International Institute of Information Technology, Bangalore

Software Testing CS 731

Project Report

Solution By:
Ashutosh Soni (MT2021026)
Manthan Shah (MT2021121)
Shivani Sheth (MT2021126)

Introduction

Control Flow Graph

A Control Flow Graph (CFG) is the graphical representation of control flow or computation during the execution of programs or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside of a program unit.

Characteristics of Control Flow Graph:

- Control flow graph is process oriented.
- Control flow graph shows all the paths that can be traversed during a program execution.
- Control flow graph is a directed graph.
- Edges in CFG portray control flow paths and the nodes in CFG portray basic blocks.

Projects that involve testing of some basic math, sorting and stack based problems by designing test cases using the control flow graph based code coverage.

Testing Strategy

- Edge coverage
- Prime path coverage

Tools Used

- Junit
- https://cs.gmu.edu:8443/offutt/coverage/GraphCoverage

Code

GitHub link: https://github.com/ShivaniSheth/Software-Testing

Directory structure:

Name	Description
Math	Has solutions for math related coding problems
Sorting	Has solutions for sorting related coding problems
Stack	Has solutions for stack related coding problems

Testing Approach

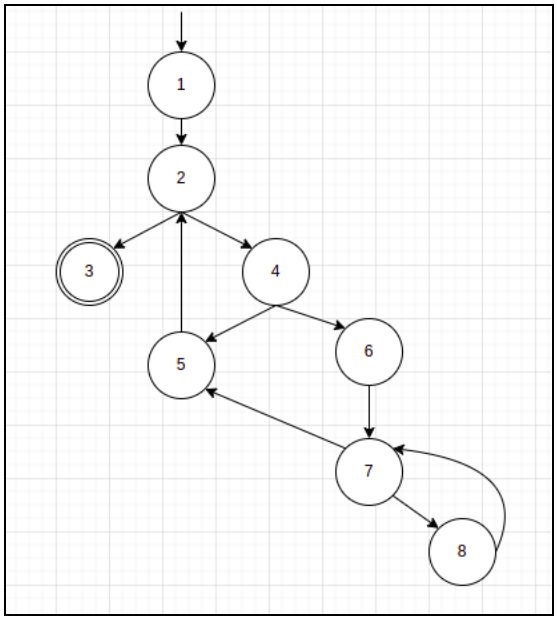
Math *(Done by Manthan Shah - MT2021121)*

1. **Count Primes**

```
3   public class CountPrimes {
4       public int countPrimes(int n) {
5           boolean[] notPrime = new boolean[n];
6           int count = 0;
7           for (int i = 2; i < n; i++) {
8               if (notPrime[i] == false) {
9                   count++;
10                  for (int j = 2; i*j < n; j++) {
11                      notPrime[i*j] = true;
12                  }
13              }
14          }
15
16          return count;
17      }
18  }
```

Control Flow Graph:

Lines	Nodes
5-7	1
7	2
16	3
8	4
7	5
9-10	6
10	7
10-11	8



Edge Coverage:

Test Requirements:
[1,2], [2,3], [2,4],[4,5], [4,6], [5,2], [6,7], [7,5], [7,8], [8,7]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,4,5,2,3]	3	1	Pass
2	[1,2,4,6,7,8,7,5,2,3]	5	2	Pass

Prime Path Coverage:

Test Requirements:
[4,6,7,5,2,3], [4,6,7,5,2,4], [2,4,6,7,5,2], [1,2,4,6,7,5], [1,2,4,6,7,8], [5,2,4,6,7,5], [7,5,2,4,6,7], [8,7,5,2,4,6] ,[6,7,5,2,4,6], [5,2,4,6,7,8], [8,7,5,2,3], [4,5,2,3], [4,5,2,4], [1,2,4,5], [2,4,5,2], [5,2,4,5], [1,2,3], [7,8,7], [8,7,8]

Test path and Test Cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,3]	2	0	Pass

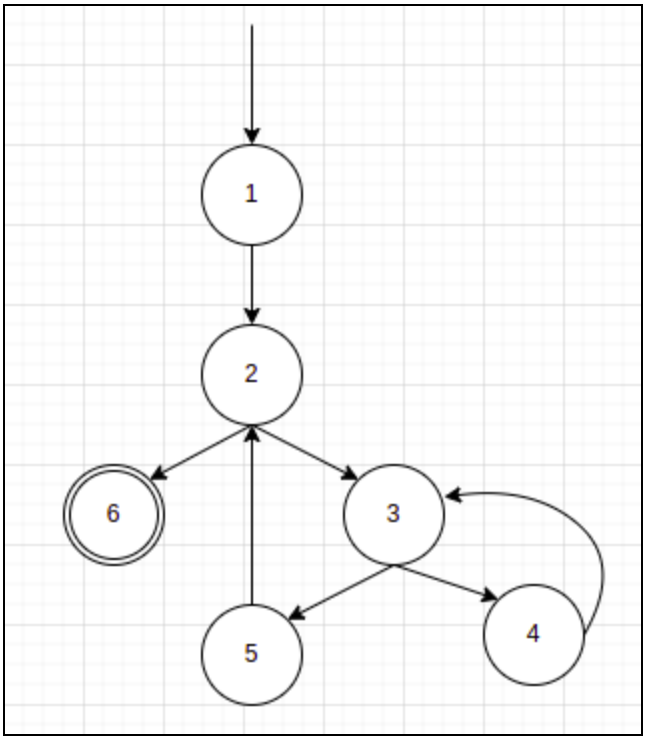
2	[1,2,4,6,7,5,2,4,6,7,5,2,3]	3	1	Pass

2. Integer to Romans

```java
public class IntegerToRoman {
    public String intToRoman(int num) {

        String romanWord[] = {"I", "IV", "V", "IX", "X", "XL", "L", "XC", "C", "CD", "D", "CM", "M"};
        int value[] = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000};
        int i = romanWord.length - 1;
        String sol = "";

        while(num>0){

            while(value[i]<=num){
                System.out.println("*");
                sol += romanWord[i];
                num -= value[i];

            }
            i--;
        }
        System.out.println(sol);
        return sol;
    }
}
```

Control flow graph:

Lines	Nodes
6-9	1
11	2
13	3
15-16	4
19	5
22	6



Edge Coverage:

Test Requirement:
[1,2], [2,6], [2,3], [3,5], [5,2], [3,4], [4,3]

Test paths and Test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,3,4,3,5,2,6]	1000	“M”	Pass

Prime path Coverage:

Test Requirement:
[4,3,5,2,6], [2,3,5,2], [1,2,3,4], [1,2,3,5], [3,5,2,3], [5,2,3,4], [5,2,3,5], [1,2,6], [4,3,4], [3,4,3]

Test paths and Test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,3,5,2,3,4,3,5,2,6]	900	“CM”	Pass
2	[1,2,6]	0	“”	Pass

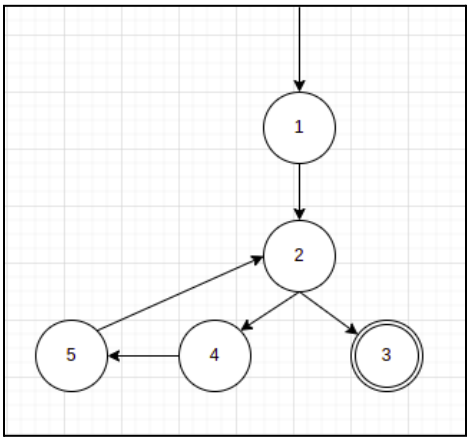
3	[1,2,3,4,3,4,3,5,2,6]	1900	“MCM”	Pass

3. **Find the missing no.**

```java
public class MissingNo {
    public int missingNumber(int[] nums) {
        int sum=0;
        int n = nums.length;
        for(int i=0;i<nums.length;i++){
            sum+=nums[i];
        }
        System.out.println( (n*(n+1)/2 - sum)  );
        return n*(n+1)/2 - sum;
    }
}
```

Control flow graph:

Lines	Nodes
5-7	1
7	2
10	3
8	4
7	5



Edge Coverage:

Test Requirement:
[1,2], [2,3], [2,4], [4,5], [5,2]

Test path and Test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,4,5,2,3]	[0]	1	Pass

Prime Path Coverage:

Test Requirement:
[2,4,5,2], [1,2,4,5], [4,5,2,3], [5,2,4,5], [4,5,2,4], [1,2,3]

Test paths and test cases[code]:

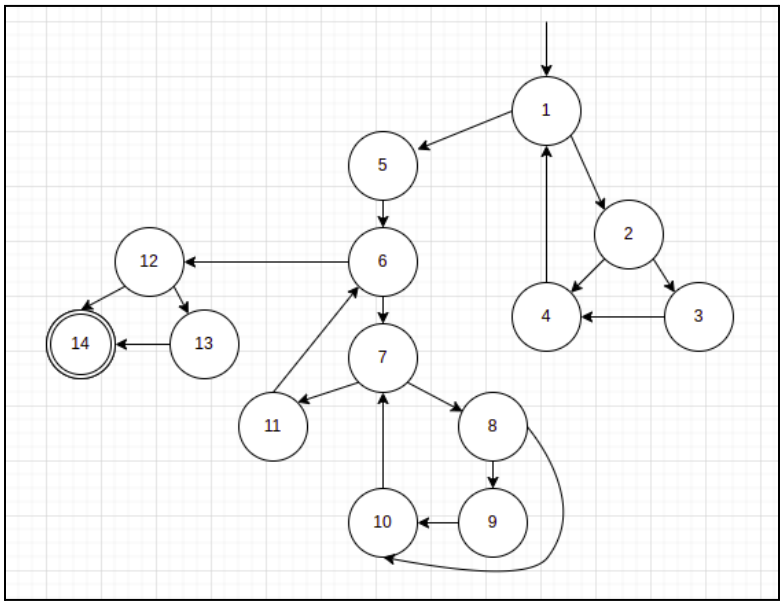
Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,4,5,2,4,5,2,3]	[1,0]	2	Pass
2	[1,2,3]	[]	0	Pass

4. Prime Factorization

```java
public class PrimeFactorization {

    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;
```

Control flow graph:

Lines	Nodes
9-14	1
15	2
16-17	3
19	4
22-23	5
23	6
24	7
25	8
26-27	9
29	10
33	12
34	13
37-42	14



Edge Coverage:

Test Requirement:
[1,2], [1,5], [2,3], [2,4], [3,4], [4,1], [5,6], [6,7], [6,12], [7,11], [7,8], [8,9], [9,10], [8,10], [10,7], [11,6], [12,13], [12,14], [13,14]

Test path and Test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,5,6,12,14]	1	[]	Pass
2	[1,2,4,3,1,5,6,12,13,14]	6	[2,3]	Pass
3	[1,5,6,7,8,9,10,7,8,10,7,11,6,12,14]	30	[2,3,5]	Pass

Prime Path Coverage:

Test Requirement:

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

Test paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,5,6,7,8,10,7,8,9,10,7,8,10,7,11,6,12,14]	70	[2,5,7]	Pass
2	[1,2,3,4,1,2,4,1,5,6,12,14]	14	[2, 7]	Pass
3	[1,2,3,4,1,2,3,4,1,5,6,12,14]	4	[2]	Pass
4	[1,2,3,4,1,2,4,1,5,6,12,14]	20	[2,5]	Pass
5	[1,5,6,7,8,9,10,7,8,9,10,7,11,6,12,14]	105	[3,5,7]	Pass

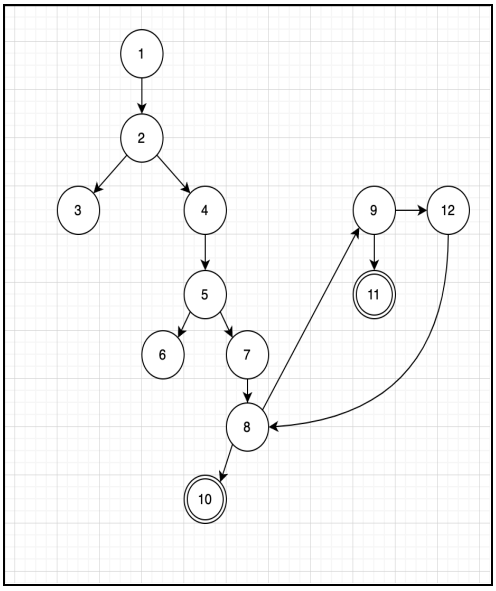
Sorting *(Done by Ashutosh Soni - MT2021026)*

1. **Find difference**

```
3  public class findDifference {
4      public char finddifference(String s, String t)
5      {
6          int[] freq = new int[26];
7
8          for(int i=0; i<s.length();i++)
9              freq[s.charAt(i) - 'a']++;
10
11         for(int i=0; i<t.length();i++)
12             freq[t.charAt(i) - 'a']--;
13
14         for(int i=0; i<26;i++)
15         {
16             if(freq[i] != 0)
17                 return (char)(i+97);
18         }
19         return 'a'; // The control would not reach here at any cost, so returned a random character here.
20     }
21 }
```

Control flow graph:

Lines	Node
6, 7, 8	1
8	2
9	3
11	4
11	5
12	6
14	7
14	8
16	9
19	10
17	11
14	12



Edge Coverage:

Test Requirements:
[1,2], [2,3], [3,2], [2,4], [4,5], [5,6], [6,5], [5,7], [7,8], [8,10], [8,9], [9,11], [9,12], [12,8]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,4,5,7,8,9,12,8,10]	String s= ”” String t= ””	“a”	Pass
2	[1,2,3,2,4,5,6,5,7,8,9,11]	String s= ”abcd” String t= ”abcde”	“e”	Pass
3	[1,2,3,4,5,7,8,9,12,8,10]	String s= ”” String t= ””	“a”	Pass

Prime Path Coverage:

Test Requirements:
[1,2,4,5,7,8,9,12], [1,2,4,5,7,8,9,11], [3,2,4,5,7,8,9,11], [3,2,4,5,7,8,9,12], [3,2,4,5,7,8,10], [1,2,4,5,7,8,10], [6,5,7,8,9,12], [6,5,7,8,9,11], [3,2,4,5,6], [1,2,4,5,6], [6,5,7,8,10], [9,12,8,9], [12,8,9,11], [12,8,9,12], [9,12,8,10], [8,9,12,8], [5,6,5], [1,2,3], [2,3,2], [3,2,3], [6,5,6]

Test Paths and test cases[code]:

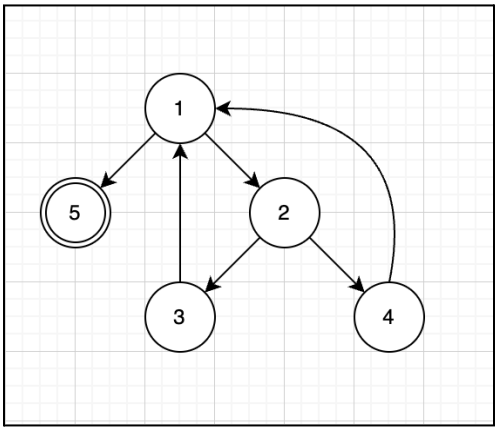
Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,3,2,4,5,7,8,9,11]	String s= ”” String t= ”y”	“y”	Pass
2	[1,2,3,4,5,7,8,9,12,8,10]	String s= ”” String t= ””	“a”	Pass
3	[1,2,3,2,3,2,3,2,4,5,7,8,10]	String s= ”abc” String t= ””	Failure	Pass
4	[1,2,4,5,6,5,7,8,9,11]	String s= ”” String t= ”z”	“z”	Pass
5	[1,2,3,2,4,5,6,5,7,8,9,11]	String s= ”abc” String t= ”abce”	“e”	Pass
6	[1,2,3,2,4,5,6,5,6,5,7,8,9,12,8,9,11]	String s= ”b” String t= ”ba”	“a”	Pass
7	[1,2,3,2,4,5,6,5,6,5,7,8,9,12,8,9,12,8,9,11]	String s= ”a” String t= ”ab”	“b”	Pass

2. **Sort array by parity**

```
3   public class sortArraybyParity {
4       public int[] sortarrayByparity(int[] nums) {
5           int p1 = 0;
6           int p2 = 0;
7           while(p2<nums.length){
8               if(nums[p2]%2==0){
9                   int temp = nums[p1];
10                  nums[p1++] = nums[p2];
11                  nums[p2++] = temp;
12              }
13              else p2++;
14          }
15          return nums;
16      }
17  }
```

Control flow graph:

Lines	Node
5, 6, 7	1
8	2
13	3
9, 10, 11	4



Edge Coverage:

Test Requirements:
[1,2], [2,3], [2,4], [3,1], [4,1], [1,5]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,4,1,5]	nums = [2,3,8,9]	[2,8,3,9]	Pass
2	[1,2,3,1,5]	nums = [3,1,2,4]	[2,4,3,1]	Pass

Prime Path Coverage:

Test Requirements:
[2,3,1,5], [2,4,1,2], [2,3,1,2], [1,2,3,1], [1,2,4,1], [2,4,1,5], [4,1,2,3], [4,1,2,4], [3,1,2,4], [3,1,2,3]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,4,1,2,4,1,5]	nums = [2,4]	[2,4]	Pass
2	[1,2,3,1,2,4,1,5]	nums = [1,4]	[4,1]	Pass
3	[1,2,4,1,2,3,1,5]	nums = [4,9]	[4,9]	Pass
4	[1,2,3,1,2,3,1,5]	nums = [11,9]	[11,9]	Pass

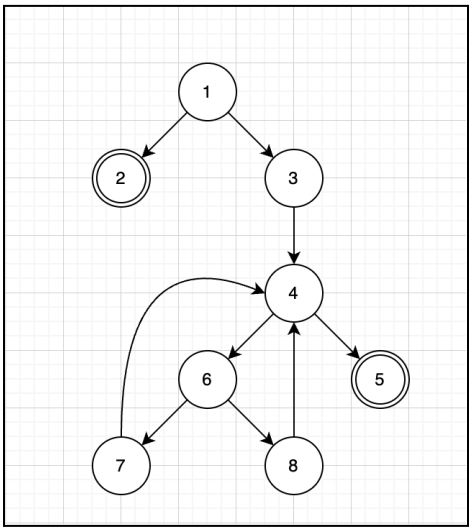
3. **Square of sorted array**

```
3   public class squaresofSortedArray {
4       public  int[] sortedSquares(int[] nums) {
5           int length = nums.length;
6           if (length == 0) return new int[0];
7
8           int[] result = new int[length];
9           int l = 0, r = length - 1;
10          for (int i = length - 1; i >= 0; i--) {
11              int start = nums[l] * nums[l];
12              int end = nums[r] * nums[r];
13              if (start > end) {
14                  result[i] = start;
15                  l++;
16              } else {
17                  result[i] = end;
18                  r--;
19              }
20          }
21          return result;
22      }
23  }
```

Control flow graph:

Lines	Node
5, 6	1

6	2
8, 9, 10	3
10	4
21	5
11, 12, 13	6
14, 15	7
17, 18	8



Edge Coverage:

Test Requirements:
[1,2], [1,3], [3,4], [4,5], [4,6], [6,7], [6,8], [7,4], [8,4]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2]	nums=[]	[]	Pass
2	[1,3,4,6,8,4,5]	nums=[3]	[9]	Pass
3	[1,3,4,6,7,4,5]	nums=[-1,3,7]	[1,9,49]	Pass

Prime Path Coverage:

Test Requirements:
[1,3,4,6,7], [1,3,4,6,8], [4,6,7,4], [4,6,8,4], [6,7,4,5], [1,3,4,5], [6,7,4,6], [7,4,6,8], [8,4,6,7], [8,4,6,8], [7,4,6,7], [6,8,4,5], [6,8,4,6], [1,2]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,3,4,5]	nums=[]	[]	Pass
2	[1,3,4,6,7,4,6,8,4,5]	nums=[3]	[9]	Pass
3	[1,3,4,6,8,4,6,7,4,5]	nums=[3,6]	[9,36]	Pass
4	[1,3,4,6,8,4,6,8,4,5]	nums=[2,5,7]	[4,25,49]	Pass
5	[1,3,4,6,7,4,6,7,4,5]	nums=[-4,1,2]	[1,4,16]	Pass
6	[1,2]	nums=[]	[]	Pass

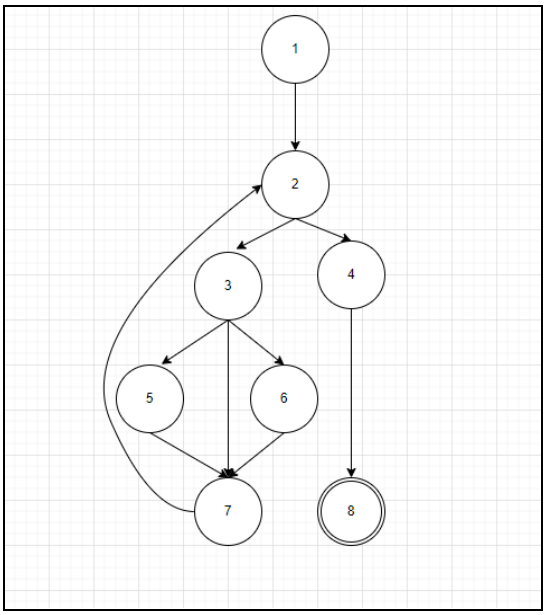
Stack *(Done by Shivani Sheth - MT2021126)*

1. **maxDepth**

```
3  public class maxDepth {
4      public int maxdepth(String s) {
5          if(s=="("||s==")")return 0;//not valid parenthesis string
6          int depth = 0;
7          int max = 0;
8          for(int i = 0;i < s.length();i++) {
9              if(s.charAt(i) == '(') {
10                 depth++;
11             } else if(s.charAt(i) == ')') {
12                 depth--;
13             }
14             max = depth > max ? depth : max;
15         }
16
17         return max;
18     }
19 }
```

Control flow graph:

Lines	Node
6,7,8	1
8	2
9	3
8	4
10	5
12	6
14	7
16	8



Edge Coverage:

Test Requirements:
[1,2][2,3],[2,4],[3,5],[3,6],[5,7],[6,7],[7,2],[4,8],[3,7]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,3,5,7,2,4,8]	"("	0	Pass
2	[1,2,3,6,7,2,4,8]	")"	0	Pass

Prime Path Coverage:

Test Requirements:
[3,5,7,2,4,8], [3,6,7,2,4,8], [3,5,7,2,3], [3,6,7,2,3], [2,3,6,7,2], [1,2,3,5,7], [1,2,3,6,7],[2,3,5,7,2], [5,7,2,3,5], [7,2,3,5,7], [7,2,3,6,7], [3,7,2,4,8], [6,7,2,3,6],[5,7,2,3,6], [6,7,2,3,5], [2,3,7,2], [1,2,3,7], [1,2,4,8], [7,2,3,7], [3,7,2,3]

Test Paths and test cases[code]:

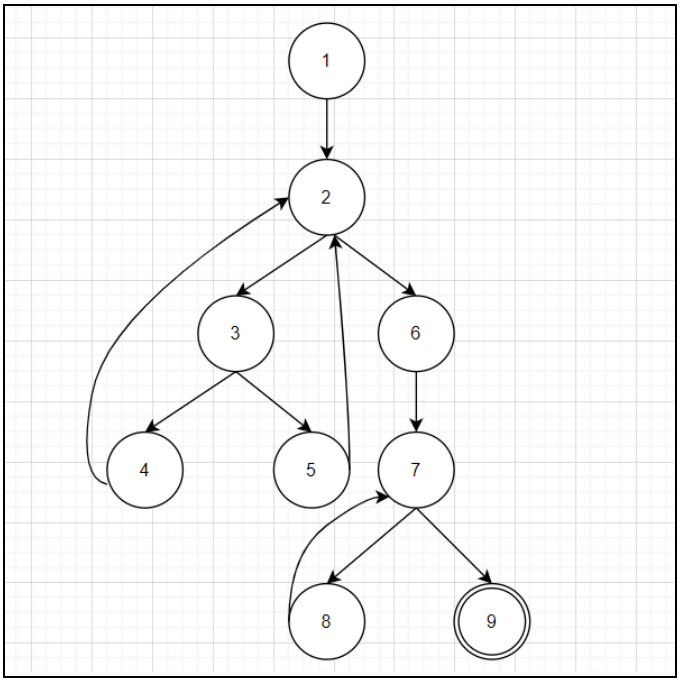
Sr. No.	Test Path	Input	Expected Output	Status
1	[1,2,3,5,7,2,3,6,7,2,4,8]	"()"	1	Pass
2	[1,2,4,8]	""	0	Pass

2. **Remove Duplicates**

```
3   import java.util.Stack;
4
5   public class removeDuplicate {
6       public String removeduplicates(String s) {
7           Stack<Character> st = new Stack<>();
8           st.push(s.charAt(0));
9           for(int i = 1; i<s.length(); i++){
10              if(!st.isEmpty() && st.peek() == s.charAt(i)){
11                  st.pop(); continue;
12              }
13              st.push(s.charAt(i));
14          }
15          StringBuilder sb = new StringBuilder();
16          while(!st.isEmpty()){
17              sb.append(st.pop());
18          }
19          return String.valueOf(sb.reverse());
20      }
21  }
```

Control flow graph:

Lines	Nodes
7,8	1
9	2
10	3
11	4
13	5
15	6
16	7
17	8
19	9



Edge Coverage:

Test Requirements:
[1,2],[2,3],[2,6],[3,4],[3,5],[4,2],[5,2],[6,7],[7,8],[7,9],[8,7]

Test Paths and test cases[code]:

Sr. No.	Test path	Input	Expected Output	Status
1	[1,2,3,4,2,6,7,9]	“aa”	“”	Pass

Prime Path Coverage:

Test Requirements:
[3,4,2,6,7,9], [3,4,2,6,7,8], [3,5,2,6,7,8], [3,5,2,6,7,9], [1,2,6,7,9], [1,2,6,7,8], [2,3,5,2],[3,4,2,3],[2,3,4,2], [1,2,3,4], [1,2,3,5], [5,2,3,4], [5,2,3,5], [4,2,3,5], [3,5,2,3], [4,2,3,4], [7,8,7], [8,7,8], [8,7,9]

Test Paths and test cases[code]:

Sr. No.	Test Path	Input	Expected Output	Status
1	[1,2,3,5,2,6,7,8,7,9]	“ab”	“ab”	Pass

2	[1,2,6,7,9]	“a”	“a”	Pass
3	[1,2,3,5,2,3,4,2,6,7,9]	“abb”	“a”	Pass
4	1,2,3,5,2,3,5,2,6,7,9]	“aba”	“aba”	Pass
5	[1,2,3,4,2,3,4,2,6,7,9]	“abba”	“”	Pass

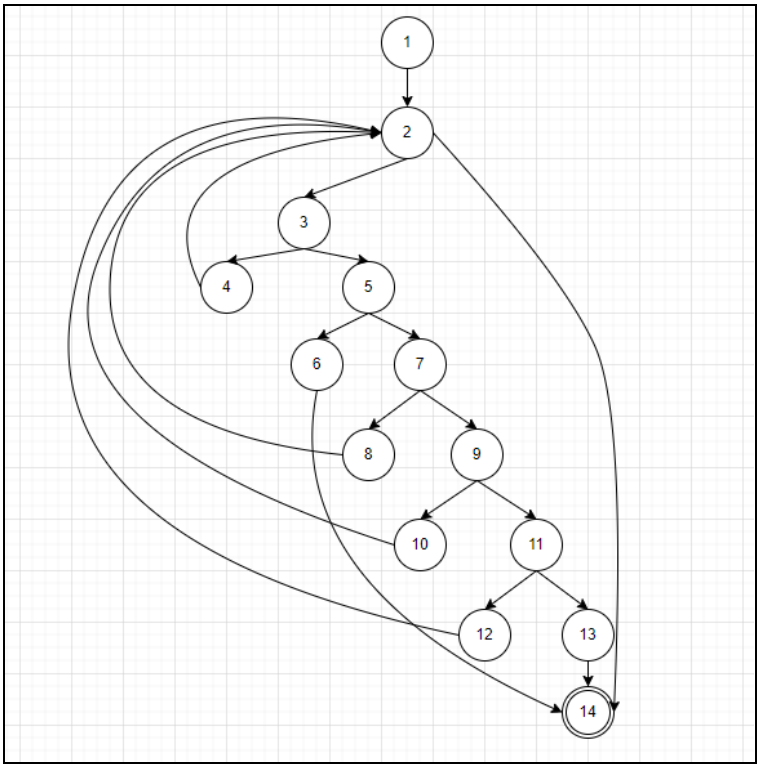
3. Valid Parentheses

```java
import java.util.Stack;

public class validParenthesis {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<Character>();
        for (int i = 0; i < s.length(); i ++) {
            char ele = s.charAt(i);
            if (s.charAt(i) == '(' || s.charAt(i) == '[' || s.charAt(i) == '{') {
                stack.push(ele);
            }
            else{
                if (stack.isEmpty())
                    return false;
                if (ele == ')' && stack.peek() == '(')
                    stack.pop();
                else if (ele == '}' && stack.peek() == '{')
                    stack.pop();
                else if (ele == ']' && stack.peek() == '[')
                    stack.pop();
                else
                    break;
            }}

        return stack.isEmpty();
    }
}
```

Control flow graph:

Lines	Node
7,8	1
8	2
9,10	3
11	4

14	5
15	6
16	7
17	8
18	9
19	10
20	11
21	12
22	13
25	14



Edge Coverage:

Test Requirements:
[1,2], [2,3], [3,4], [3,5], [5,6], [5,7], [7,8], [7,9], [9,10], [9,11], [11,12], [11,13], [13,14], [2,14], [6,14], [4,2], [8,2], [10,2], [12,2]

Test Paths and test cases[code]:

Sr.	Test path	Input	Expected Output	Status
1	[1,2,3,4,2,14]	“(“	false	Pass

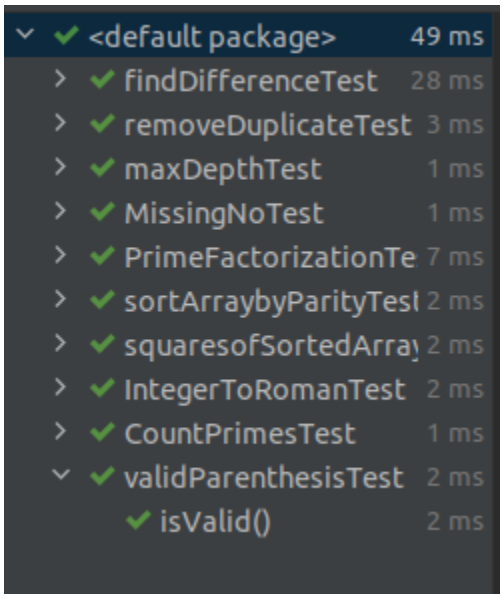
Prime Path Coverage:

Test Requirements:
[4,2,3,5,7,9,11,13,14], [7,9,11,12,2,3,5,6,14], [1,2,3,5,7,9,11,13,14],
[8,2,3,5,7,9,11,13,14],[12,2,3,5,7,9,11,13,14], [10,2,3,5,7,9,11,13,14],
[7,9,10,2,3,5,6,14],[5,7,9,11,12,2,3,5], [5,7,9,11,12,2,3,4], [7,9,11,12,2,3,5,7],
[3,5,7,9,11,12,2,14], [1,2,3,5,7,9,11,12], [2,3,5,7,9,11,12,2],
[3,5,7,9,11,12,2,3],[9,11,12,2,3,5,7,8], [10,2,3,5,7,9,11,12],
[12,2,3,5,7,9,11,12],[8,2,3,5,7,9,11,12], [11,12,2,3,5,7,9,10],
[9,11,12,2,3,5,7,9],[11,12,2,3,5,7,9,11], [4,2,3,5,7,9,11,12],[5,7,9,10,2,3,5],
[5,7,9,10,2,3,4],[7,8,2,3,5,6,14], [7,9,10,2,3,5,7], [2,3,5,7,9,10,2],
[1,2,3,5,7,9,10],[3,5,7,9,10,2,14], [3,5,7,9,10,2,3], [8,2,3,5,7,9,10],
[4,2,3,5,7,9,10],[10,2,3,5,7,9,10], [9,10,2,3,5,7,9], [9,10,2,3,5,7,8], [5,7,8,2,3,5],
[5,7,8,2,3,4],[7,8,2,3,5,7], [2,3,5,7,8,2], [1,2,3,5,7,8], [1,2,3,5,6,14], [3,5,7,8,2,14],
[3,5,7,8,2,3],[8,2,3,5,7,8], [4,2,3,5,7,8], [4,2,3,5,6,14], [2,3,4,2], [1,2,3,4], [3,4,2,14],
[3,4,2,3],[4,2,3,4], [1,2,14]

Test Paths and test cases[code]:

Sr. No.	Test Path	Input	Expected Output	Status
1	[1,2,3,4,2,3,5,7,9,11,13,14]	“)(”	false	Pass
2	[1,2,3,5,7,9,11,13,14]	“)”	false	Pass
3	[1,2,3,5,6,14]	“”	true	Pass
4	[1,2,3,4,2,3,4,2,14]	“((”	false	Pass
5	[1,2,14]	“”	true	Pass

Results




```
[INFO] Running org.example.math.CountPrimesTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.013 s - in org.example.math.CountPrimesTest
[INFO] Running org.example.math.IntegerToRomanTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.011 s - in org.example.math.IntegerToRomanTest
[INFO] Running org.example.Sorting.sortArraybyParityTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 s - in org.example.Sorting.sortArraybyParityTest
[INFO] Running org.example.Sorting.squaresofSortedArrayTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.01 s - in org.example.Sorting.squaresofSortedArrayTest
[INFO] Running org.example.Sorting.findDifferenceTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.012 s - in org.example.Sorting.findDifferenceTest
[INFO] Running org.example.Stack.removeDuplicateTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 s - in org.example.Stack.removeDuplicateTest
[INFO] Running org.example.Stack.maxDepthTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 s - in org.example.Stack.maxDepthTest
[INFO] Running org.example.Stack.validParenthesisTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 s - in org.example.Stack.validParenthesisTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  16.247 s
```