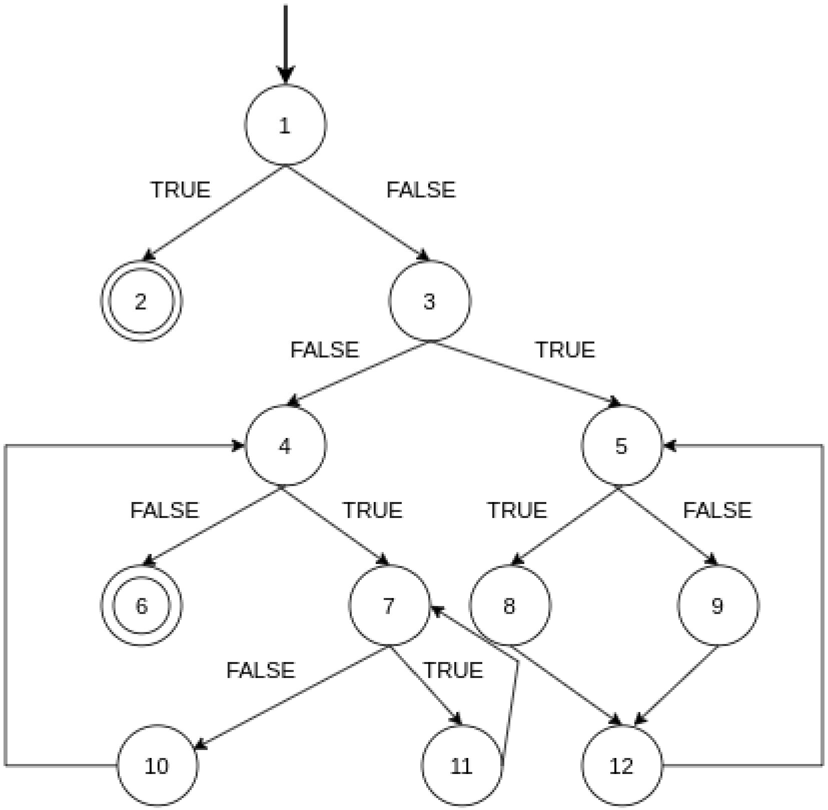
```java
public int[][] magicsquare(int num) {

    if ((num % 2 == 0) || (num <= 0)) {
        System.out.print("Input number must be odd and >0");
        return null;
    }

    int[][] magic_square = new int[num][num];

    int row_num = num / 2;
    int col_num = num - 1;
    magic_square[row_num][col_num] = 1;

    for (int i = 2; i <= num * num; i++) {
        if (magic_square[(row_num - 1 + num) % num][(col_num + 1) % num] == 0) {
            row_num = (row_num - 1 + num) % num;
            col_num = (col_num + 1) % num;
        } else {
            col_num = (col_num - 1 + num) % num;
        }
        magic_square[row_num][col_num] = i;
    }

    // print the square
    System.out.println("Result: ");
    for (int i = 0; i < num; i++) {
        for (int j = 0; j < num; j++) {
            System.out.print(magic_square[i][j] + " ");
        }
        System.out.println();
    }

    return magic_square;

}
```

- Basic Block



Lines	Block Number
8-10	1
11-12	2
14-21	3
32-33	4
22	5
40	6
34	7
23-24	8
25-26	9
37	10
35	11
28	12

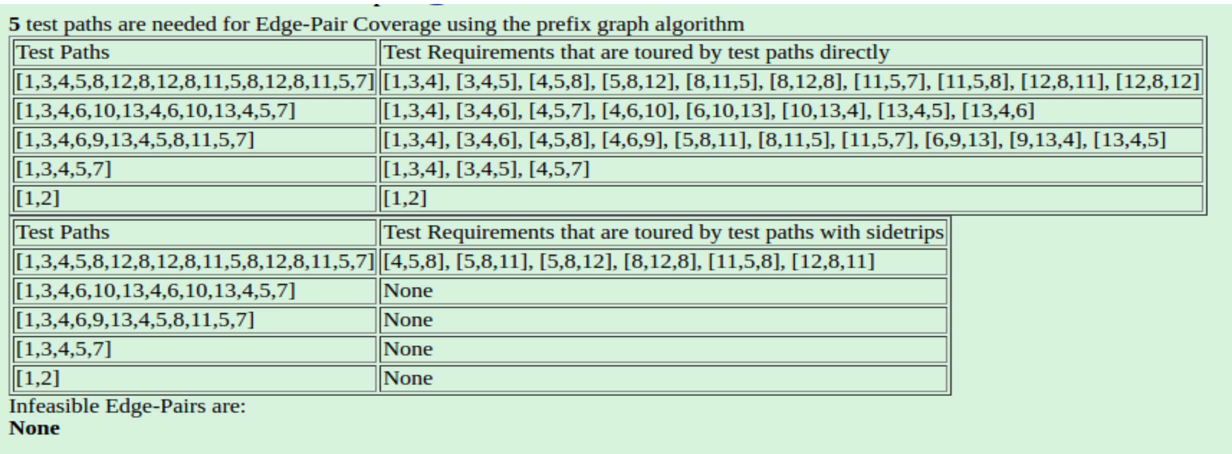
- CFG



- Edge coverage

#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,2,End]	num = 4	null
2	[Start,1,3,4,5,8,12,8,11,5,7, End]	num = 1	[1]

- Edge Pair coverage



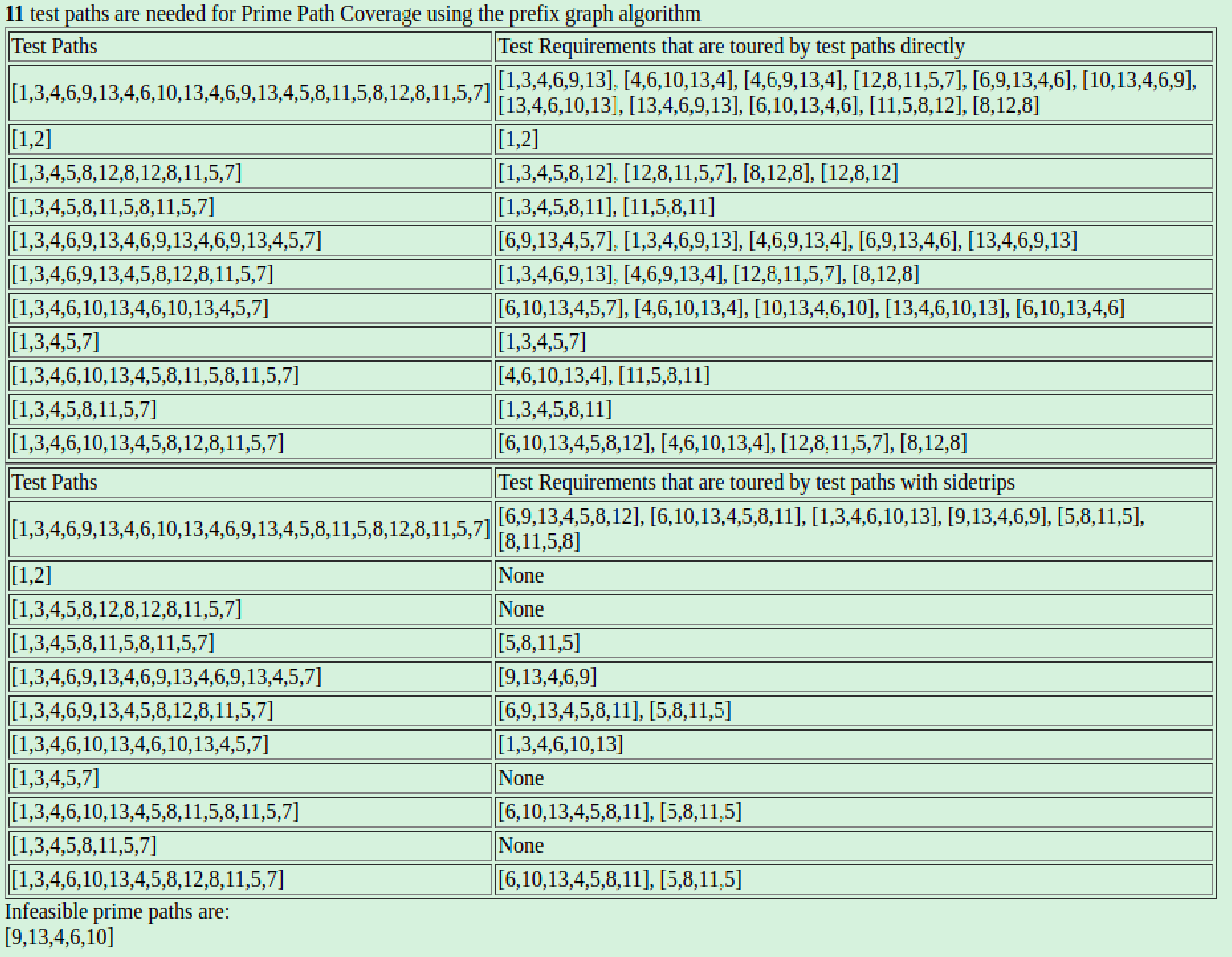
5 test paths are needed for Edge-Pair Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,3,4,5,8,12,8,12,8,11,5,8,12,8,11,5,7]	[1,3,4], [3,4,5], [4,5,8], [5,8,12], [8,11,5], [8,12,8], [11,5,7], [11,5,8], [12,8,11], [12,8,12]
[1,3,4,6,10,13,4,6,10,13,4,5,7]	[1,3,4], [3,4,6], [4,5,7], [4,6,10], [6,10,13], [10,13,4], [13,4,5], [13,4,6]
[1,3,4,6,9,13,4,5,8,11,5,7]	[1,3,4], [3,4,6], [4,5,8], [4,6,9], [5,8,11], [8,11,5], [11,5,7], [6,9,13], [9,13,4], [13,4,5]
[1,3,4,5,7]	[1,3,4], [3,4,5], [4,5,7]
[1,2]	[1,2]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,3,4,5,8,12,8,12,8,11,5,8,12,8,11,5,7]	[4,5,8], [5,8,11], [5,8,12], [8,12,8], [11,5,8], [12,8,11]
[1,3,4,6,10,13,4,6,10,13,4,5,7]	None
[1,3,4,6,9,13,4,5,8,11,5,7]	None
[1,3,4,5,7]	None
[1,2]	None

Infeasible Edge-Pairs are:
None

- Prime Path coverage



11 test paths are needed for Prime Path Coverage using the prefix graph algorithm

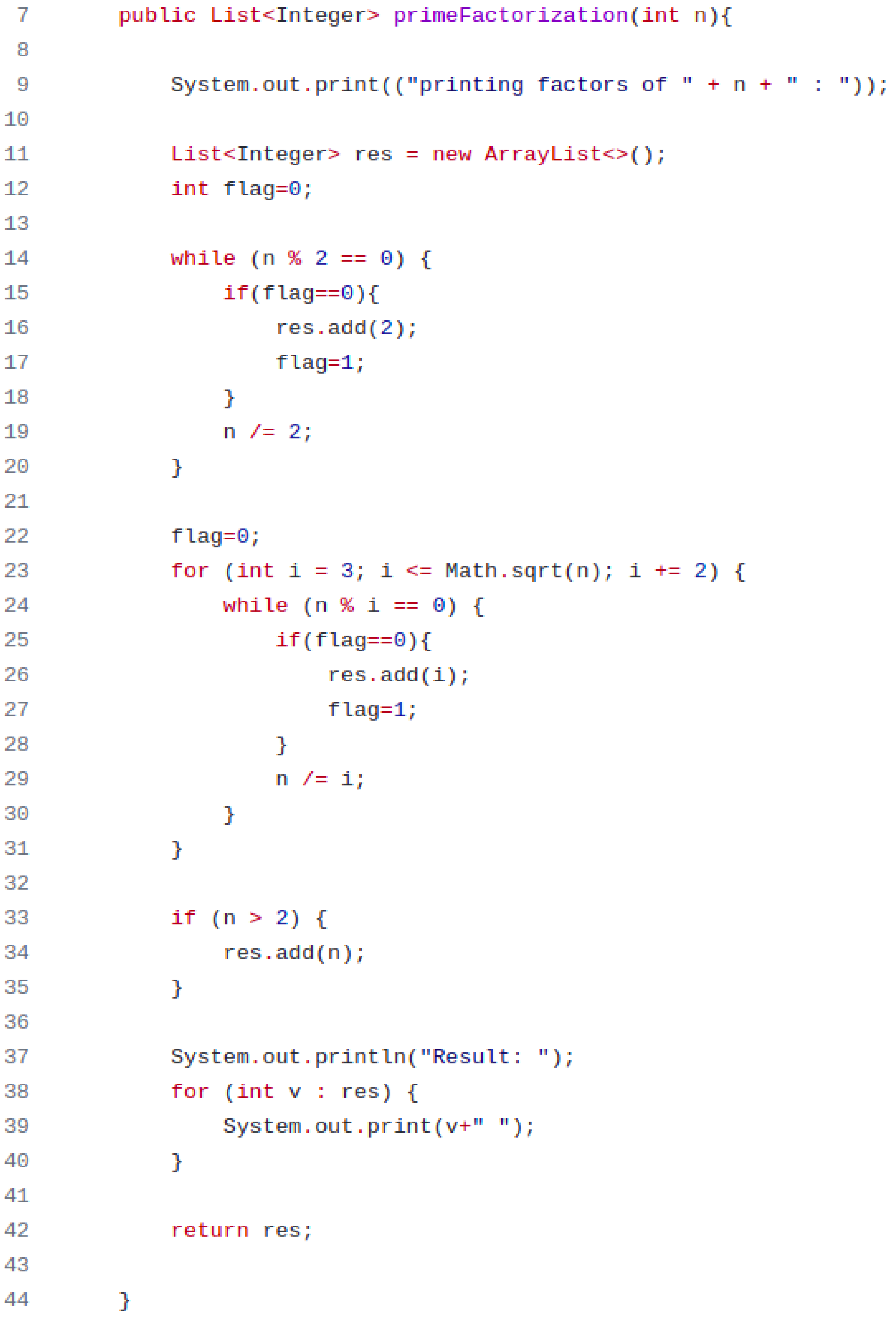
Test Paths	Test Requirements that are toured by test paths directly
[1,3,4,6,9,13,4,6,10,13,4,6,9,13,4,5,8,11,5,8,12,8,11,5,7]	[1,3,4,6,9,13], [4,6,10,13,4], [4,6,9,13,4], [12,8,11,5,7], [6,9,13,4,6], [10,13,4,6,9], [13,4,6,10,13], [13,4,6,9,13], [6,10,13,4,6], [11,5,8,12], [8,12,8]
[1,2]	[1,2]
[1,3,4,5,8,12,8,12,8,11,5,7]	[1,3,4,5,8,12], [12,8,11,5,7], [8,12,8], [12,8,12]
[1,3,4,5,8,11,5,8,11,5,7]	[1,3,4,5,8,11], [11,5,8,11]
[1,3,4,6,9,13,4,6,9,13,4,6,9,13,4,5,7]	[6,9,13,4,5,7], [1,3,4,6,9,13], [4,6,9,13,4], [6,9,13,4,6], [13,4,6,9,13]
[1,3,4,6,9,13,4,5,8,12,8,11,5,7]	[1,3,4,6,9,13], [4,6,9,13,4], [12,8,11,5,7], [8,12,8]
[1,3,4,6,10,13,4,6,10,13,4,5,7]	[6,10,13,4,5,7], [4,6,10,13,4], [10,13,4,6,10], [13,4,6,10,13], [6,10,13,4,6]
[1,3,4,5,7]	[1,3,4,5,7]
[1,3,4,6,10,13,4,5,8,11,5,8,11,5,7]	[4,6,10,13,4], [11,5,8,11]
[1,3,4,5,8,11,5,7]	[1,3,4,5,8,11]
[1,3,4,6,10,13,4,5,8,12,8,11,5,7]	[6,10,13,4,5,8,12], [4,6,10,13,4], [12,8,11,5,7], [8,12,8]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,3,4,6,9,13,4,6,10,13,4,6,9,13,4,5,8,11,5,8,12,8,11,5,7]	[6,9,13,4,5,8,12], [6,10,13,4,5,8,11], [1,3,4,6,10,13], [9,13,4,6,9], [5,8,11,5], [8,11,5,8]
[1,2]	None
[1,3,4,5,8,12,8,12,8,11,5,7]	None
[1,3,4,5,8,11,5,8,11,5,7]	[5,8,11,5]
[1,3,4,6,9,13,4,6,9,13,4,6,9,13,4,5,7]	[9,13,4,6,9]
[1,3,4,6,9,13,4,5,8,12,8,11,5,7]	[6,9,13,4,5,8,11], [5,8,11,5]
[1,3,4,6,10,13,4,6,10,13,4,5,7]	[1,3,4,6,10,13]
[1,3,4,5,7]	None
[1,3,4,6,10,13,4,5,8,11,5,8,11,5,7]	[6,10,13,4,5,8,11], [5,8,11,5]
[1,3,4,5,8,11,5,7]	None
[1,3,4,6,10,13,4,5,8,12,8,11,5,7]	[6,10,13,4,5,8,11], [5,8,11,5]

Infeasible prime paths are:
[9,13,4,6,10]

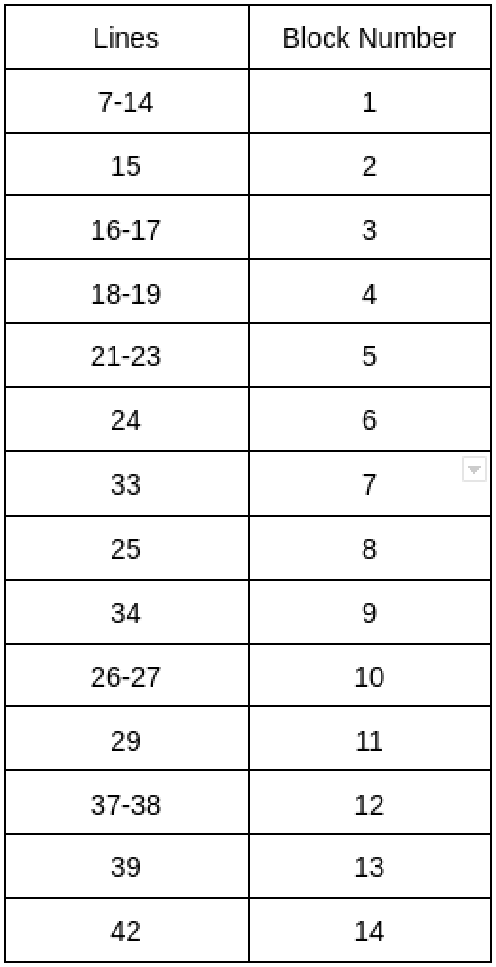
PrimeFactorization.java

- Code



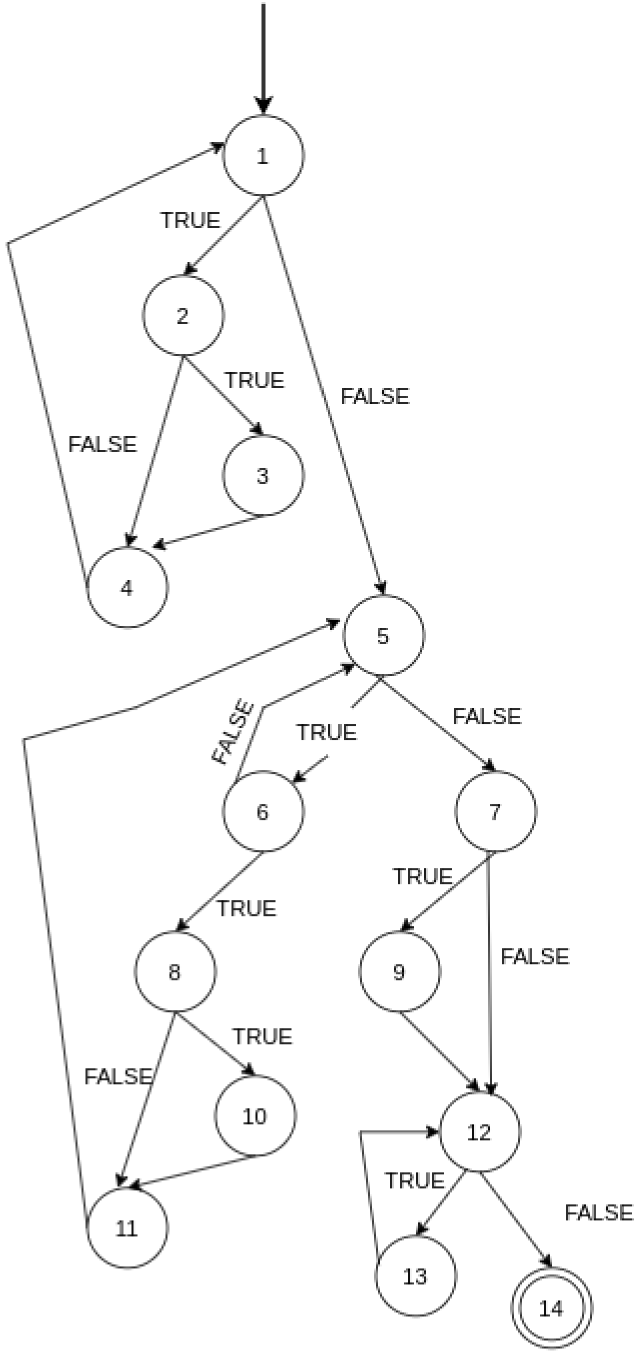
```java
public List<Integer> primeFactorization(int n){

    System.out.print(("printing factors of " + n + " : "));

    List<Integer> res = new ArrayList<>();
    int flag=0;

    while (n % 2 == 0) {
        if(flag==0){
            res.add(2);
            flag=1;
        }
        n /= 2;
    }

    flag=0;
    for (int i = 3; i <= Math.sqrt(n); i += 2) {
        while (n % i == 0) {
            if(flag==0){
                res.add(i);
                flag=1;
            }
            n /= i;
        }
    }

    if (n > 2) {
        res.add(n);
    }

    System.out.println("Result: ");
    for (int v : res) {
        System.out.print(v+" ");
    }

    return res;

}
```

- Basic Block



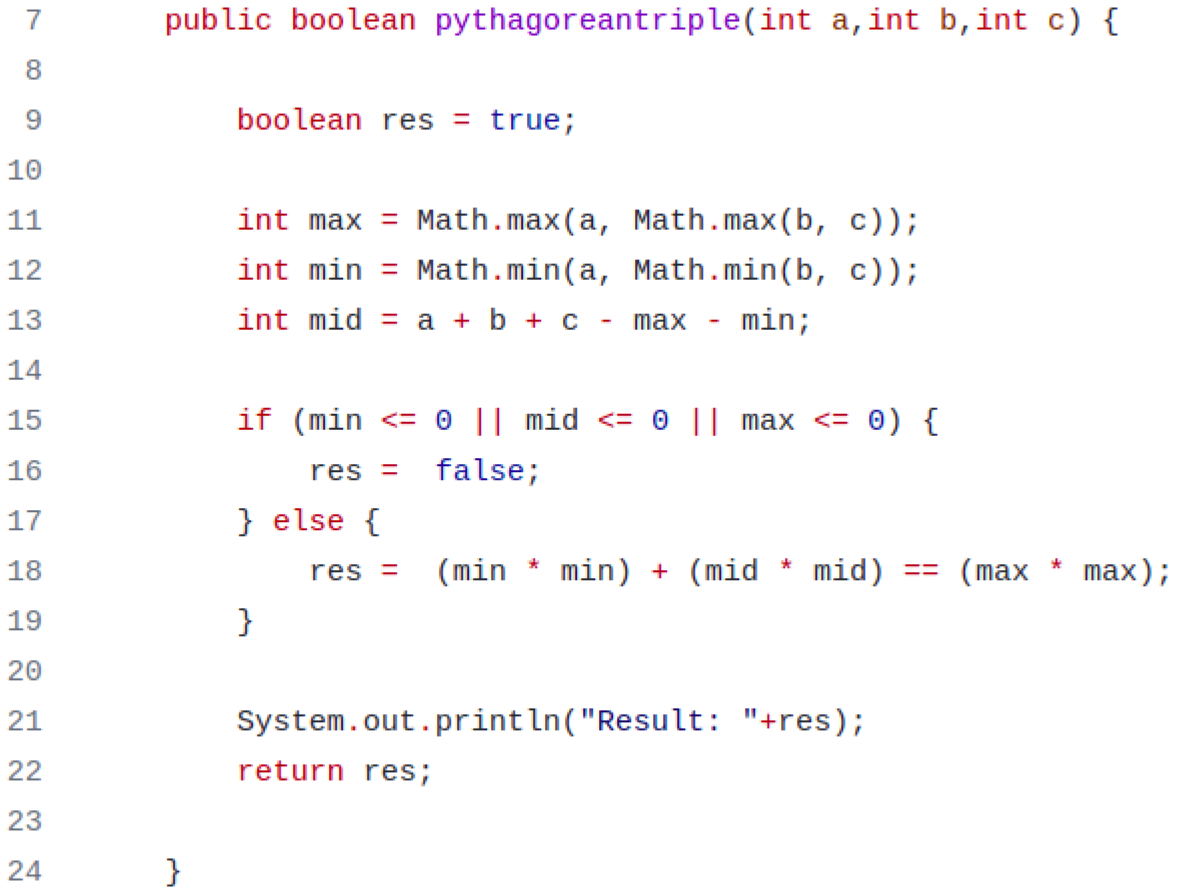
Lines	Block Number
7-14	1
15	2
16-17	3
18-19	4
21-23	5
24	6
33	7
25	8
34	9
26-27	10
29	11
37-38	12
39	13
42	14

- CFG


1
TRUE
2
TRUE
FALSE
FALSE
3
4
5
FALSE
TRUE
FALSE
6
7
TRUE
TRUE
8
9
FALSE
TRUE
FALSE
10
12
11
TRUE
FALSE
13
14

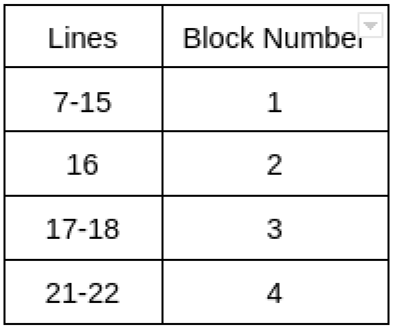
PythagoreanTriple.java

- Code



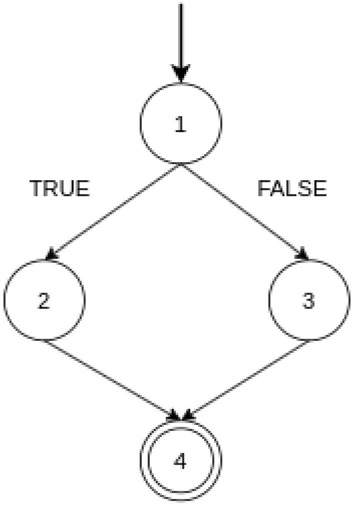
```java
public boolean pythagoreantriple(int a,int b,int c) {

    boolean res = true;

    int max = Math.max(a, Math.max(b, c));
    int min = Math.min(a, Math.min(b, c));
    int mid = a + b + c - max - min;

    if (min <= 0 || mid <= 0 || max <= 0) {
        res =  false;
    } else {
        res =  (min * min) + (mid * mid) == (max * max);
    }

    System.out.println("Result: "+res);
    return res;

}
```

- Basic Block

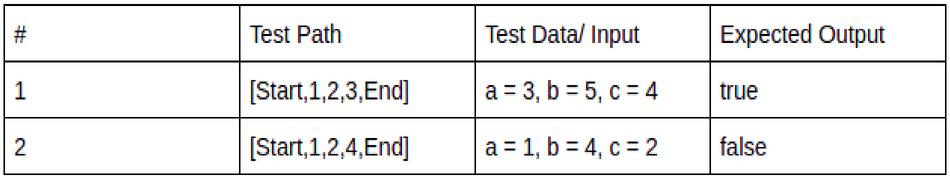


Lines	Block Number
7-15	1
16	2
17-18	3
21-22	4

- CFG



- Edge coverage

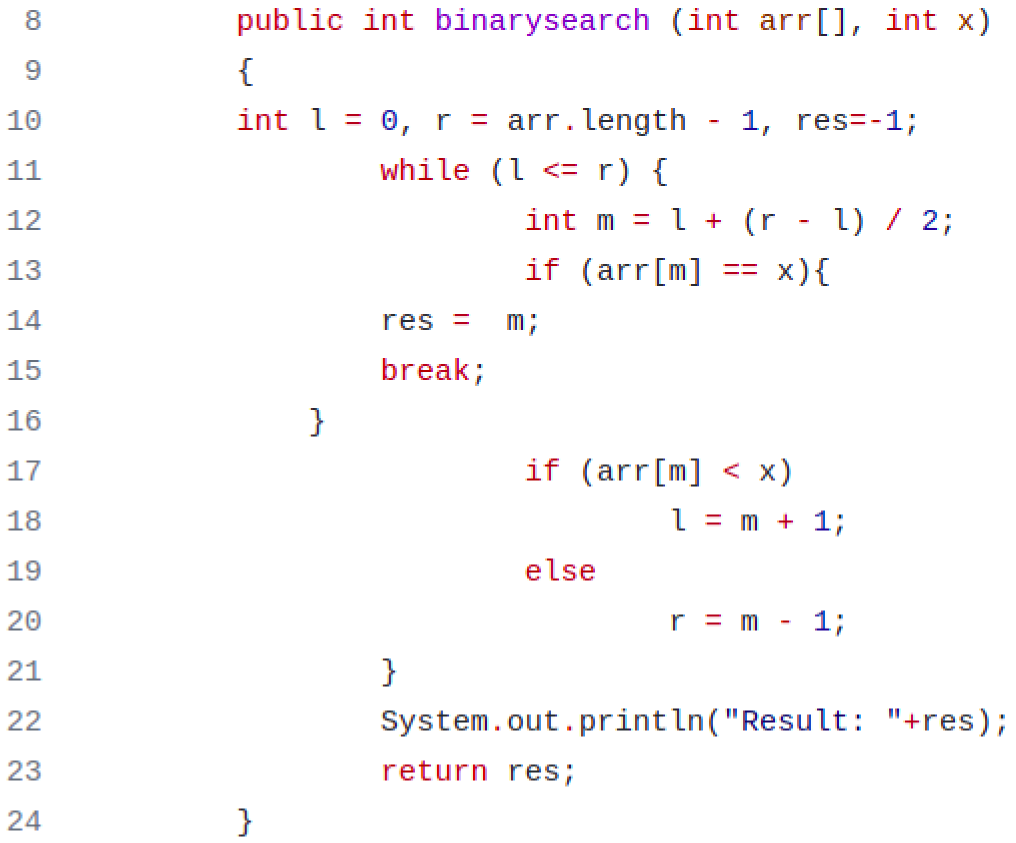


#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,2,3,End]	a = 3, b = 5, c = 4	true
2	[Start,1,2,4,End]	a = 1, b = 4, c = 2	false

2. Search

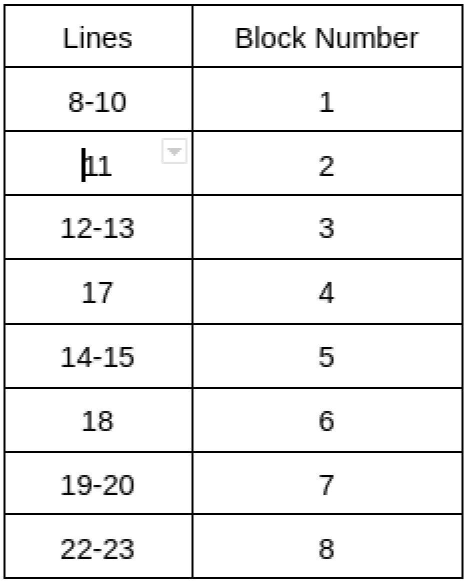
BinarySearch.java

- Code



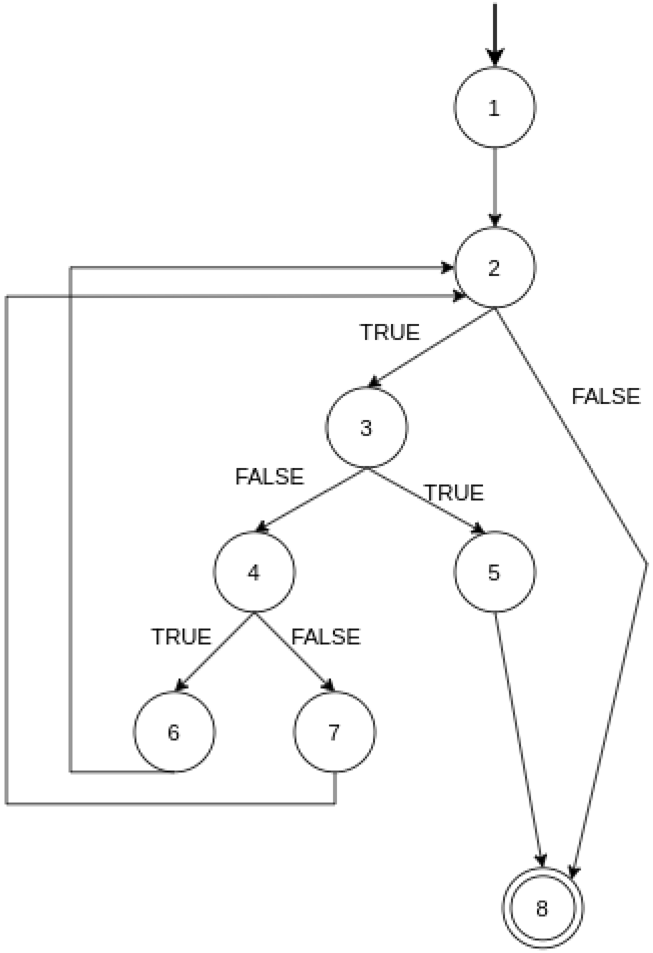
```
 8  public int binarysearch (int arr[], int x)
 9  {
10  int l = 0, r = arr.length - 1, res=-1;
11          while (l <= r) {
12                  int m = l + (r - l) / 2;
13                  if (arr[m] == x){
14          res =  m;
15          break;
16      }
17                  if (arr[m] < x)
18                          l = m + 1;
19                  else
20                          r = m - 1;
21          }
22          System.out.println("Result: "+res);
23          return res;
24  }
```

- Basic Block



Lines	Block Number
8-10	1
11	2
12-13	3
17	4
14-15	5
18	6
19-20	7
22-23	8

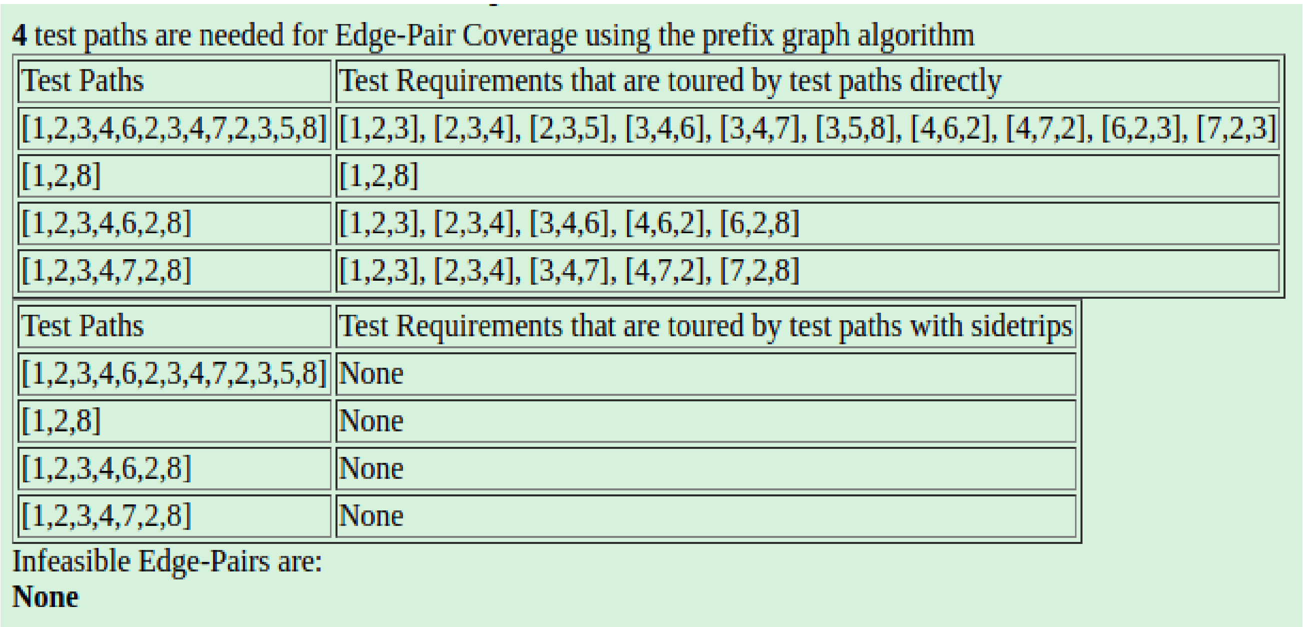
- CFG



- Edge coverage

#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,2,8,End]	arr[] = [], x = 2	-1
2	[Start,1,2,3,5,8,End]	arr[] = [1,2,3], x = 2	1
3	[Start,1,2,3,4,7,2,8,End]	arr[] = [1,2,3,4,5], x=1	0
4	[Start,1,2,3,4,6,2,8,End]	arr[] = [1,2,3,4,5], x = 5	4

- Edge Pair coverage



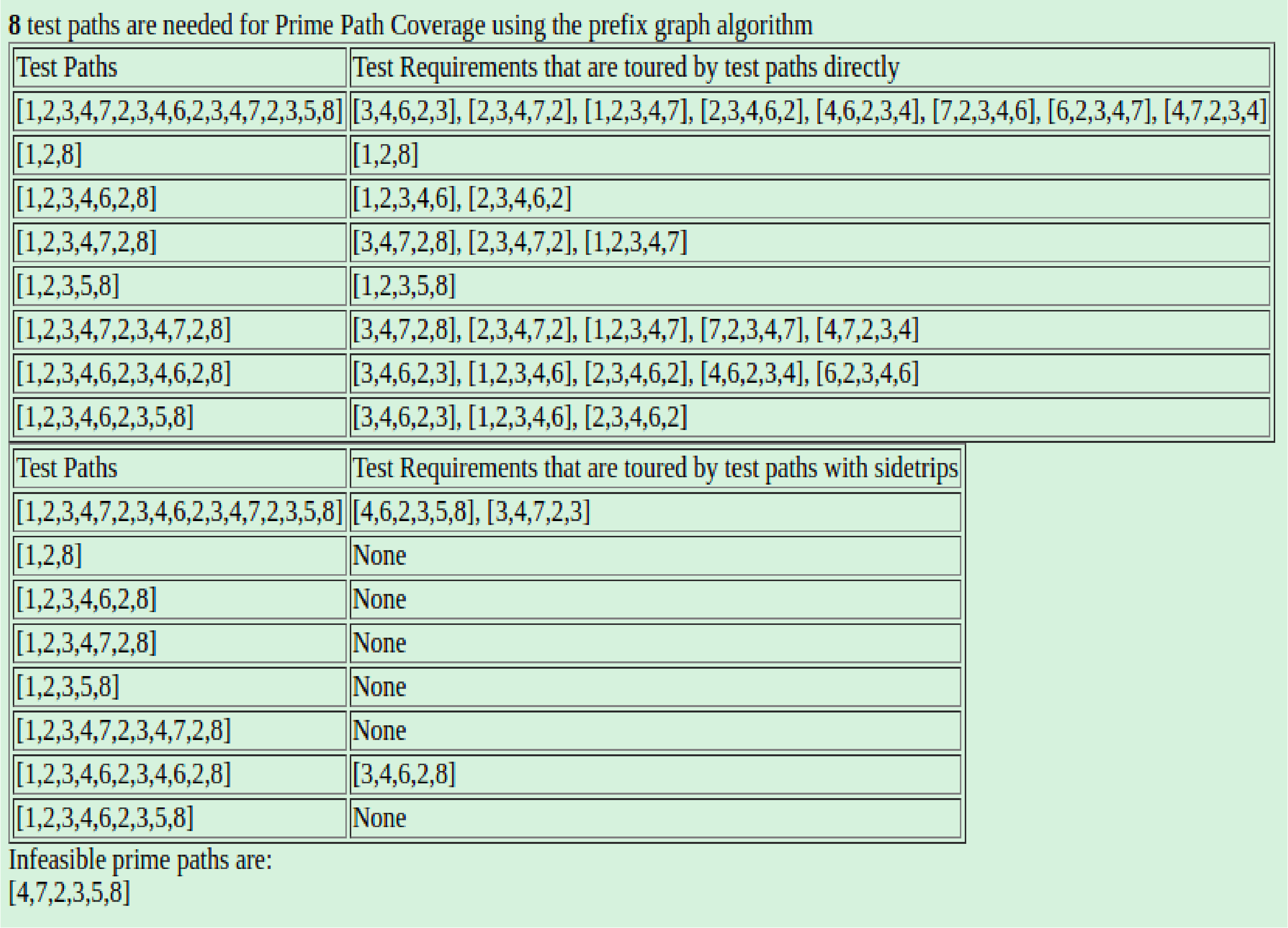
4 test paths are needed for Edge-Pair Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,2,3,4,6,2,3,4,7,2,3,5,8]	[1,2,3], [2,3,4], [2,3,5], [3,4,6], [3,4,7], [3,5,8], [4,6,2], [4,7,2], [6,2,3], [7,2,3]
[1,2,8]	[1,2,8]
[1,2,3,4,6,2,8]	[1,2,3], [2,3,4], [3,4,6], [4,6,2], [6,2,8]
[1,2,3,4,7,2,8]	[1,2,3], [2,3,4], [3,4,7], [4,7,2], [7,2,8]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,2,3,4,6,2,3,4,7,2,3,5,8]	None
[1,2,8]	None
[1,2,3,4,6,2,8]	None
[1,2,3,4,7,2,8]	None

Infeasible Edge-Pairs are:
None

- Prime Path coverage



8 test paths are needed for Prime Path Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,2,3,4,7,2,3,4,6,2,3,4,7,2,3,5,8]	[3,4,6,2,3], [2,3,4,7,2], [1,2,3,4,7], [2,3,4,6,2], [4,6,2,3,4], [7,2,3,4,6], [6,2,3,4,7], [4,7,2,3,4]
[1,2,8]	[1,2,8]
[1,2,3,4,6,2,8]	[1,2,3,4,6], [2,3,4,6,2]
[1,2,3,4,7,2,8]	[3,4,7,2,8], [2,3,4,7,2], [1,2,3,4,7]
[1,2,3,5,8]	[1,2,3,5,8]
[1,2,3,4,7,2,3,4,7,2,8]	[3,4,7,2,8], [2,3,4,7,2], [1,2,3,4,7], [7,2,3,4,7], [4,7,2,3,4]
[1,2,3,4,6,2,3,4,6,2,8]	[3,4,6,2,3], [1,2,3,4,6], [2,3,4,6,2], [4,6,2,3,4], [6,2,3,4,6]
[1,2,3,4,6,2,3,5,8]	[3,4,6,2,3], [1,2,3,4,6], [2,3,4,6,2]

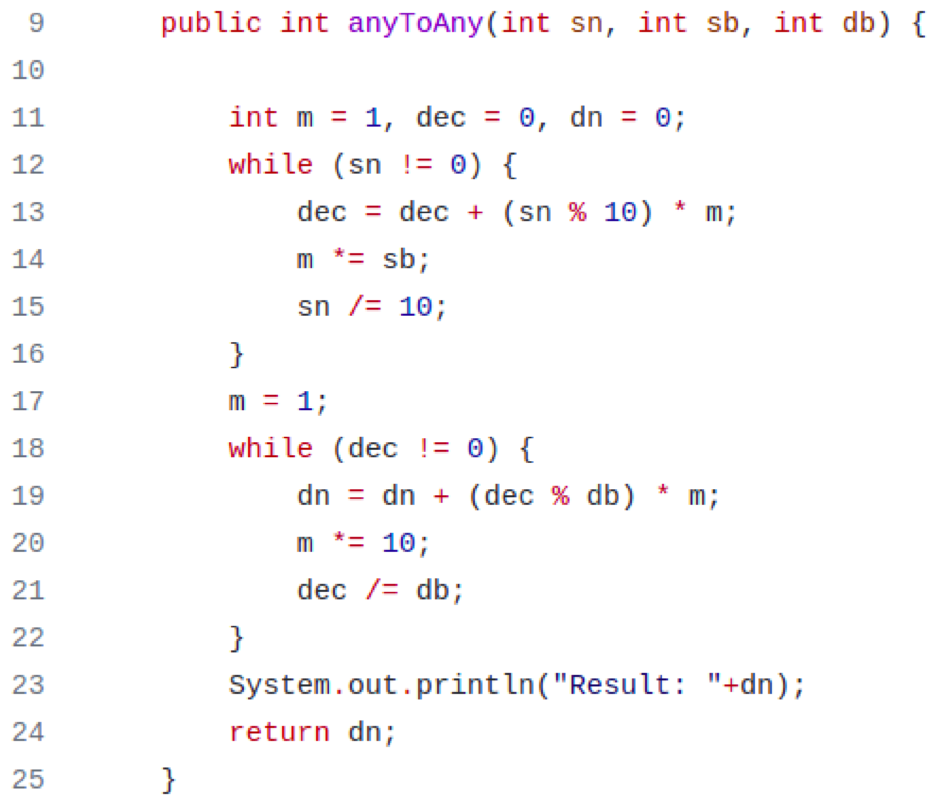
Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,2,3,4,7,2,3,4,6,2,3,4,7,2,3,5,8]	[4,6,2,3,5,8], [3,4,7,2,3]
[1,2,8]	None
[1,2,3,4,6,2,8]	None
[1,2,3,4,7,2,8]	None
[1,2,3,5,8]	None
[1,2,3,4,7,2,3,4,7,2,8]	None
[1,2,3,4,6,2,3,4,6,2,8]	[3,4,6,2,8]
[1,2,3,4,6,2,3,5,8]	None

Infeasible prime paths are:
[4,7,2,3,5,8]

3. Conversion

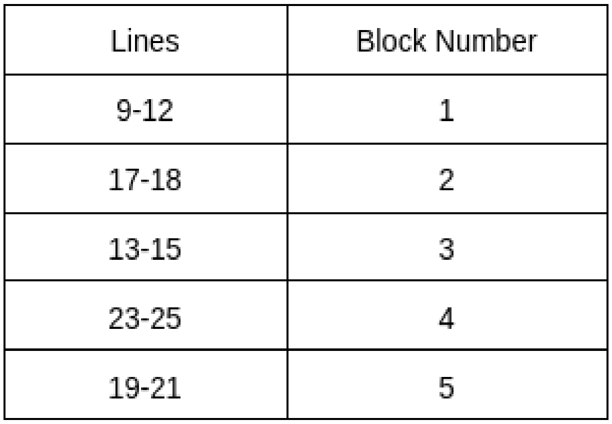
AnyToAny.java

- Code



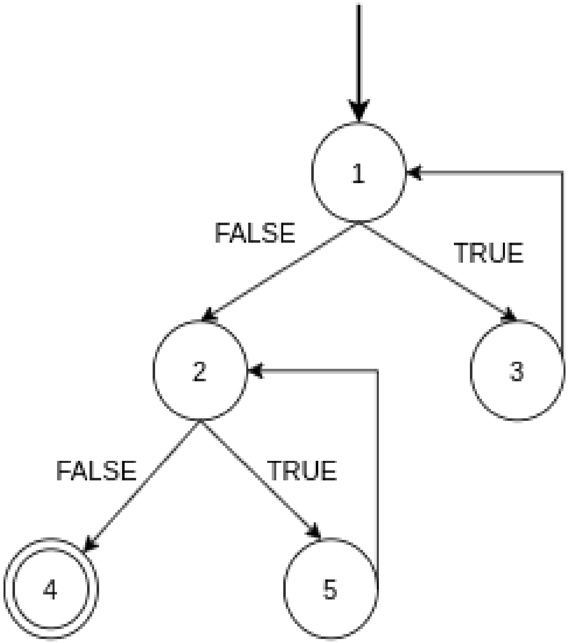
```
 9     public int anyToAny(int sn, int sb, int db) {
10
11         int m = 1, dec = 0, dn = 0;
12         while (sn != 0) {
13             dec = dec + (sn % 10) * m;
14             m *= sb;
15             sn /= 10;
16         }
17         m = 1;
18         while (dec != 0) {
19             dn = dn + (dec % db) * m;
20             m *= 10;
21             dec /= db;
22         }
23         System.out.println("Result: "+dn);
24         return dn;
25     }
```

- Basic Block



Lines	Block Number
9-12	1
17-18	2
13-15	3
23-25	4
19-21	5

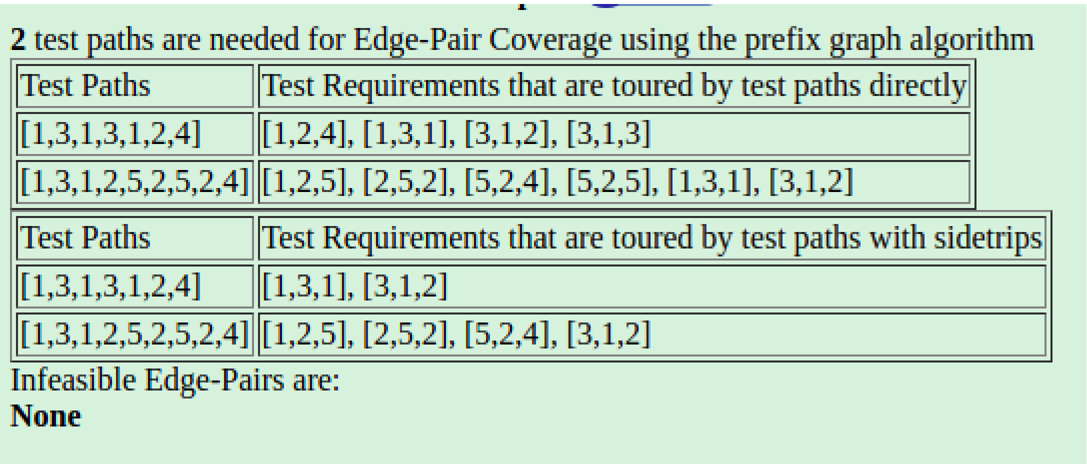
- CFG



- Edge coverage

#	Test Path	Test Data/ Input	Expected Output
1	[Start, 1,3,1,2,5,2,4, End]	sn = 10, sb = 2, db = 4	2
2	[Start, 1,2,4, End]	sn =0, sb=2, db = 4	0

- Edge Pair coverage



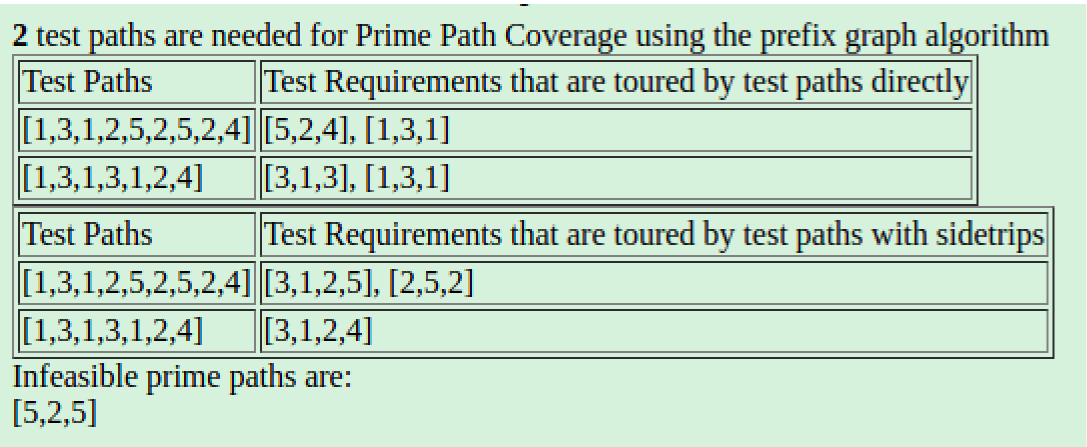
2 test paths are needed for Edge-Pair Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,3,1,3,1,2,4]	[1,2,4], [1,3,1], [3,1,2], [3,1,3]
[1,3,1,2,5,2,5,2,4]	[1,2,5], [2,5,2], [5,2,4], [5,2,5], [1,3,1], [3,1,2]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,3,1,3,1,2,4]	[1,3,1], [3,1,2]
[1,3,1,2,5,2,5,2,4]	[1,2,5], [2,5,2], [5,2,4], [3,1,2]

Infeasible Edge-Pairs are:
None

- Prime Path coverage



2 test paths are needed for Prime Path Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,3,1,2,5,2,5,2,4]	[5,2,4], [1,3,1]
[1,3,1,3,1,2,4]	[3,1,3], [1,3,1]

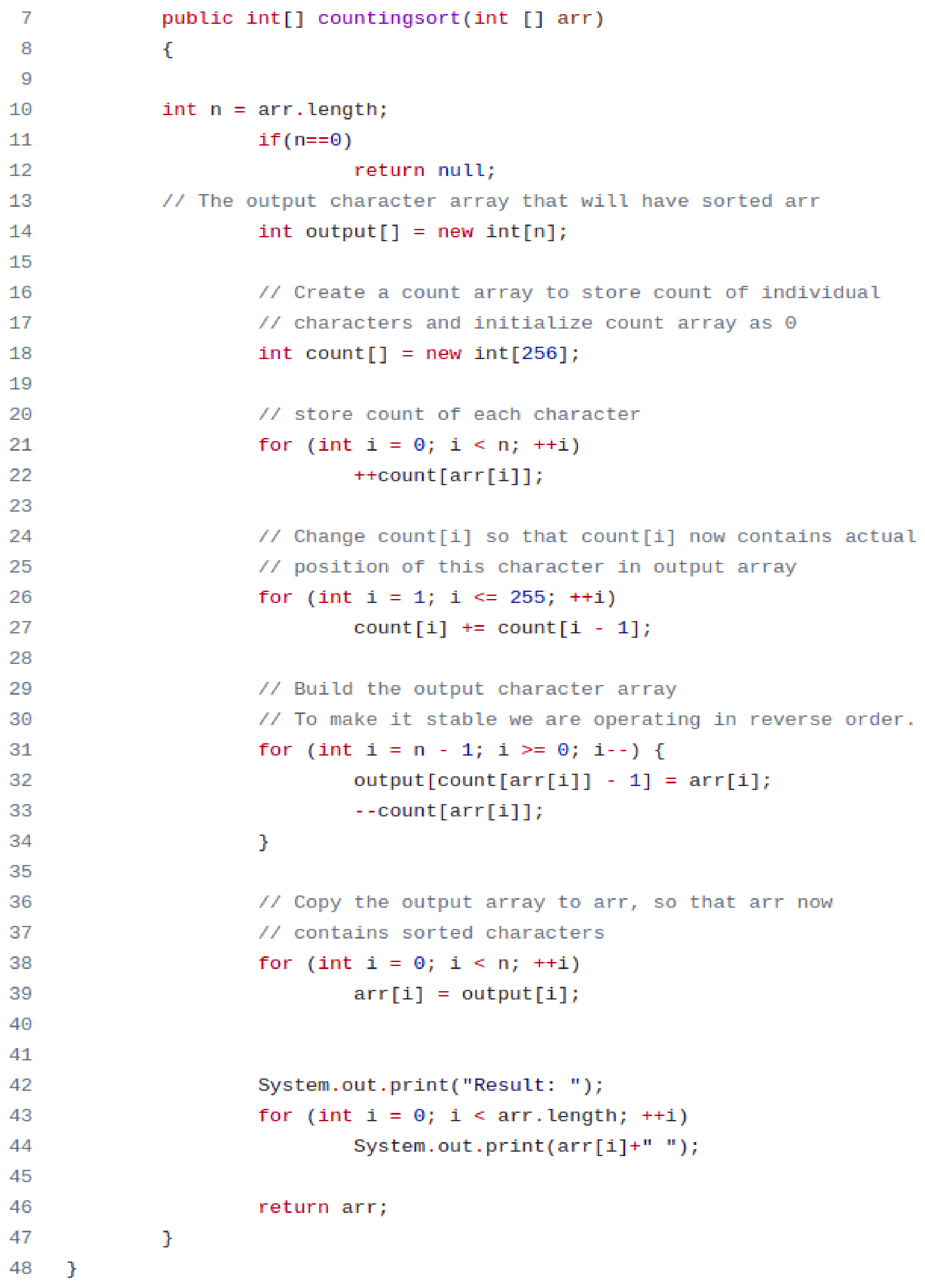
Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,3,1,2,5,2,5,2,4]	[3,1,2,5], [2,5,2]
[1,3,1,3,1,2,4]	[3,1,2,4]

Infeasible prime paths are:
[5,2,5]

4. Sort

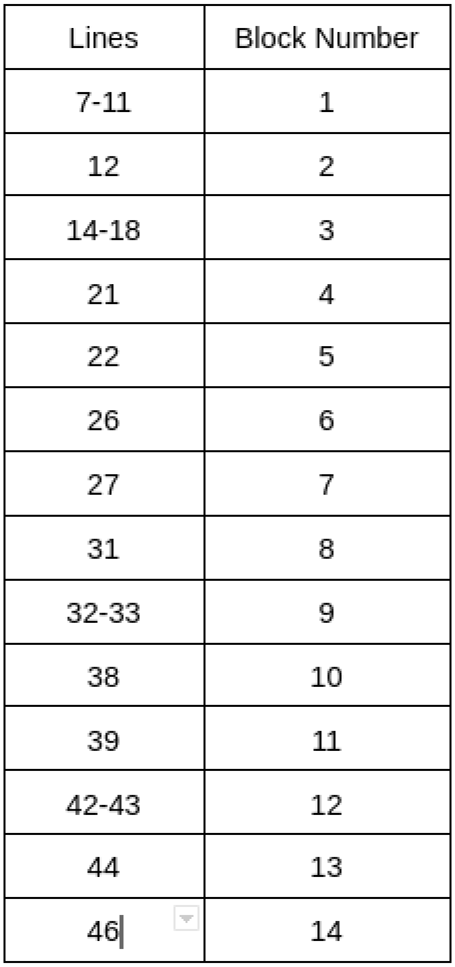
CountingSort.java

- Code



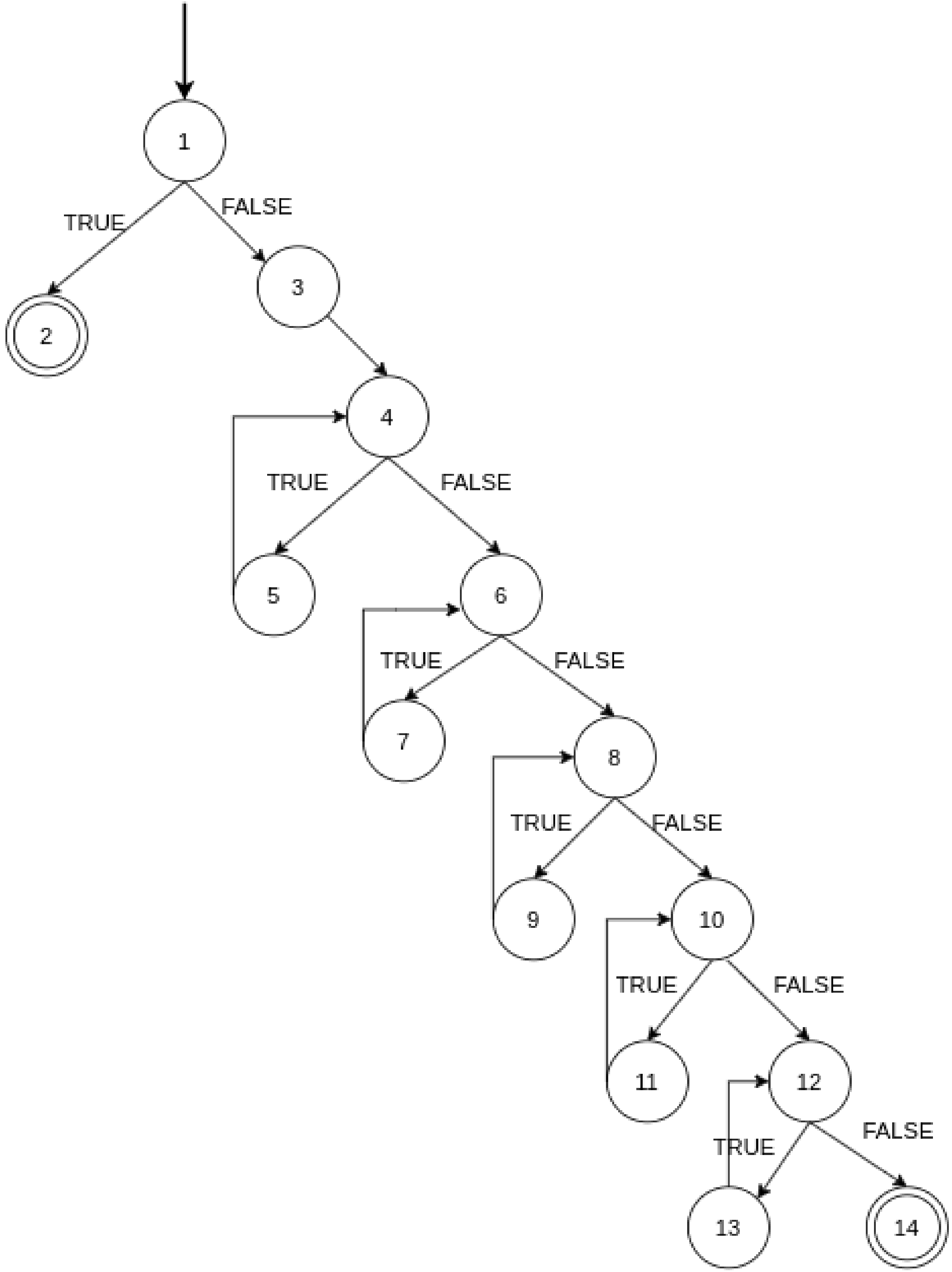
```java
    public int[] countingsort(int [] arr)
    {

    int n = arr.length;
        if(n==0)
            return null;
    // The output character array that will have sorted arr
        int output[] = new int[n];

        // Create a count array to store count of individual
        // characters and initialize count array as 0
        int count[] = new int[256];

        // store count of each character
        for (int i = 0; i < n; ++i)
            ++count[arr[i]];

        // Change count[i] so that count[i] now contains actual
        // position of this character in output array
        for (int i = 1; i <= 255; ++i)
            count[i] += count[i - 1];

        // Build the output character array
        // To make it stable we are operating in reverse order.
        for (int i = n - 1; i >= 0; i--) {
            output[count[arr[i]] - 1] = arr[i];
            --count[arr[i]];
        }

        // Copy the output array to arr, so that arr now
        // contains sorted characters
        for (int i = 0; i < n; ++i)
            arr[i] = output[i];


        System.out.print("Result: ");
        for (int i = 0; i < arr.length; ++i)
            System.out.print(arr[i]+" ");

        return arr;
    }
}
```

- Basic Block

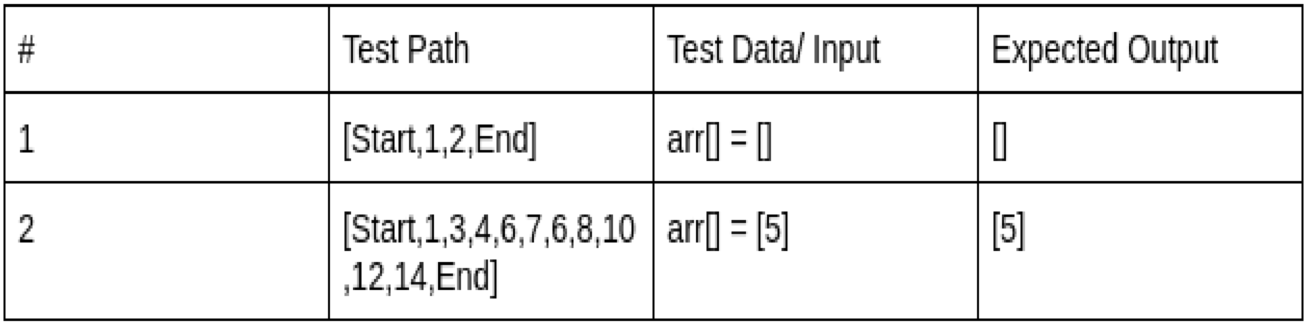


Lines	Block Number
7-11	1
12	2
14-18	3
21	4
22	5
26	6
27	7
31	8
32-33	9
38	10
39	11
42-43	12
44	13
46	14

- CFG

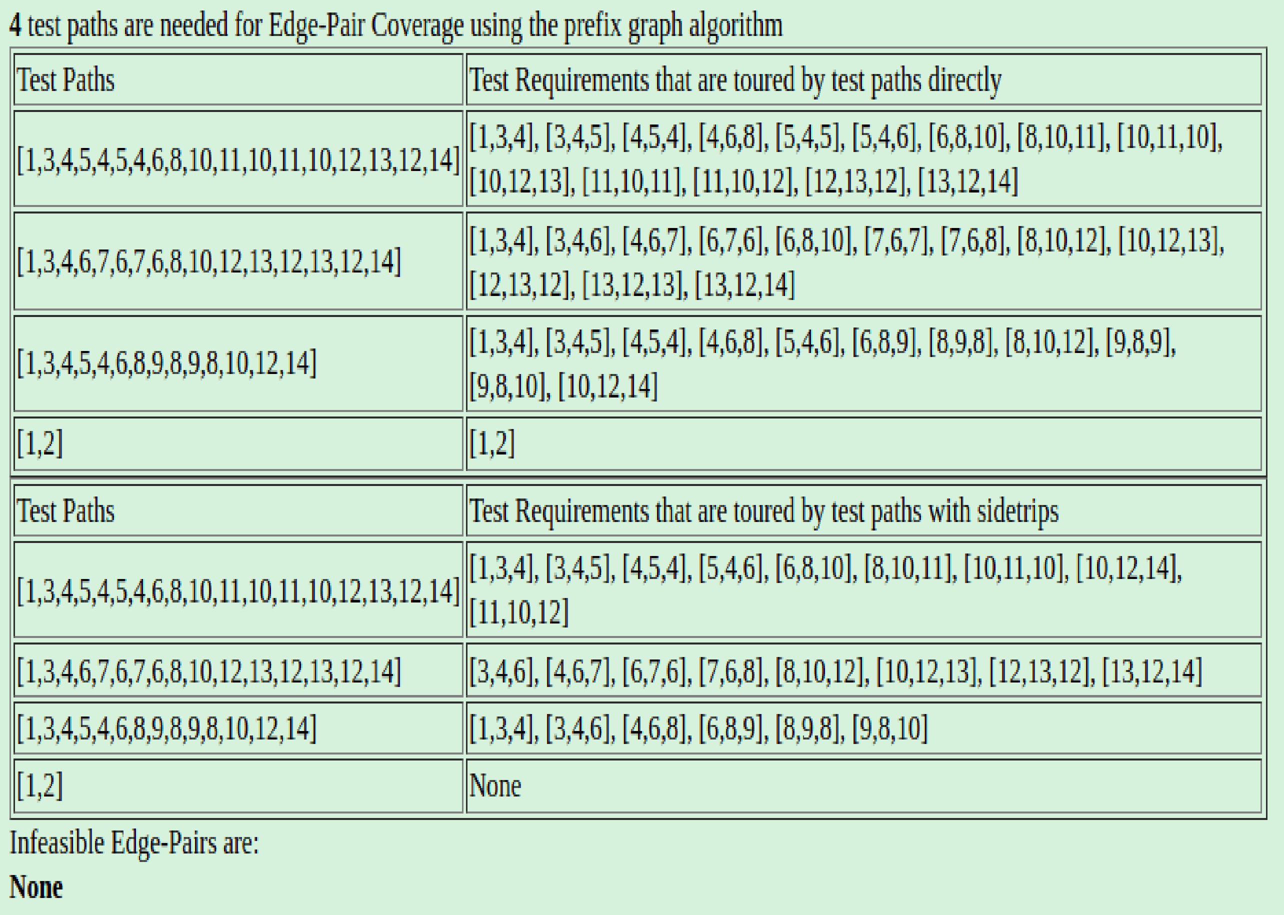

1
TRUE
FALSE
2
3
4
TRUE
FALSE
5
6
TRUE
FALSE
7
8
TRUE
FALSE
9
10
TRUE
FALSE
11
12
TRUE
FALSE
13
14

- Edge coverage



#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,2,End]	arr[] = []	[]
2	[Start,1,3,4,6,7,6,8,10,12,14,End]	arr[] = [5]	[5]

- Edge Pair coverage



4 test paths are needed for Edge-Pair Coverage using the prefix graph algorithm

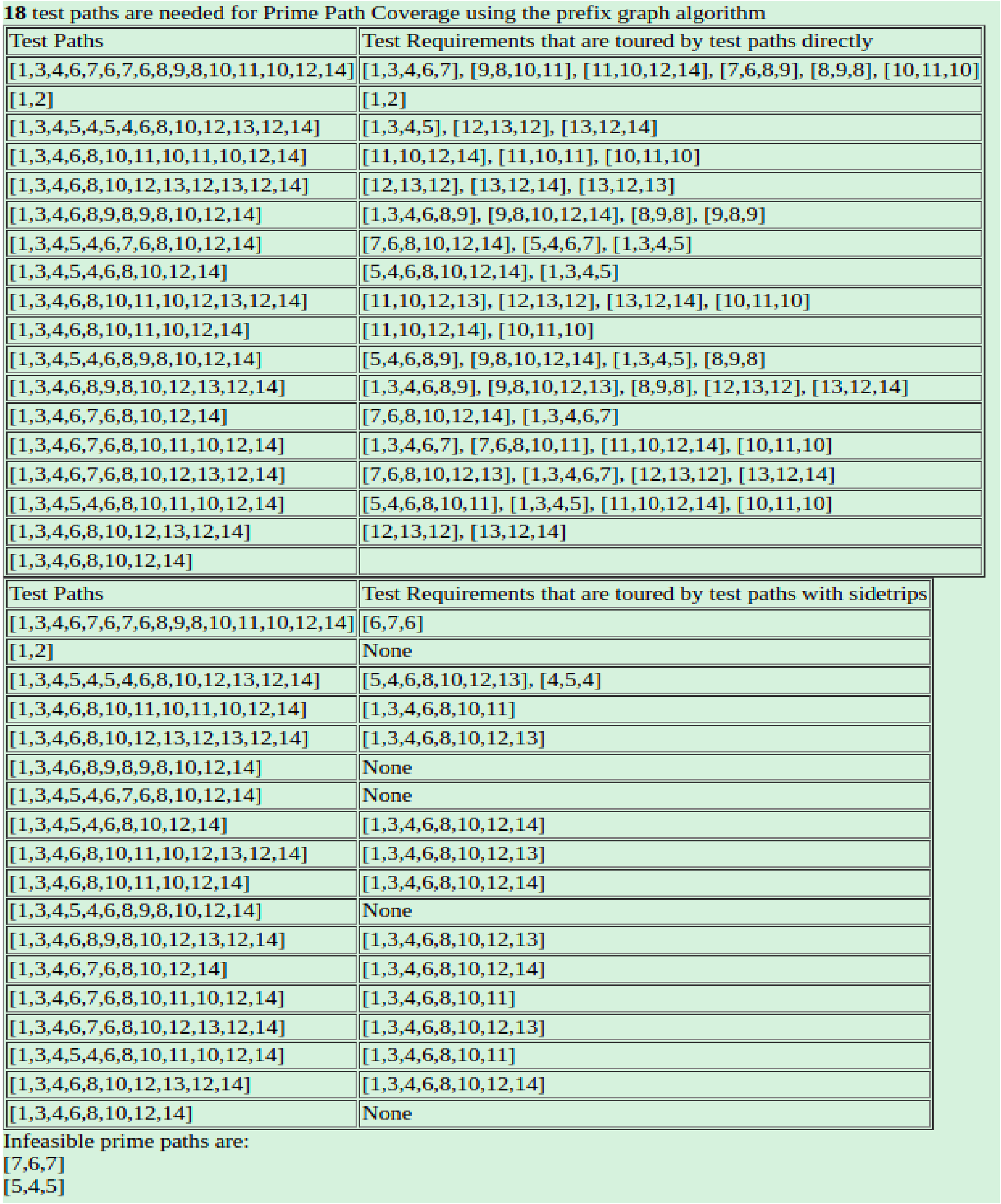
Test Paths	Test Requirements that are toured by test paths directly
[1,3,4,5,4,5,4,6,8,10,11,10,11,10,12,13,12,14]	[1,3,4], [3,4,5], [4,5,4], [4,6,8], [5,4,5], [5,4,6], [6,8,10], [8,10,11], [10,11,10], [10,12,13], [11,10,11], [11,10,12], [12,13,12], [13,12,14]
[1,3,4,6,7,6,7,6,8,10,12,13,12,13,12,14]	[1,3,4], [3,4,6], [4,6,7], [6,7,6], [6,8,10], [7,6,7], [7,6,8], [8,10,12], [10,12,13], [12,13,12], [13,12,13], [13,12,14]
[1,3,4,5,4,6,8,9,8,9,8,10,12,14]	[1,3,4], [3,4,5], [4,5,4], [4,6,8], [5,4,6], [6,8,9], [8,9,8], [8,10,12], [9,8,9], [9,8,10], [10,12,14]
[1,2]	[1,2]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,3,4,5,4,5,4,6,8,10,11,10,11,10,12,13,12,14]	[1,3,4], [3,4,5], [4,5,4], [5,4,6], [6,8,10], [8,10,11], [10,11,10], [10,12,14], [11,10,12]
[1,3,4,6,7,6,7,6,8,10,12,13,12,13,12,14]	[3,4,6], [4,6,7], [6,7,6], [7,6,8], [8,10,12], [10,12,13], [12,13,12], [13,12,14]
[1,3,4,5,4,6,8,9,8,9,8,10,12,14]	[1,3,4], [3,4,6], [4,6,8], [6,8,9], [8,9,8], [9,8,10]
[1,2]	None

Infeasible Edge-Pairs are:

None

- Prime Path coverage



18 test paths are needed for Prime Path Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,3,4,6,7,6,7,6,8,9,8,10,11,10,12,14]	[1,3,4,6,7], [9,8,10,11], [11,10,12,14], [7,6,8,9], [8,9,8], [10,11,10]
[1,2]	[1,2]
[1,3,4,5,4,5,4,6,8,10,12,13,12,14]	[1,3,4,5], [12,13,12], [13,12,14]
[1,3,4,6,8,10,11,10,11,10,12,14]	[11,10,12,14], [11,10,11], [10,11,10]
[1,3,4,6,8,10,12,13,12,13,12,14]	[12,13,12], [13,12,14], [13,12,13]
[1,3,4,6,8,9,8,9,8,10,12,14]	[1,3,4,6,8,9], [9,8,10,12,14], [8,9,8], [9,8,9]
[1,3,4,5,4,6,7,6,8,10,12,14]	[7,6,8,10,12,14], [5,4,6,7], [1,3,4,5]
[1,3,4,5,4,6,8,10,12,14]	[5,4,6,8,10,12,14], [1,3,4,5]
[1,3,4,6,8,10,11,10,12,13,12,14]	[11,10,12,13], [12,13,12], [13,12,14], [10,11,10]
[1,3,4,6,8,10,11,10,12,14]	[11,10,12,14], [10,11,10]
[1,3,4,5,4,6,8,9,8,10,12,14]	[5,4,6,8,9], [9,8,10,12,14], [1,3,4,5], [8,9,8]
[1,3,4,6,8,9,8,10,12,13,12,14]	[1,3,4,6,8,9], [9,8,10,12,13], [8,9,8], [12,13,12], [13,12,14]
[1,3,4,6,7,6,8,10,12,14]	[7,6,8,10,12,14], [1,3,4,6,7]
[1,3,4,6,7,6,8,10,11,10,12,14]	[1,3,4,6,7], [7,6,8,10,11], [11,10,12,14], [10,11,10]
[1,3,4,6,7,6,8,10,12,13,12,14]	[7,6,8,10,12,13], [1,3,4,6,7], [12,13,12], [13,12,14]
[1,3,4,5,4,6,8,10,11,10,12,14]	[5,4,6,8,10,11], [1,3,4,5], [11,10,12,14], [10,11,10]
[1,3,4,6,8,10,12,13,12,14]	[12,13,12], [13,12,14]
[1,3,4,6,8,10,12,14]	

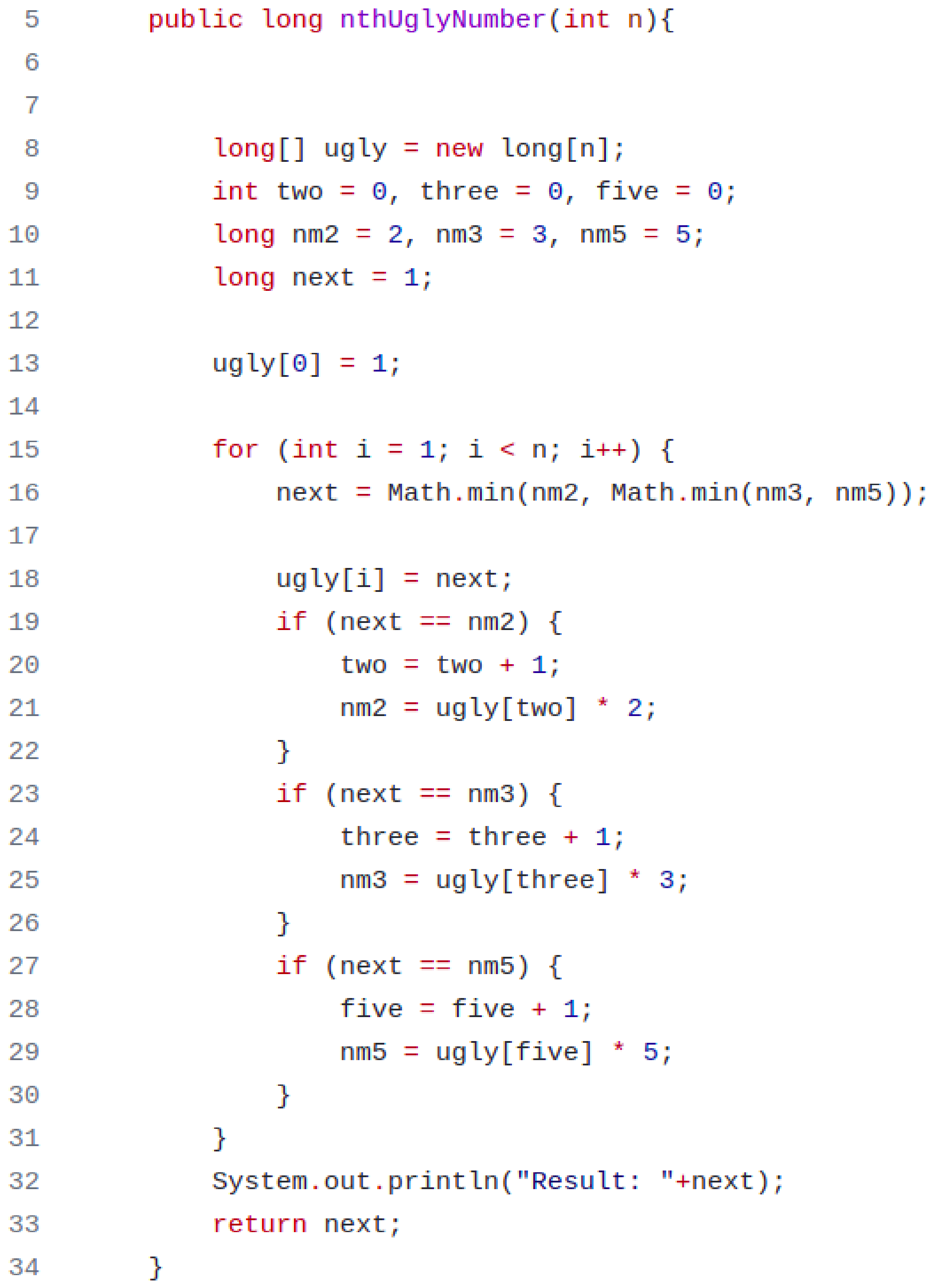
Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,3,4,6,7,6,7,6,8,9,8,10,11,10,12,14]	[6,7,6]
[1,2]	None
[1,3,4,5,4,5,4,6,8,10,12,13,12,14]	[5,4,6,8,10,12,13], [4,5,4]
[1,3,4,6,8,10,11,10,11,10,12,14]	[1,3,4,6,8,10,11]
[1,3,4,6,8,10,12,13,12,13,12,14]	[1,3,4,6,8,10,12,13]
[1,3,4,6,8,9,8,9,8,10,12,14]	None
[1,3,4,5,4,6,7,6,8,10,12,14]	None
[1,3,4,5,4,6,8,10,12,14]	[1,3,4,6,8,10,12,14]
[1,3,4,6,8,10,11,10,12,13,12,14]	[1,3,4,6,8,10,12,13]
[1,3,4,6,8,10,11,10,12,14]	[1,3,4,6,8,10,12,14]
[1,3,4,5,4,6,8,9,8,10,12,14]	None
[1,3,4,6,8,9,8,10,12,13,12,14]	[1,3,4,6,8,10,12,13]
[1,3,4,6,7,6,8,10,12,14]	[1,3,4,6,8,10,12,14]
[1,3,4,6,7,6,8,10,11,10,12,14]	[1,3,4,6,8,10,11]
[1,3,4,6,7,6,8,10,12,13,12,14]	[1,3,4,6,8,10,12,13]
[1,3,4,5,4,6,8,10,11,10,12,14]	[1,3,4,6,8,10,11]
[1,3,4,6,8,10,12,13,12,14]	[1,3,4,6,8,10,12,14]
[1,3,4,6,8,10,12,14]	None

Infeasible prime paths are:
[7,6,7]
[5,4,5]

5. Misc

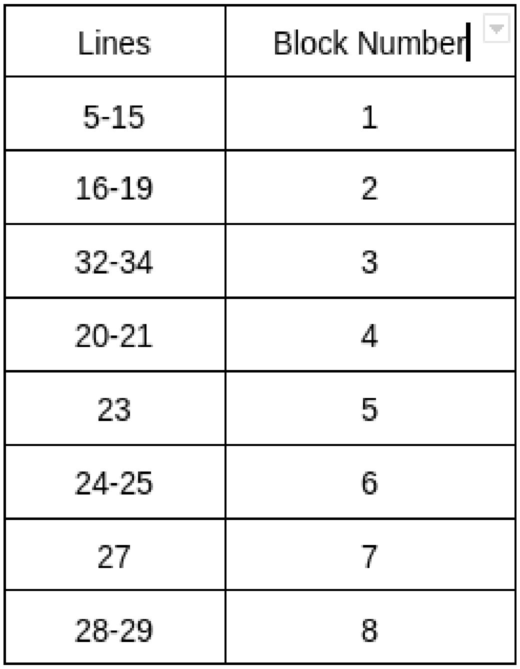
NthUglyNumber.java

- Code



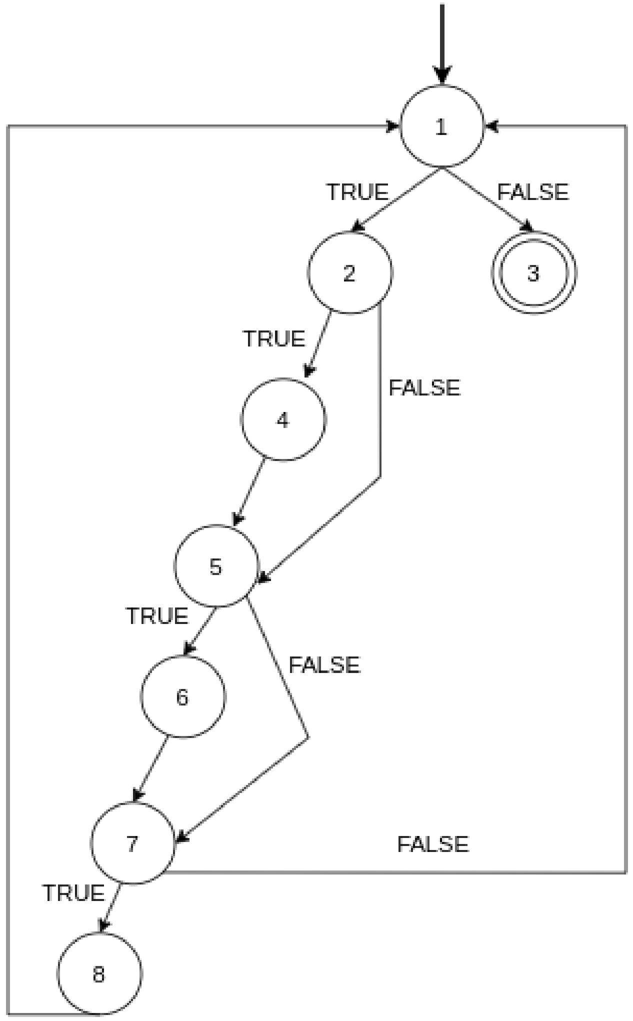
```java
    public long nthUglyNumber(int n){


        long[] ugly = new long[n];
        int two = 0, three = 0, five = 0;
        long nm2 = 2, nm3 = 3, nm5 = 5;
        long next = 1;

        ugly[0] = 1;

        for (int i = 1; i < n; i++) {
            next = Math.min(nm2, Math.min(nm3, nm5));

            ugly[i] = next;
            if (next == nm2) {
                two = two + 1;
                nm2 = ugly[two] * 2;
            }
            if (next == nm3) {
                three = three + 1;
                nm3 = ugly[three] * 3;
            }
            if (next == nm5) {
                five = five + 1;
                nm5 = ugly[five] * 5;
            }
        }
        System.out.println("Result: "+next);
        return next;
    }
```

- Basic Block



Lines	Block Number
5-15	1
16-19	2
32-34	3
20-21	4
23	5
24-25	6
27	7
28-29	8

- CFG

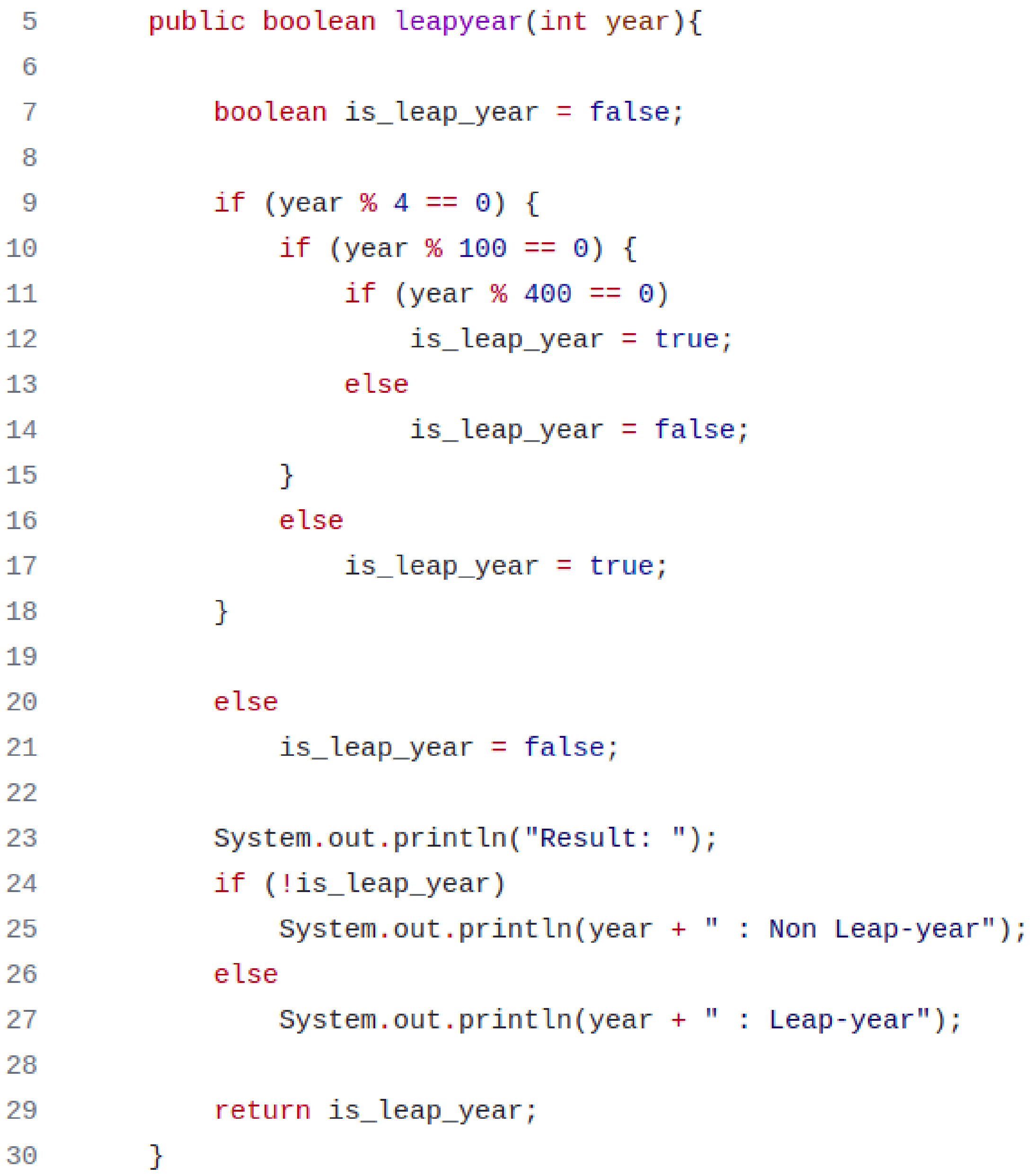


- Edge coverage

#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,3, End]	n = 1	1
2	[Start, 1,2,4,5,7,1,3,End]	n = 2	2
3	[Start,1,2,4,5,7,1,2,5, 6,7,1,3,End]	n = 3	3

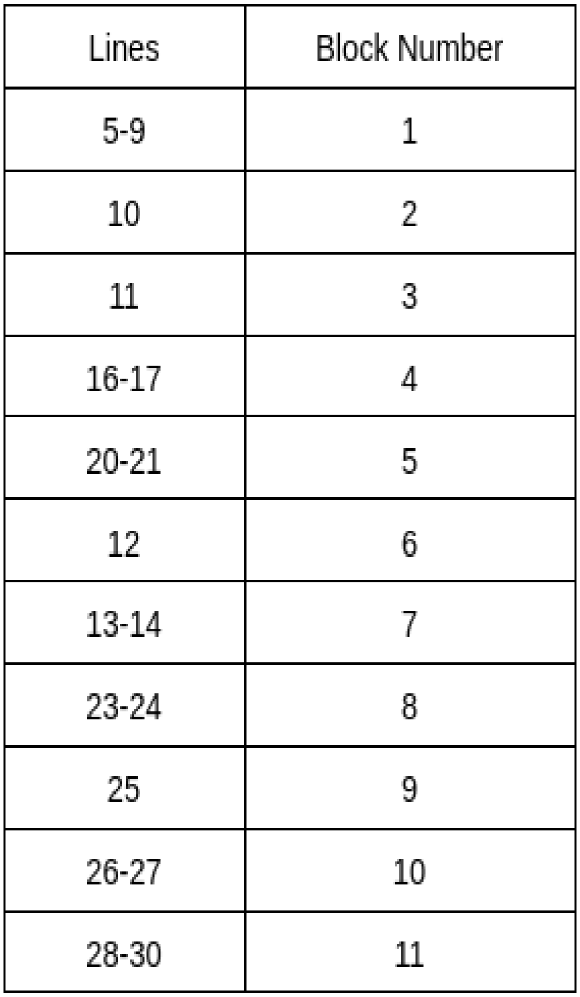
LeapYear.java

- Code



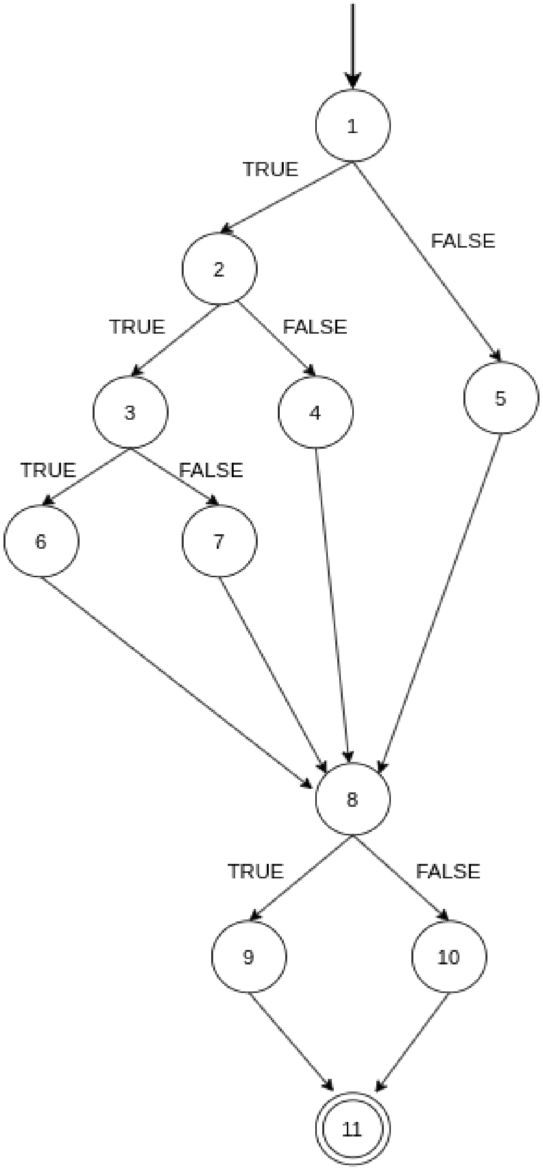
```java
public boolean leapyear(int year){

    boolean is_leap_year = false;

    if (year % 4 == 0) {
        if (year % 100 == 0) {
            if (year % 400 == 0)
                is_leap_year = true;
            else
                is_leap_year = false;
        }
        else
            is_leap_year = true;
    }

    else
        is_leap_year = false;

    System.out.println("Result: ");
    if (!is_leap_year)
        System.out.println(year + " : Non Leap-year");
    else
        System.out.println(year + " : Leap-year");

    return is_leap_year;
}
```

- Basic Block



Lines	Block Number
5-9	1
10	2
11	3
16-17	4
20-21	5
12	6
13-14	7
23-24	8
25	9
26-27	10
28-30	11

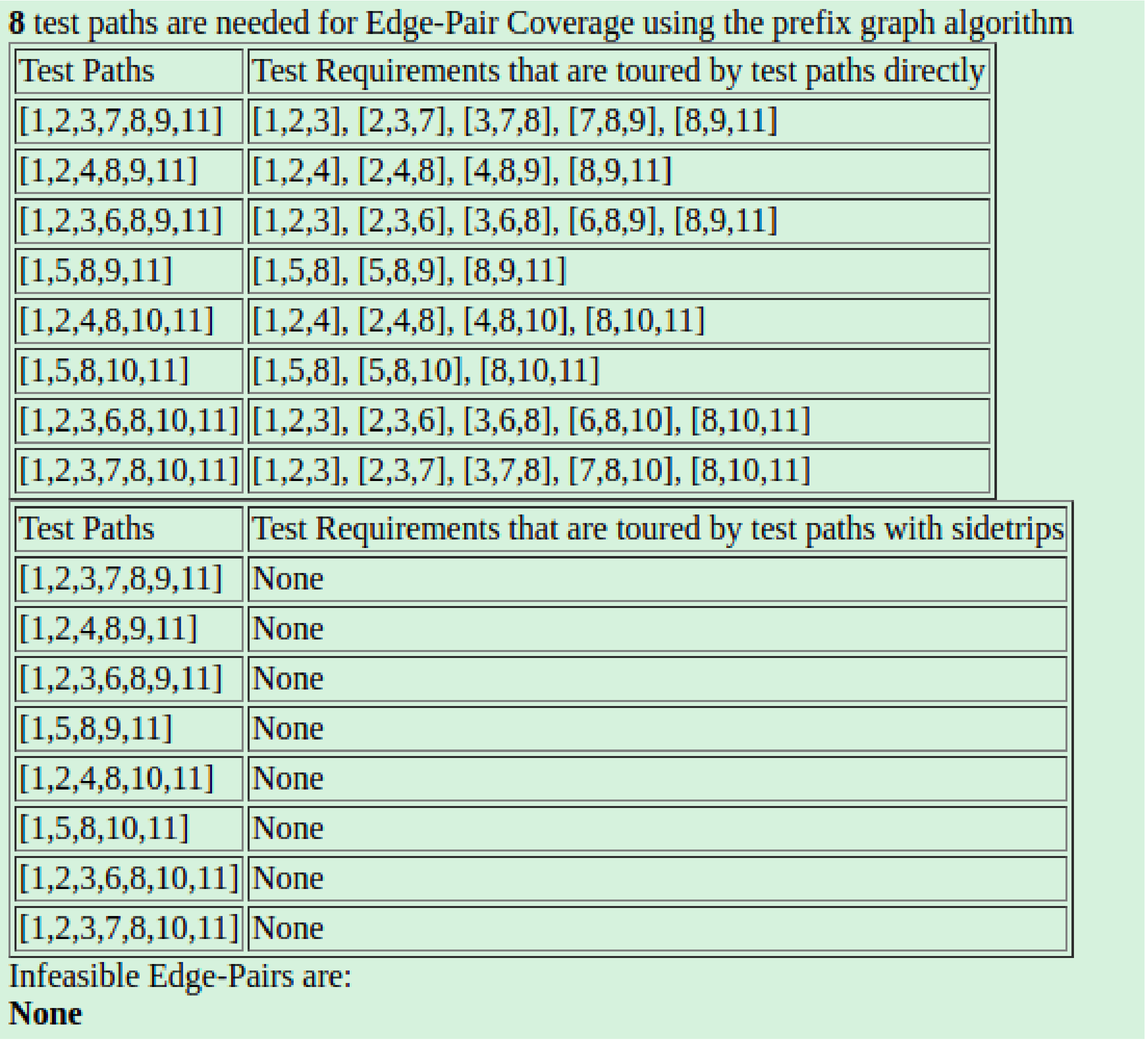
- CFG



- Edge coverage

#	Test Path	Test Data/ Input	Expected Output
1	[Start,1,2,3,6,8,9,11,End]	year = 2000	true
2	[Start,1,5,8,10,11,End]	year = 2001	false
3	[Start,1,2,4,8,10,11,End]	year = 2004	true
4	[Start,1,2,3,7,8,9,11,End]	year = 1900	false

- Edge Pair coverage



8 test paths are needed for Edge-Pair Coverage using the prefix graph algorithm

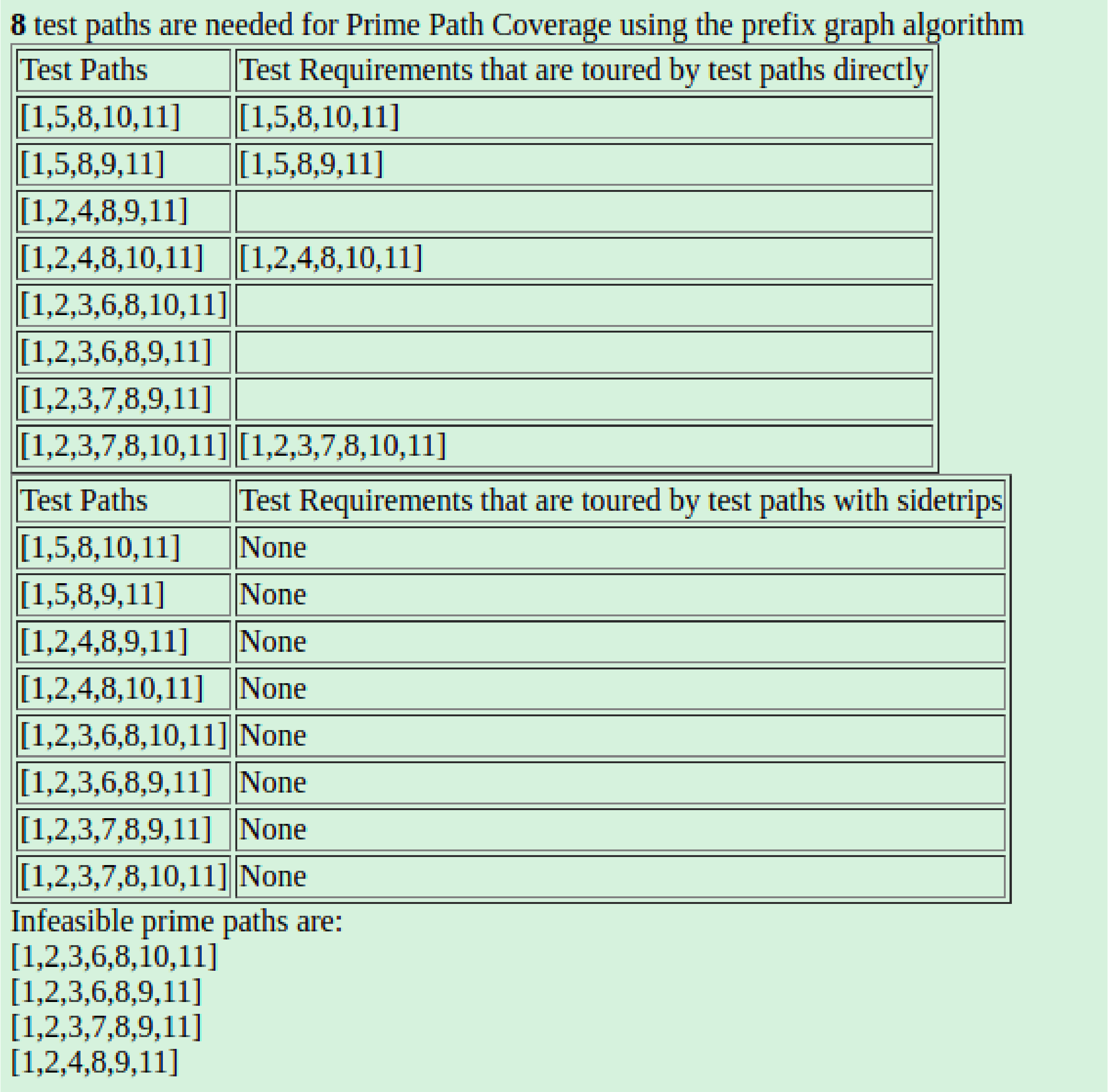
Test Paths	Test Requirements that are toured by test paths directly
[1,2,3,7,8,9,11]	[1,2,3], [2,3,7], [3,7,8], [7,8,9], [8,9,11]
[1,2,4,8,9,11]	[1,2,4], [2,4,8], [4,8,9], [8,9,11]
[1,2,3,6,8,9,11]	[1,2,3], [2,3,6], [3,6,8], [6,8,9], [8,9,11]
[1,5,8,9,11]	[1,5,8], [5,8,9], [8,9,11]
[1,2,4,8,10,11]	[1,2,4], [2,4,8], [4,8,10], [8,10,11]
[1,5,8,10,11]	[1,5,8], [5,8,10], [8,10,11]
[1,2,3,6,8,10,11]	[1,2,3], [2,3,6], [3,6,8], [6,8,10], [8,10,11]
[1,2,3,7,8,10,11]	[1,2,3], [2,3,7], [3,7,8], [7,8,10], [8,10,11]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,2,3,7,8,9,11]	None
[1,2,4,8,9,11]	None
[1,2,3,6,8,9,11]	None
[1,5,8,9,11]	None
[1,2,4,8,10,11]	None
[1,5,8,10,11]	None
[1,2,3,6,8,10,11]	None
[1,2,3,7,8,10,11]	None

Infeasible Edge-Pairs are:
None

-

- Prime Path coverage



8 test paths are needed for Prime Path Coverage using the prefix graph algorithm

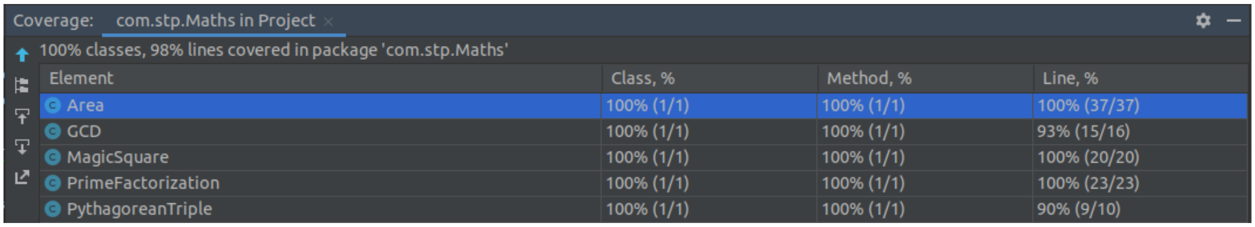
Test Paths	Test Requirements that are toured by test paths directly
[1,5,8,10,11]	[1,5,8,10,11]
[1,5,8,9,11]	[1,5,8,9,11]
[1,2,4,8,9,11]	
[1,2,4,8,10,11]	[1,2,4,8,10,11]
[1,2,3,6,8,10,11]	
[1,2,3,6,8,9,11]	
[1,2,3,7,8,9,11]	
[1,2,3,7,8,10,11]	[1,2,3,7,8,10,11]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,5,8,10,11]	None
[1,5,8,9,11]	None
[1,2,4,8,9,11]	None
[1,2,4,8,10,11]	None
[1,2,3,6,8,10,11]	None
[1,2,3,6,8,9,11]	None
[1,2,3,7,8,9,11]	None
[1,2,3,7,8,10,11]	None

Infeasible prime paths are:
[1,2,3,6,8,10,11]
[1,2,3,6,8,9,11]
[1,2,3,7,8,9,11]
[1,2,4,8,9,11]

Results

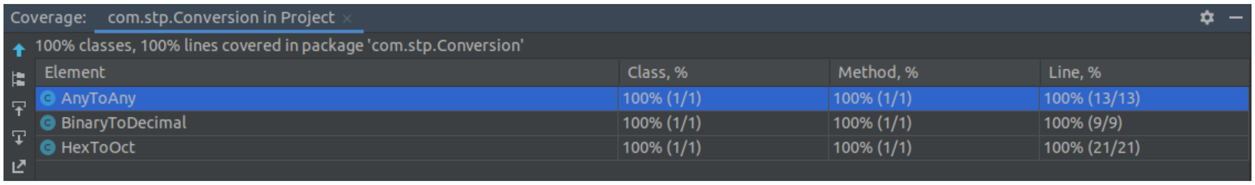
- Class Coverage



Coverage: com.stp.Maths in Project

100% classes, 98% lines covered in package 'com.stp.Maths'

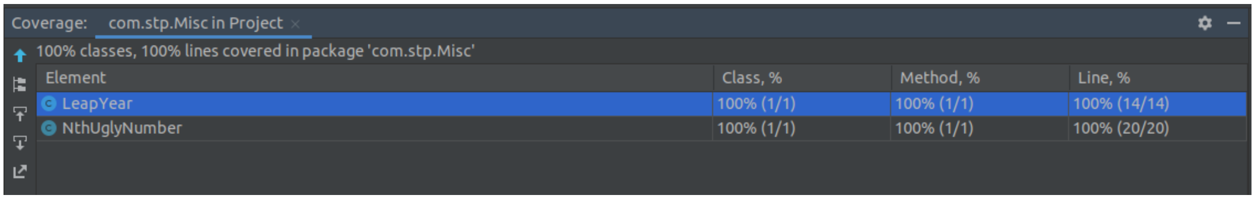
Element	Class, %	Method, %	Line, %
Area	100% (1/1)	100% (1/1)	100% (37/37)
GCD	100% (1/1)	100% (1/1)	93% (15/16)
MagicSquare	100% (1/1)	100% (1/1)	100% (20/20)
PrimeFactorization	100% (1/1)	100% (1/1)	100% (23/23)
PythagoreanTriple	100% (1/1)	100% (1/1)	90% (9/10)



Coverage: com.stp.Conversion in Project

100% classes, 100% lines covered in package 'com.stp.Conversion'

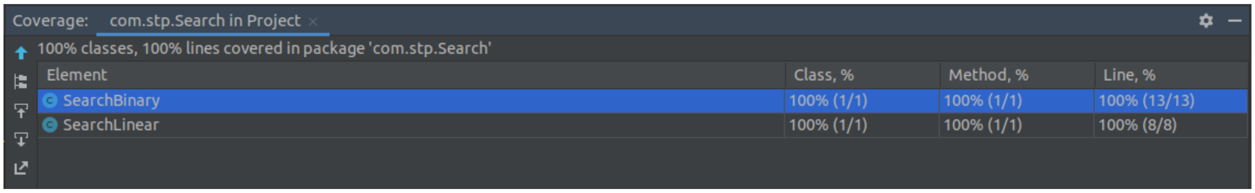
Element	Class, %	Method, %	Line, %
AnyToAny	100% (1/1)	100% (1/1)	100% (13/13)
BinaryToDecimal	100% (1/1)	100% (1/1)	100% (9/9)
HexToOct	100% (1/1)	100% (1/1)	100% (21/21)



Coverage: com.stp.Misc in Project

100% classes, 100% lines covered in package 'com.stp.Misc'

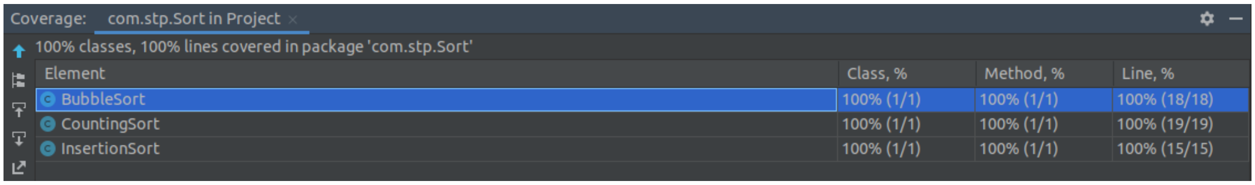
Element	Class, %	Method, %	Line, %
LeapYear	100% (1/1)	100% (1/1)	100% (14/14)
NthUglyNumber	100% (1/1)	100% (1/1)	100% (20/20)



Coverage: com.stp.Search in Project

100% classes, 100% lines covered in package 'com.stp.Search'

Element	Class, %	Method, %	Line, %
SearchBinary	100% (1/1)	100% (1/1)	100% (13/13)
SearchLinear	100% (1/1)	100% (1/1)	100% (8/8)

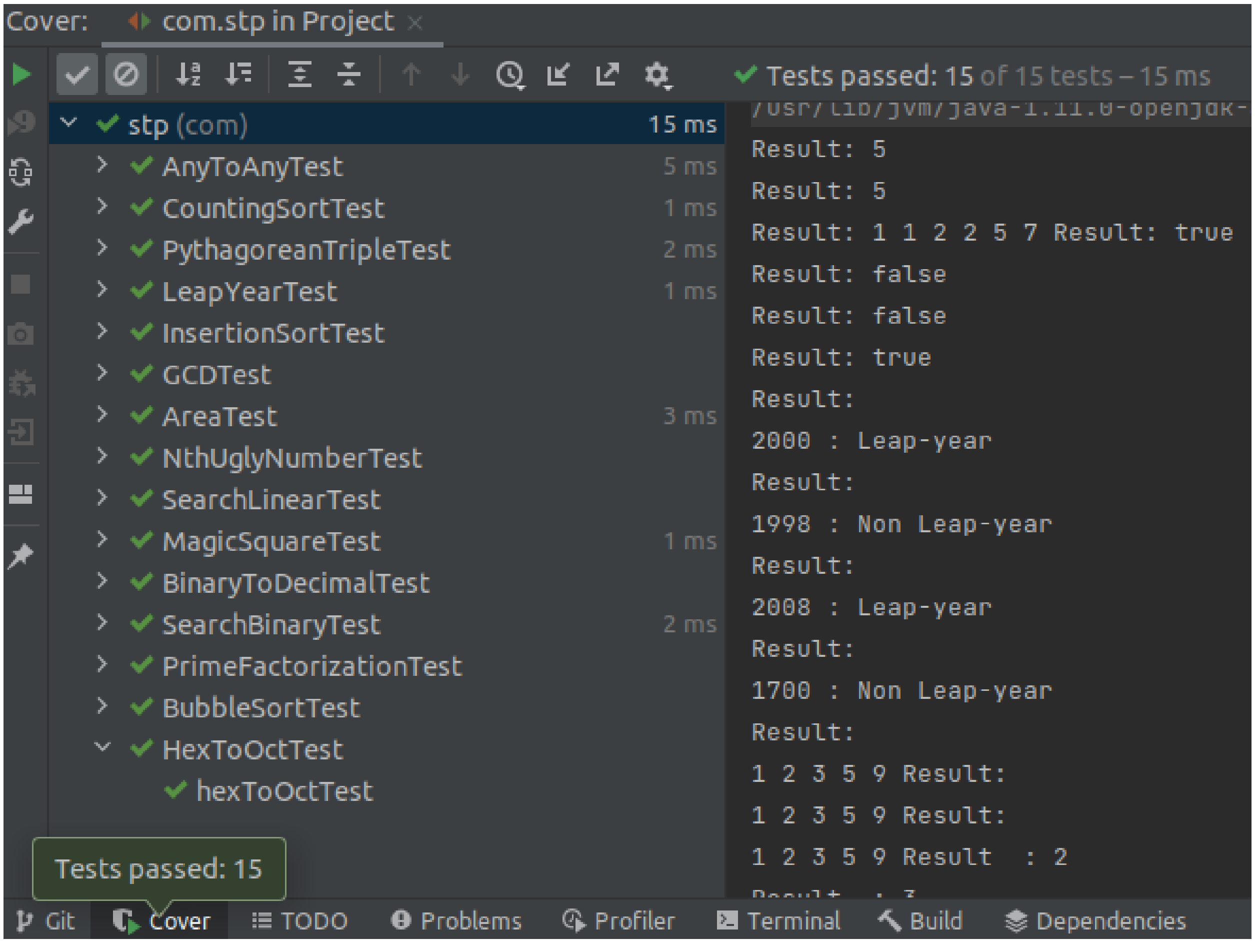


Coverage: com.stp.Sort in Project

100% classes, 100% lines covered in package 'com.stp.Sort'

Element	Class, %	Method, %	Line, %
BubbleSort	100% (1/1)	100% (1/1)	100% (18/18)
CountingSort	100% (1/1)	100% (1/1)	100% (19/19)
InsertionSort	100% (1/1)	100% (1/1)	100% (15/15)

Test results



```
✓ Tests passed: 15 of 15 tests – 15 ms
```

```
✓ stp (com)                    15 ms
  ✓ AnyToAnyTest                5 ms
  ✓ CountingSortTest            1 ms
  ✓ PythagoreanTripleTest       2 ms
  ✓ LeapYearTest                1 ms
  ✓ InsertionSortTest
  ✓ GCDTest
  ✓ AreaTest                    3 ms
  ✓ NthUglyNumberTest
  ✓ SearchLinearTest
  ✓ MagicSquareTest             1 ms
  ✓ BinaryToDecimalTest
  ✓ SearchBinaryTest            2 ms
  ✓ PrimeFactorizationTest
  ✓ BubbleSortTest
  ✓ HexToOctTest
    ✓ hexToOctTest
```

```
/usr/lib/jvm/java-1.11.0-openjdk-
Result: 5
Result: 5
Result: 1 1 2 2 5 7 Result: true
Result: false
Result: false
Result: true
Result:
2000 : Leap-year
Result:
1998 : Non Leap-year
Result:
2008 : Leap-year
Result:
1700 : Non Leap-year
Result:
1 2 3 5 9 Result:
1 2 3 5 9 Result:
1 2 3 5 9 Result  : 2
Result  : 3
```

Tests passed: 15