Software Testing Project

Akshil Gadhiya MT2020007

Alay Dhagia MT2020102

Source Code Description

- We have used All In one calculator code for this project which contains all functionality provided by programmers calculator.
- It is console based java application.
- The code provides following functionalities.
    - All basic arithmetic operations which includes Addition, Subtraction, Multiplication, Division, Modulo, Square, Square root, Multiplicative Inverse.
    - All advanced arithmetic operations which includes Log (Natural), Log (base b), Factorial, Permutations (nPr), Combinations (nCr), Calculate y th Power of x (x ^ y).
    - Conversion from one number system to another number system.
    - Area and volume of shapes

Testing strategy

- Data flow graph.
    - For each function we are first creating Control flow graph from code and from cfg we are creating dfg.
    - After creating dfg we are finding **du pairs**, **du paths** and **All du path coverage** with the help of https://cs.gmu.edu:8443/offutt/coverage/DFGraphCoverage.
    - Based on all du paths we are creating test cases and then testing using Junit.

Tools used

- Junit for test cases.
- https://cs.gmu.edu:8443/offutt/coverage/DFGraphCoverage for TR generation.

Below are some of the functions amongst the one we have tested.

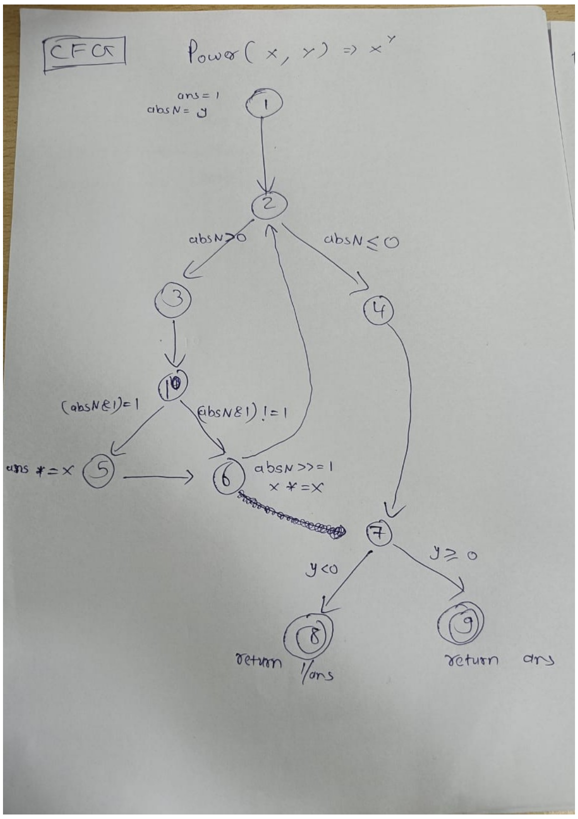
Power(x,y)

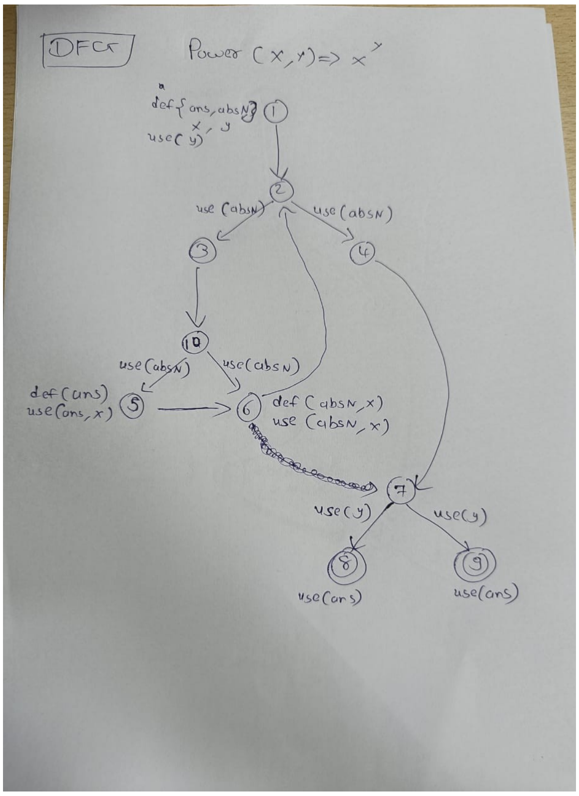
- This function is used to find yth power of x.

Code

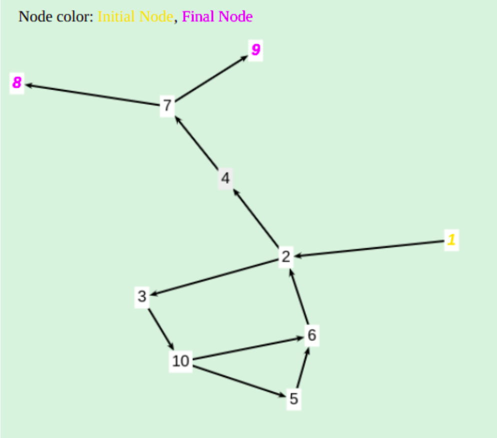
```
public double power(double x, int y) {
        double ans = 1;
        long absN = Math.abs((long)y);
        while(absN > 0) {
            if((absN&1)==1)
            {
                ans *= x;
            }
            absN >>= 1;
            x *= x;
        }
        if(y < 0)
        {
            return 1/ans;
        }
        else
        {
            return ans;
        }
    }
```

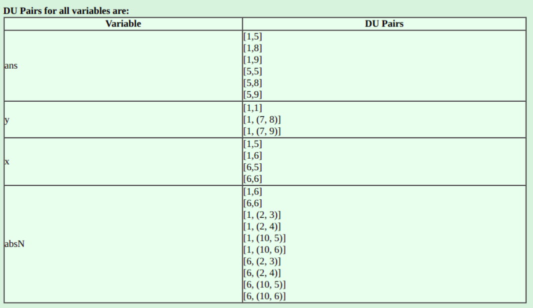
CFG & DFG





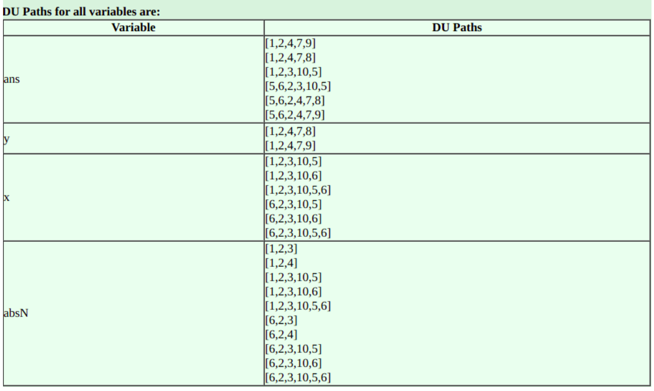
Graph, DU Pairs, Du Paths and All du path coverage





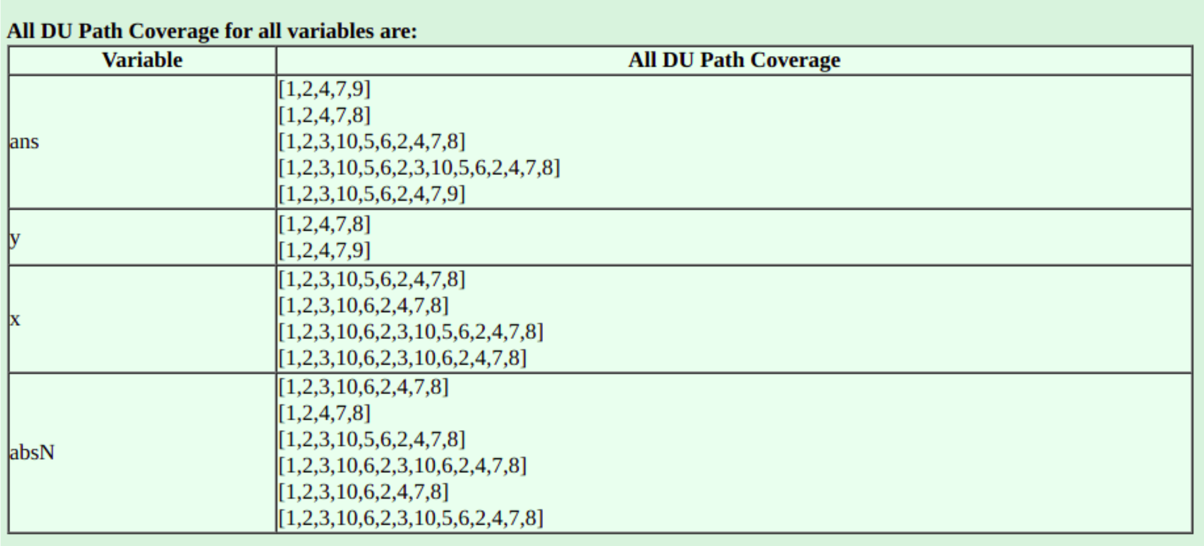
DU Pairs for all variables are:

Variable	DU Pairs
ans	[1,5] [1,8] [1,9] [5,5] [5,8] [5,9]
y	[1,1] [1, (7, 8)] [1, (7, 9)]
x	[1,5] [1,6] [6,5] [6,6]
absN	[1,6] [6,6] [1, (2, 3)] [1, (2, 4)] [1, (10, 5)] [1, (10, 6)] [6, (2, 3)] [6, (2, 4)] [6, (10, 5)] [6, (10, 6)]



DU Paths for all variables are:

Variable	DU Paths
ans	[1,2,4,7,9] [1,2,4,7,8] [1,2,3,10,5] [5,6,2,3,10,5] [5,6,2,4,7,8] [5,6,2,4,7,9]
y	[1,2,4,7,8] [1,2,4,7,9]
x	[1,2,3,10,5] [1,2,3,10,6] [1,2,3,10,5,6] [6,2,3,10,5] [6,2,3,10,6] [6,2,3,10,5,6]
absN	[1,2,3] [1,2,4] [1,2,3,10,5] [1,2,3,10,6] [1,2,3,10,5,6] [6,2,3] [6,2,4] [6,2,3,10,5] [6,2,3,10,6] [6,2,3,10,5,6]



All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
ans	[1,2,4,7,9] [1,2,4,7,8] [1,2,3,10,5,6,2,4,7,8] [1,2,3,10,5,6,2,3,10,5,6,2,4,7,8] [1,2,3,10,5,6,2,4,7,9]
y	[1,2,4,7,8] [1,2,4,7,9]
x	[1,2,3,10,5,6,2,4,7,8] [1,2,3,10,6,2,4,7,8] [1,2,3,10,6,2,3,10,5,6,2,4,7,8] [1,2,3,10,6,2,3,10,6,2,4,7,8]
absN	[1,2,3,10,6,2,4,7,8] [1,2,4,7,8] [1,2,3,10,5,6,2,4,7,8] [1,2,3,10,6,2,3,10,6,2,4,7,8] [1,2,3,10,6,2,4,7,8] [1,2,3,10,6,2,3,10,5,6,2,4,7,8]

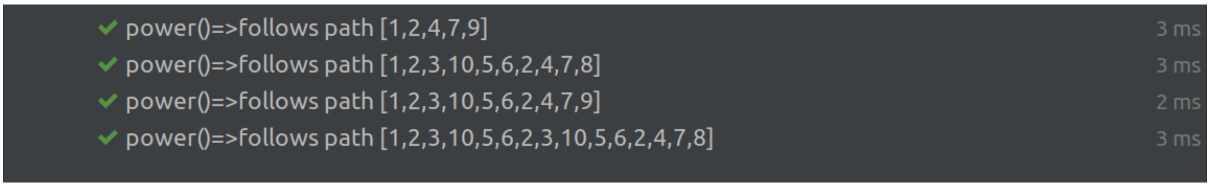
There are total 7 unique paths:

1. That skip the loop:
   - [1,2,4,7,9], [1,2,4,7,8]
     - Test case for [1,2,4,7,9]:
       - power(x,y) ⇒ power(2,0)

        - Expected output = 1
    - Test case for [1,2,4,7,8]:
        - Infeasible to cover this case because if y is -Ve then it will go inside the loop, Only one way to execute this code without loop is y==0 but then condition y<0 will never true. Hence we can not reach node 8 without execution loop.

2. That requires at least one iteration of loop:
    - [1,2,3,10,5,6,2,4,7,8], [1,2,3,10,5,6,2,4,7,9], [1,2,3,10,6,2,4,7,8]
        - Test case for [1,2,3,10,5,6,2,4,7,8]:
            - power(x,y) ⇒ power(2,-1)
            - Expected output = 0.5
        - Test case for [1,2,3,10,5,6,2,4,7,9]:
            - power(x,y) ⇒ power(2,1)
            - Expected output = 2
        - Test case for [1,2,3,10,6,2,4,7,8]:
            - Again this case is infeasible to cover because if (absN&1)≠1 then it requires one more iteration.
3. That requires at least two iteration of loop:
    - [1,2,3,10,5,6,2,3,10,5,6,2,4,7,8], [1,2,3,10,5,6,2,3,10,6,2,4,7,8]
        - Test case for [1,2,3,10,5,6,2,3,10,5,6,2,4,7,8]:
            - Power(x,y) ⇒ power(2,-3)
            - Expected output = 0.125
        - Test case for [1,2,3,10,5,6,2,3,10,5,6,2,4,7,8]:
            - once again this case is infeasible to cover because of the same reason above if (absN&1)≠1 then it requires one more iteration.

Result



```
✓ power()=>follows path [1,2,4,7,9]                          3 ms
✓ power()=>follows path [1,2,3,10,5,6,2,4,7,8]               3 ms
✓ power()=>follows path [1,2,3,10,5,6,2,4,7,9]               2 ms
✓ power()=>follows path [1,2,3,10,5,6,2,3,10,5,6,2,4,7,8]    3 ms
```

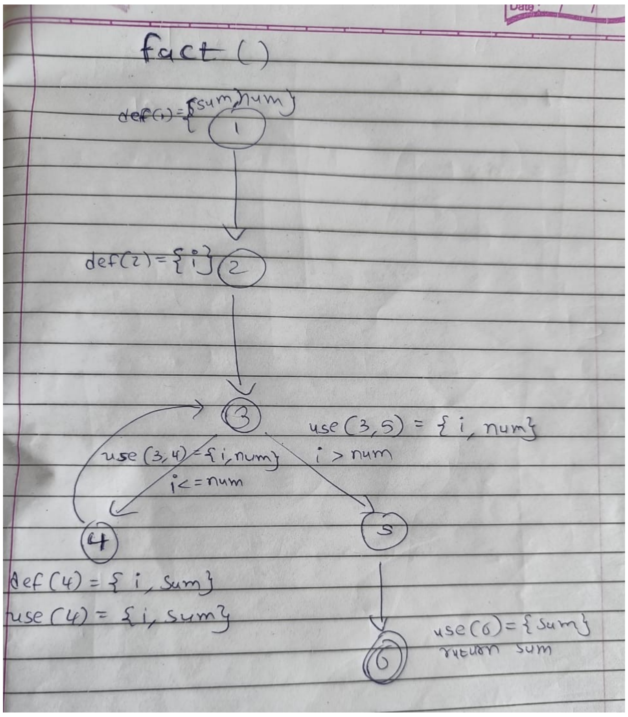
Fact(num)

- This function is used to find factorial of a given number.

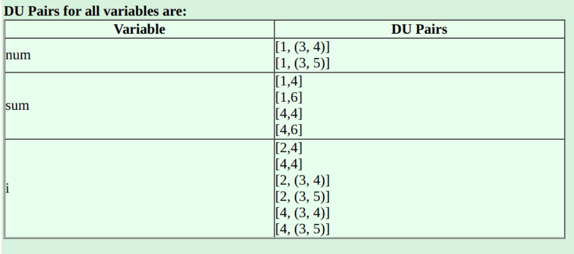
code

```
public long fact(int num) {
        long sum = 1;
        for(int i=2; i<=num; i++) {
            sum = sum * i;
        }
        return sum;
    }
```

DFG

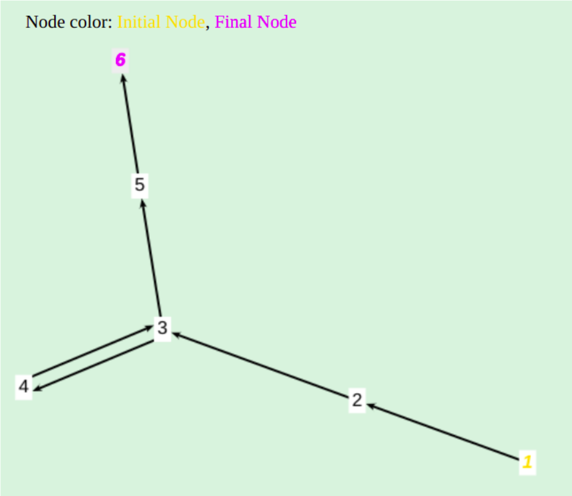


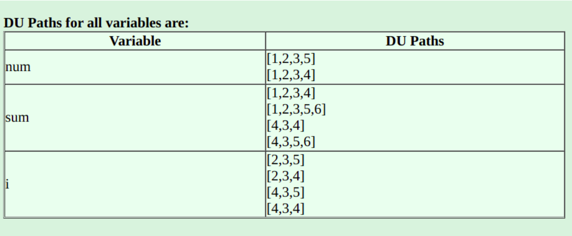
Graph, DU Pairs, Du Paths and All du path coverage



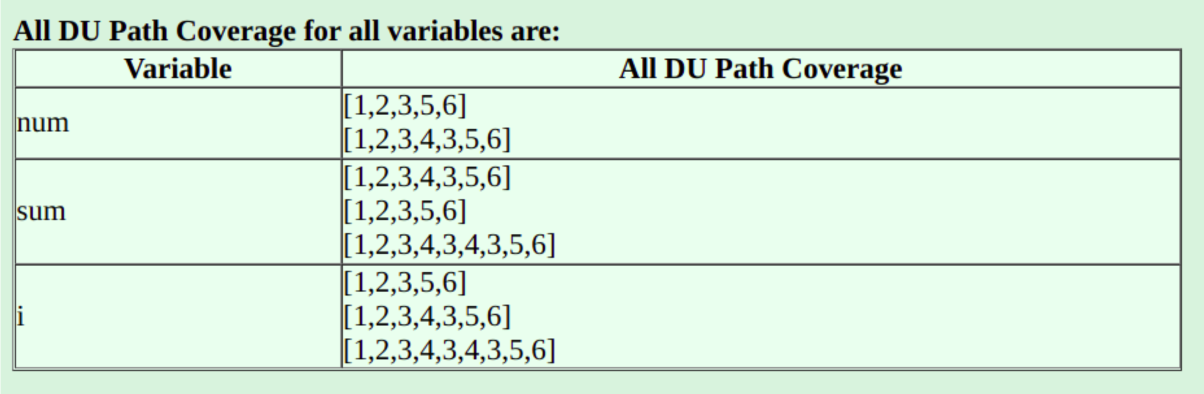
DU Pairs for all variables are:

Variable	DU Pairs
num	[1, (3, 4)] [1, (3, 5)]
sum	[1,4] [1,6] [4,4] [4,6]
i	[2,4] [4,4] [2, (3, 4)] [2, (3, 5)] [4, (3, 4)] [4, (3, 5)]




DU Paths for all variables are:

Variable	DU Paths
num	[1,2,3,5] [1,2,3,4]
sum	[1,2,3,4] [1,2,3,5,6] [4,3,4] [4,3,5,6]
i	[2,3,5] [2,3,4] [4,3,5] [4,3,4]

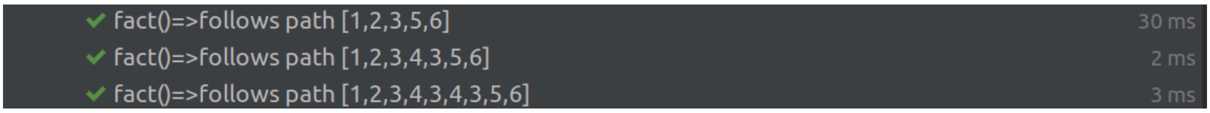

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
num	[1,2,3,5,6] [1,2,3,4,3,5,6]
sum	[1,2,3,4,3,5,6] [1,2,3,5,6] [1,2,3,4,3,4,3,5,6]
i	[1,2,3,5,6] [1,2,3,4,3,5,6] [1,2,3,4,3,4,3,5,6]

Out of these there are 3 unique paths

1. That skips the loop
   a. [1,2,3,5,6]
      i. Test case for this is ⇒ fact(1)
      - Expected output = 1
2. That iterate loop 1 time
   a. [1,2,3,4,3,5,6]
      i. Test case for this is ⇒ fact(2)
      - Expected output = 2
3. That iterate loop 2 or more times
   a. [1,2,3,4,3,4,3,5,6]
      i. Test case for this is ⇒ fact(5)
      - Expected output = 120

Result



```
✓ fact()=>follows path [1,2,3,5,6]          30 ms
✓ fact()=>follows path [1,2,3,4,3,5,6]       2 ms
✓ fact()=>follows path [1,2,3,4,3,4,3,5,6]   3 ms
```

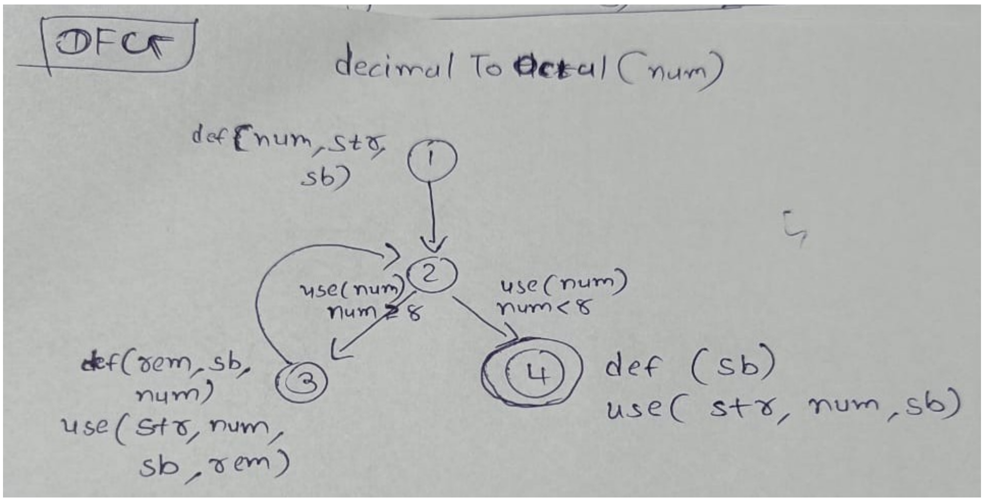
DecimalToOctal(num)

- This function converts decimal number to octal number.

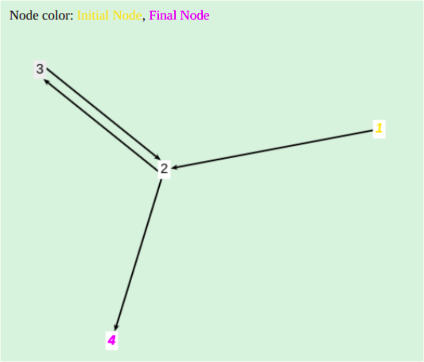
code

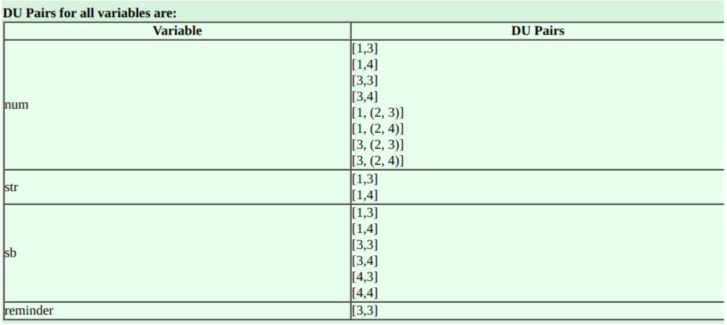
```
public String decimalToOctal(int num) {
        String str = "01234567";
        StringBuilder sb = new StringBuilder();
        while(num>=8)
        {
            int reminder = num%8;
            sb.append(str.charAt(reminder));
            num=num/8;
        }
        sb.append(str.charAt(num));
        sb.reverse();
        return sb.toString();
    }
```

DFG

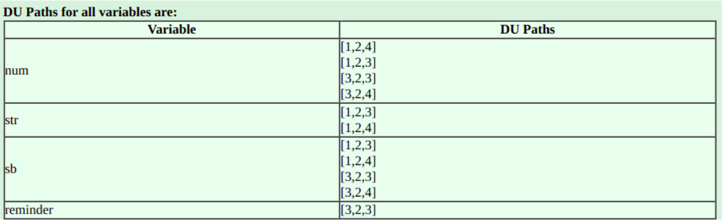


Graph, DU Pairs, Du Paths and All du path coverage

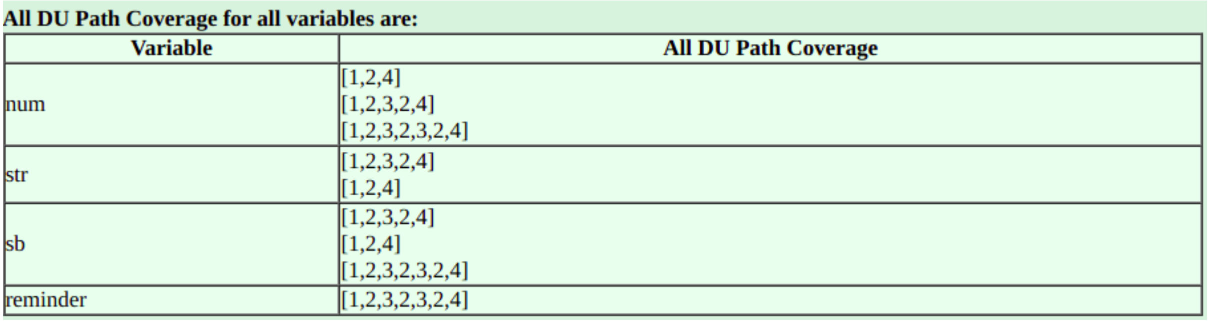



DU Pairs for all variables are:

Variable	DU Pairs
num	[1,3] [1,4] [3,3] [3,4] [1, (2, 3)] [1, (2, 4)] [3, (2, 3)] [3, (2, 4)]
str	[1,3] [1,4]
sb	[1,3] [1,4] [3,3] [3,4] [4,3] [4,4]
reminder	[3,3]


DU Paths for all variables are:

Variable	DU Paths
num	[1,2,4] [1,2,3] [3,2,3] [3,2,4]
str	[1,2,3] [1,2,4]
sb	[1,2,3] [1,2,4] [3,2,3] [3,2,4]
reminder	[3,2,3]


All DU Path Coverage for all variables are:

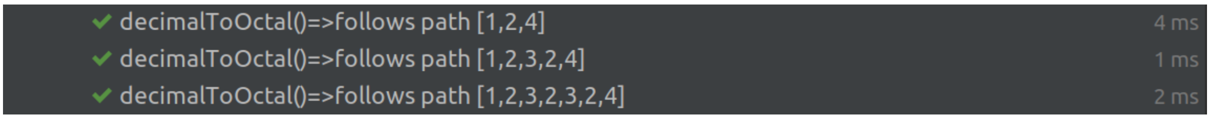
Variable	All DU Path Coverage
num	[1,2,4] [1,2,3,2,4] [1,2,3,2,3,2,4]
str	[1,2,3,2,4] [1,2,4]
sb	[1,2,3,2,4] [1,2,4] [1,2,3,2,3,2,4]
reminder	[1,2,3,2,3,2,4]

There are total 3 unique paths:

1. That skips the loop: [1,2,4]
   - Test case for [1,2,4]:
     - decimalToOctal(num) ⇒ decimalToOctal(7)
     - Expected output = 7
2. That requires at least one iteration of loop: [1,2,3,2,4]
   - Test case for [1,2,3,2,4]:
     - decimalToOctal(num) ⇒ decimalToOctal(10)
     - Expected output = 12
3. That requires at least one iteration of loop: [1,2,3,2,3,2,4]
   - Test case for [1,2,3,2,3,2,4]:
     - decimalToOctal(num) ⇒ decimalToOctal(164)

    - Expected output = 244

Result



```
✓ decimalToOctal()=>follows path [1,2,4]              4 ms
✓ decimalToOctal()=>follows path [1,2,3,2,4]          1 ms
✓ decimalToOctal()=>follows path [1,2,3,2,3,2,4]      2 ms
```

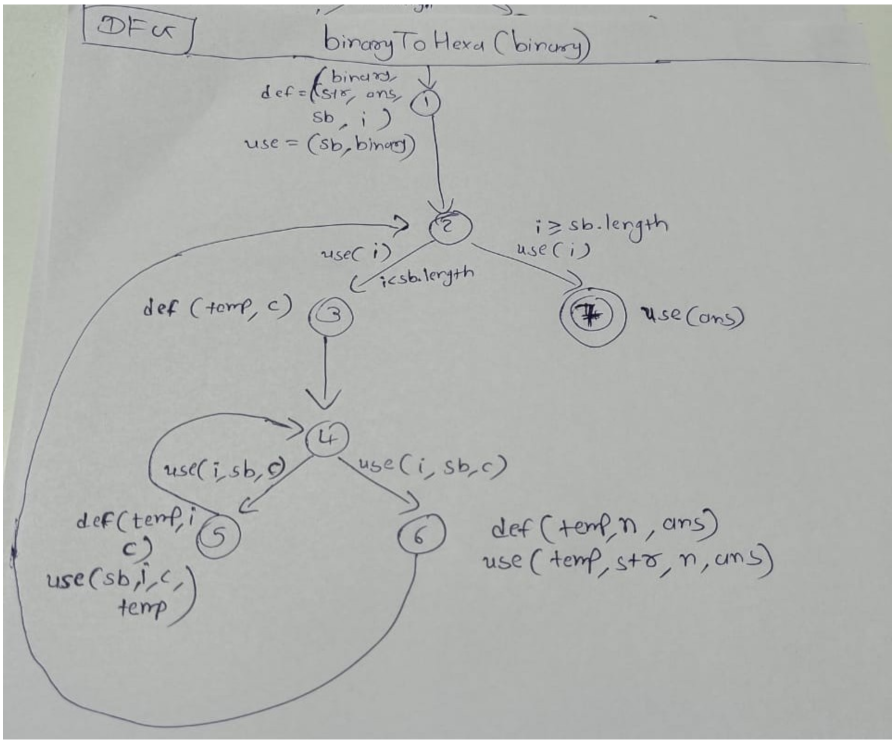
binaryToHexa()

- This function converts number from binary number system to hexadecimal number system.

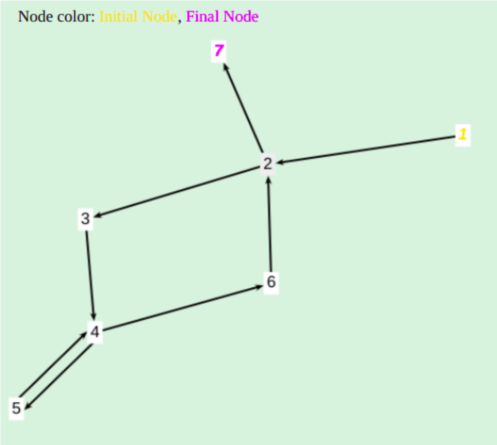
Code

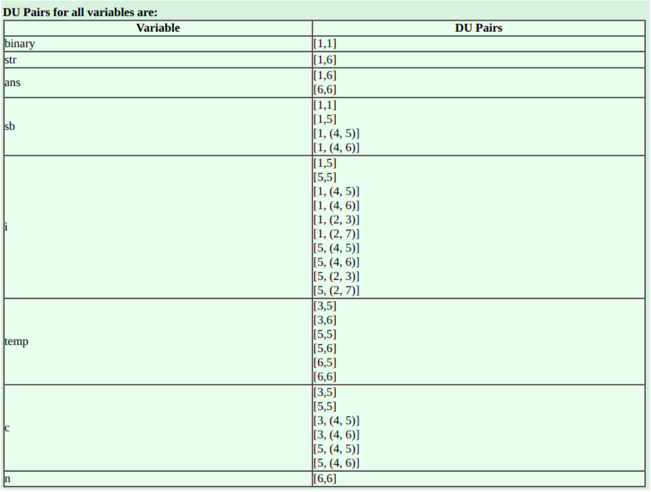
```
public String binaryToHexa(String binary) {
        String str = "0123456789ABCDEF";
        StringBuilder ans = new StringBuilder();
        StringBuilder sb = new StringBuilder(binary);
        sb.reverse();
        int i=0;
        while(i<sb.length())
        {
            int c=0;
            StringBuilder temp = new StringBuilder();
            while(i<sb.length() && c<4)
            {
                temp.append(sb.charAt(i));
                i++;
                c++;
            }
            temp.reverse();
            int n = binaryToDecimal(temp.toString());
            ans.append(str.charAt(n));
        }
        ans.reverse();
        return ans.toString();
    }
```

DFG



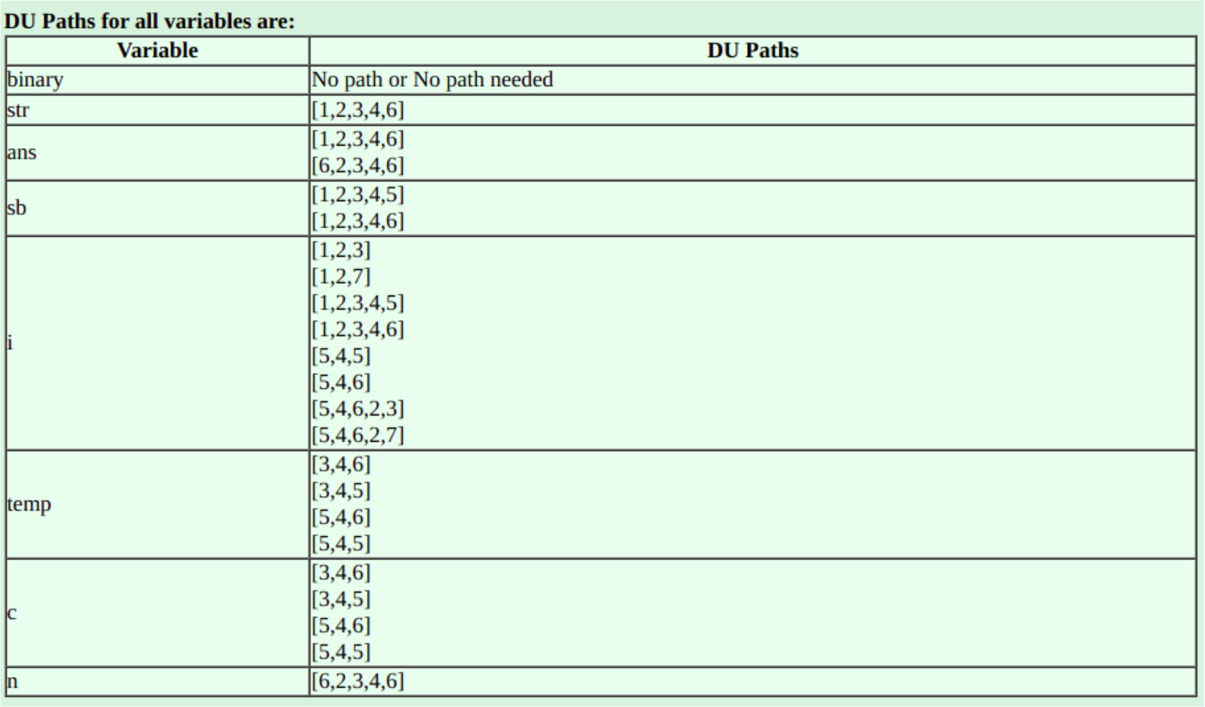
Graph, DU Pairs, Du Paths and All du path coverage





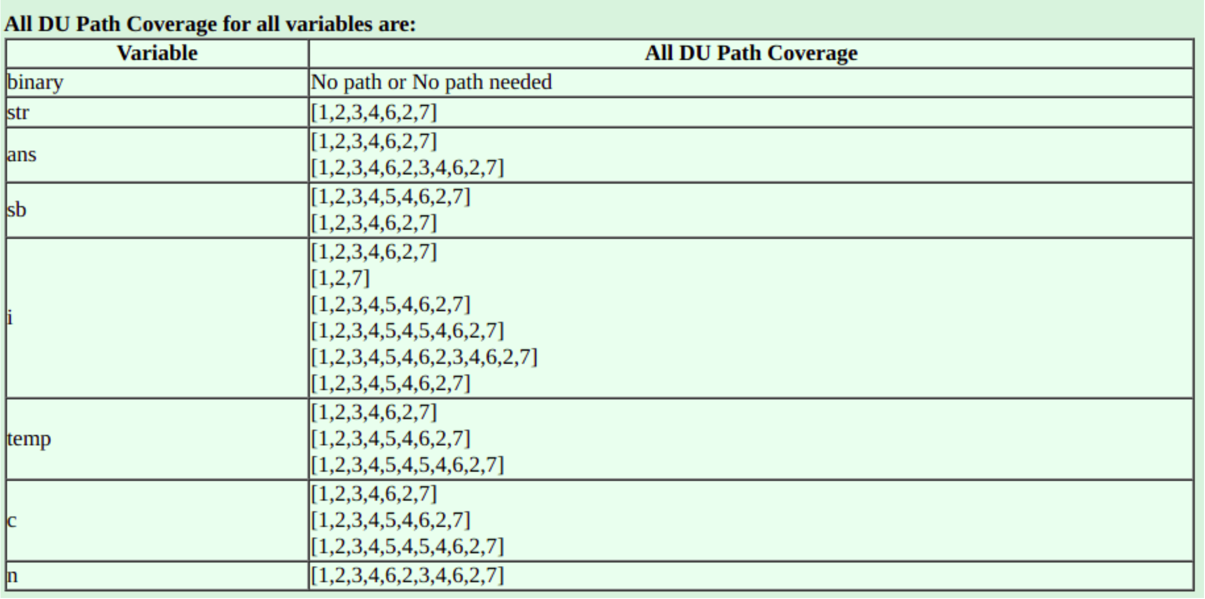
DU Pairs for all variables are:

Variable	DU Pairs
binary	[1,1]
str	[1,6]
ans	[1,6] [6,6]
sb	[1,1] [1,5] [1, (4, 5)] [1, (4, 6)]
i	[1,5] [5,5] [1, (4, 5)] [1, (4, 6)] [1, (2, 3)] [1, (2, 7)] [5, (4, 5)] [5, (4, 6)] [5, (2, 3)] [5, (2, 7)]
temp	[3,5] [3,6] [5,5] [5,6] [6,5] [6,6]
c	[3,5] [5,5] [3, (4, 5)] [3, (4, 6)] [5, (4, 5)] [5, (4, 6)]
n	[6,6]



DU Paths for all variables are:

Variable	DU Paths
binary	No path or No path needed
str	[1,2,3,4,6]
ans	[1,2,3,4,6] [6,2,3,4,6]
sb	[1,2,3,4,5] [1,2,3,4,6]
i	[1,2,3] [1,2,7] [1,2,3,4,5] [1,2,3,4,6] [5,4,5] [5,4,6] [5,4,6,2,3] [5,4,6,2,7]
temp	[3,4,6] [3,4,5] [5,4,6] [5,4,5]
c	[3,4,6] [3,4,5] [5,4,6] [5,4,5]
n	[6,2,3,4,6]



All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
binary	No path or No path needed
str	[1,2,3,4,6,2,7]
ans	[1,2,3,4,6,2,7] [1,2,3,4,6,2,3,4,6,2,7]
sb	[1,2,3,4,5,4,6,2,7] [1,2,3,4,6,2,7]
i	[1,2,3,4,6,2,7] [1,2,7] [1,2,3,4,5,4,6,2,7] [1,2,3,4,5,4,5,4,6,2,7] [1,2,3,4,5,4,6,2,3,4,6,2,7] [1,2,3,4,5,4,6,2,7]
temp	[1,2,3,4,6,2,7] [1,2,3,4,5,4,6,2,7] [1,2,3,4,5,4,5,4,6,2,7]
c	[1,2,3,4,6,2,7] [1,2,3,4,5,4,6,2,7] [1,2,3,4,5,4,5,4,6,2,7]
n	[1,2,3,4,6,2,3,4,6,2,7]

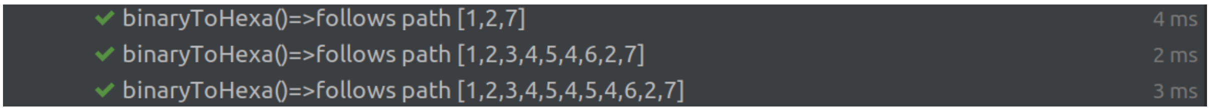
There are total 6 unique paths:

1. That skips all the loop: [1,2,7]
   - Test case for [1,2,7]:
     - binaryToHexa(binaryString) ⇒ binaryToHexa("") (An empty String)
     - Expected output = ""
2. That skip the second loop: [1,2,3,4,6,2,7], [1,2,3,4,6,2,3,4,6,2,7]
   - Test case for [1,2,3,4,6,2,7]:

    - This case is infeasible to cover because if first while condition is true (i<sb.length()) then the second while condition (i<sb.length() & c<4) will always be true for first time.
  - Test case for [1,2,3,4,6,2,3,4,6,2,7]:
    - Again for the same reason this case is also infeasible.

3. That considers both the loop: [1,2,3,4,5,4,6,2,7], [1,2,3,4,5,4,5,4,6,2,7], [1,2,3,4,5,4,6,2,3,4,6,2,7]
   - Test case for [1,2,3,4,5,4,6,2,7]:
     - binaryToHexa(binaryString) ⇒ binaryToHexa("1")
     - Expected output = "1"
   - Test case for [1,2,3,4,5,4,5,4,6,2,7]:
     - binaryToHexa(binaryString) ⇒ binaryToHexa("10")
     - Expected output = "2"
   - Test case for [1,2,3,4,5,4,6,2,3,4,6,2,7]:
     - This path is infeasible to cover because of the same reason again if the second loop condition becomes false after first iteration then first loop condition will always become false. so 4,5,5,6,2,**3,2,4** this path will never be possible.

Result



```
✓ binaryToHexa()=>follows path [1,2,7]                      4 ms
✓ binaryToHexa()=>follows path [1,2,3,4,5,4,6,2,7]          2 ms
✓ binaryToHexa()=>follows path [1,2,3,4,5,4,5,4,6,2,7]      3 ms
```

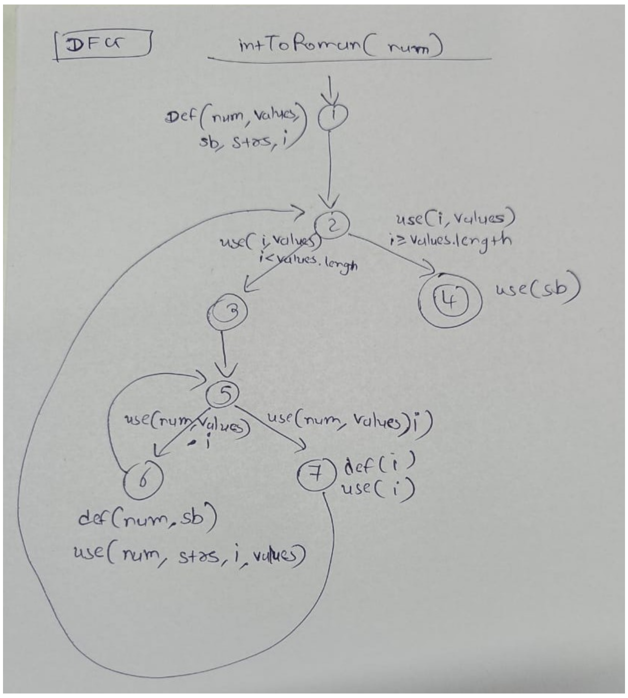
intToRoman(num)

- This function converts number from integer to roman number system.

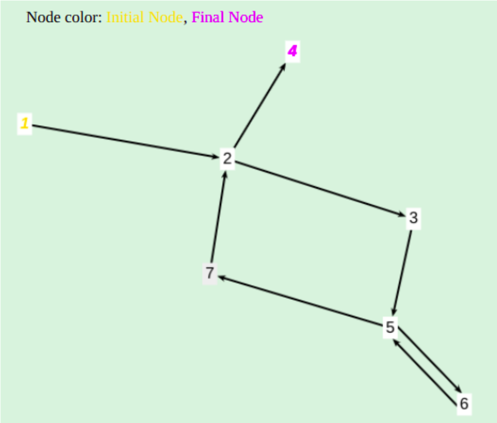
Code

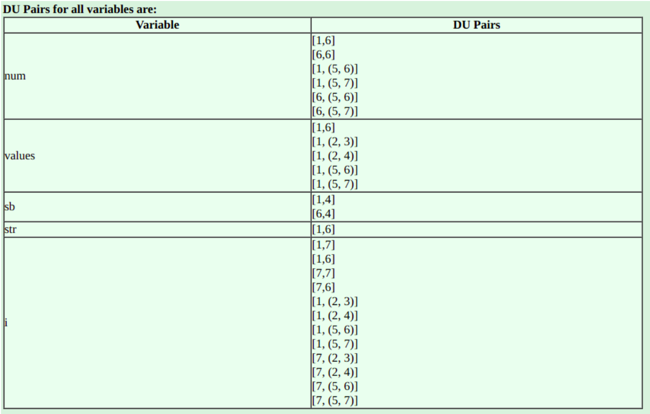
```
public String intToRoman(int num) {
        int[] values = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
        String[] strs = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};

        StringBuilder sb = new StringBuilder();

        for(int i=0;i<values.length;i++) {
            while(num >= values[i]) {
                num -= values[i];
                sb.append(strs[i]);
            }
        }
        return sb.toString();
    }
```

DFG


DFG
intToRoman(num)
Def(num, values, sb, stas, i)
1
2
use(i, values) i≥values.length
use(i, values) i<values. length
3
4
use(sb)
5
use(num, values) i
use(num, values) i)
6
7
def(i)
use(i)
def(num, sb)
use(num, stas, i, values)

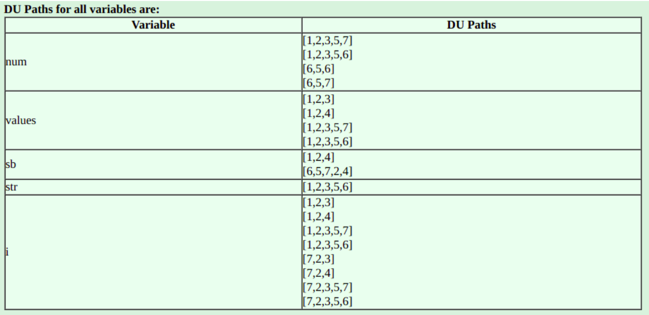
Graph, DU Pairs, Du Paths and All du path coverage





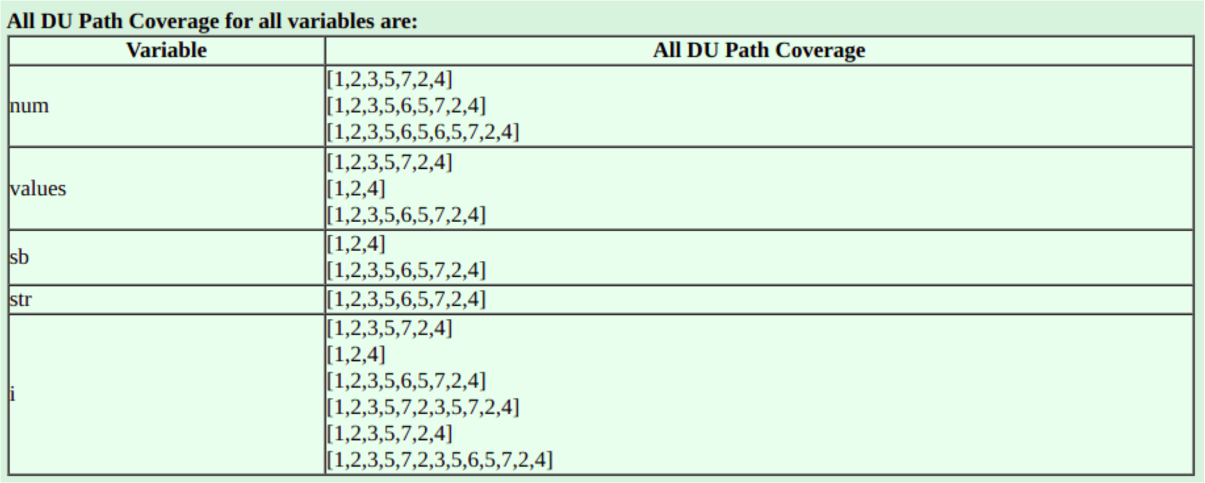
DU Pairs for all variables are:

Variable	DU Pairs
num	[1,6] [6,6] [1, (5, 6)] [1, (5, 7)] [6, (5, 6)] [6, (5, 7)]
values	[1,6] [1, (2, 3)] [1, (2, 4)] [1, (5, 6)] [1, (5, 7)]
sb	[1,4] [6,4]
str	[1,6]
i	[1,7] [1,6] [7,7] [7,6] [1, (2, 3)] [1, (2, 4)] [1, (5, 6)] [1, (5, 7)] [7, (2, 3)] [7, (2, 4)] [7, (5, 6)] [7, (5, 7)]



DU Paths for all variables are:

Variable	DU Paths
num	[1,2,3,5,7] [1,2,3,5,6] [6,5,6] [6,5,7]
values	[1,2,3] [1,2,4] [1,2,3,5,7] [1,2,3,5,6]
sb	[1,2,4] [6,5,7,2,4]
str	[1,2,3,5,6]
i	[1,2,3] [1,2,4] [1,2,3,5,7] [1,2,3,5,6] [7,2,3] [7,2,4] [7,2,3,5,7] [7,2,3,5,6]



All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
num	[1,2,3,5,7,2,4] [1,2,3,5,6,5,7,2,4] [1,2,3,5,6,5,6,5,7,2,4]
values	[1,2,3,5,7,2,4] [1,2,4] [1,2,3,5,6,5,7,2,4]
sb	[1,2,4] [1,2,3,5,6,5,7,2,4]
str	[1,2,3,5,6,5,7,2,4]
i	[1,2,3,5,7,2,4] [1,2,4] [1,2,3,5,6,5,7,2,4] [1,2,3,5,7,2,3,5,7,2,4] [1,2,3,5,7,2,4] [1,2,3,5,7,2,3,5,6,5,7,2,4]

- There are total 16 unique paths: [1,2,4], [1,2,3,5,7,2,4], [1,2,3,5,6,5,7,2,4], [1,2,3,5,6,5,6,5,7,2,4], [1,2,3,5,7,2,3,5,6,5,7,2,4]
- These all paths are infeasible to cover because the outer for loop always requires 13 iterations because of length of the values array is 13.

Contributions

- Code, DFGs & CFGs, Du pairs, Du paths and All Du path coverage TR and Test case design of power(), binaryToHexa() and decimalToOctal() done by **Akshil Gadhiya(MT2020007).**
- Code, DFGs & CFGs, Du pairs, Du paths and All Du path coverage TR and Test case design of fact() and intToRoman() done by **Aaly Dhagia(MT2020102).**