Software Testing Project

https://github.com/jainpranay20/AptitudeSolver.git

Gopal Goyal MT2021048

Pranay Jain MT2021098

Source Code Description

- We have developed a project-specific aptitude solver that combines all major aptitude features.

- It is a console-based java application.

  The code provides the following functionalities.

  1. It calculates the surface areas and volumes of various 2D and 3D figures like squares, rectangles, circles, cubes, spheres, etc. Therefore, it covers the measurement topic of quantitative aptitude.

  2. It calculates and returns the status of profit and loss based on the cost price and selling price provided as inputs.

  3. Compound interest and simple interest can also be calculated using it.

  4. The program provides a number of core features such as finding the nth term, printing the first n terms, computing the sum of the first n terms, etc, for progressions such as Arithmetic Progressions (AP) and Geometric Progressions (GP).

  5. Additionally, we can compute percentage changes, percentage values for fractions, and the number of days it will take if people with given efficiencies cooperate.

Testing strategy

Data flow graph:-

- For each function, we are first creating a 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.

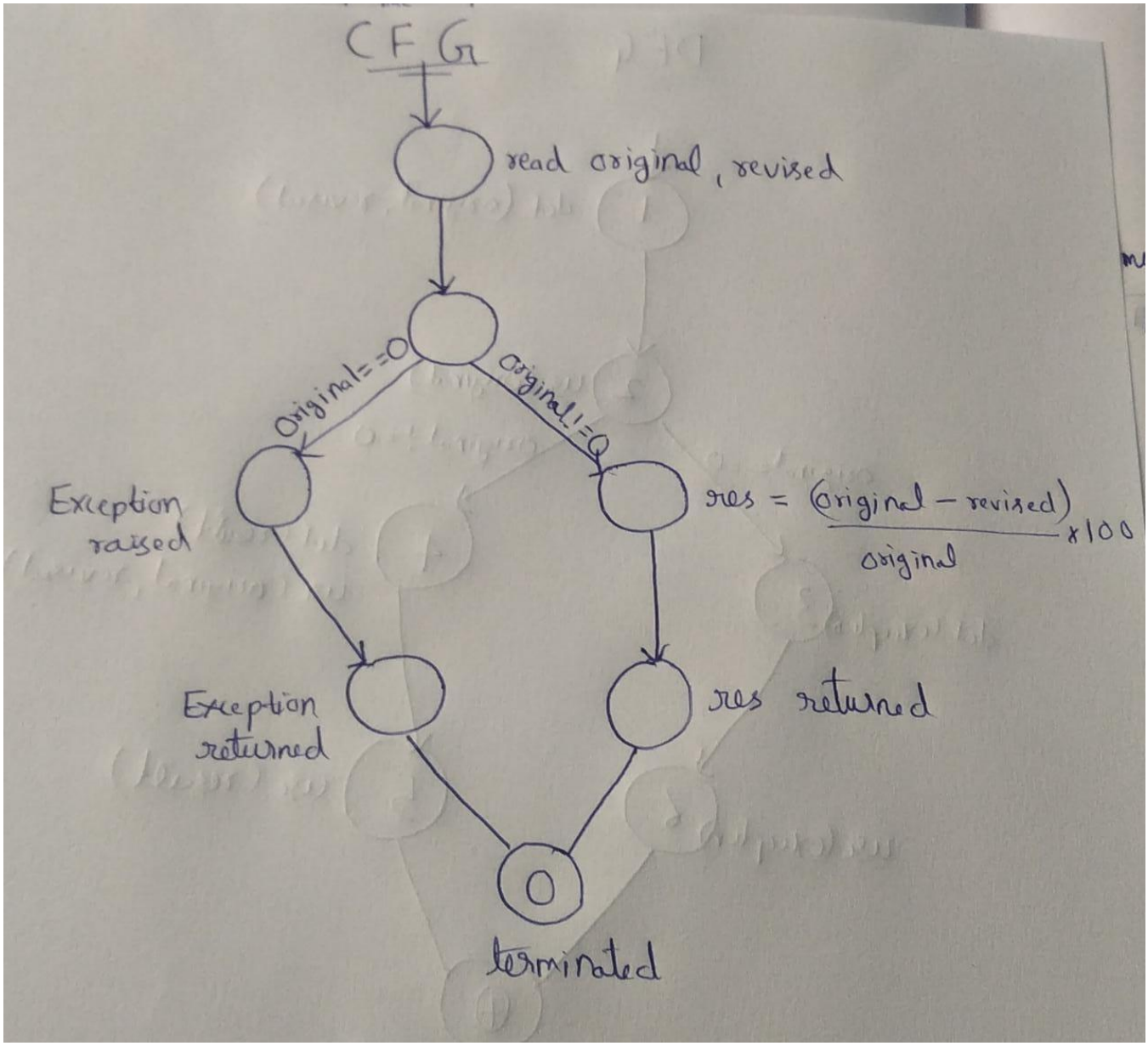
Tools used:-

- **Junit for test cases**
- **https://cs.gmu.edu:8443/offutt/coverage/DFGraphCoverage for TR generation.**
- **Below are some of the functions among the ones we have tested.**

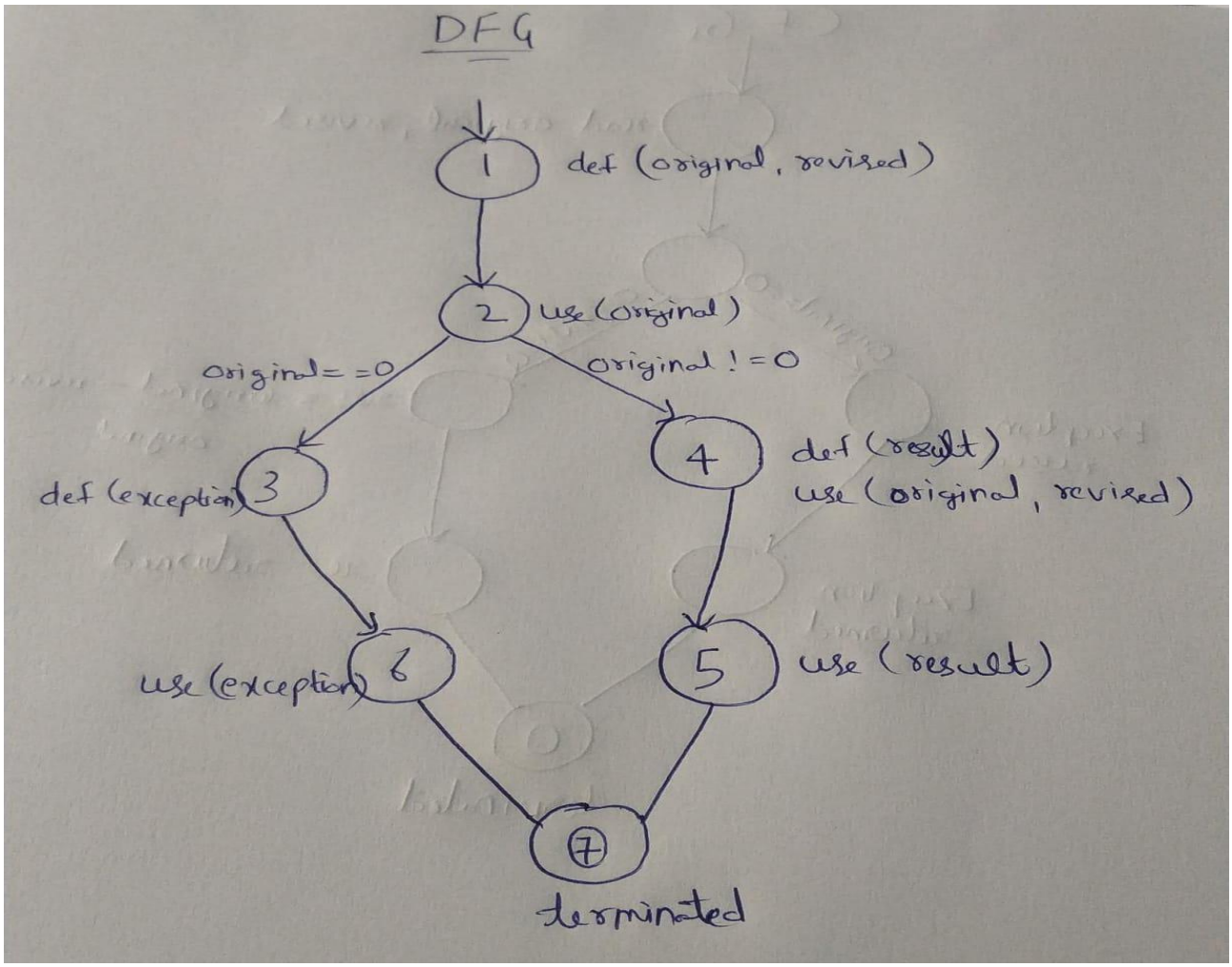
1. Percentage Change:-

```java
public String percentageChange(double original, double revised)
{
    try {
        if (original == 0) {
            throw new Exception();
        }
        double res = ((revised - original) / original) * 100;
        return Double.toString(res);
    }catch (Exception e)
    {
        return "Invalid original value";
    }
}
```

CFG


CFG
read original, revised
original==0
original!=0
Exception raised
res = (original − revised) / original * 100
Exception returned
res returned
0
terminated

DFG

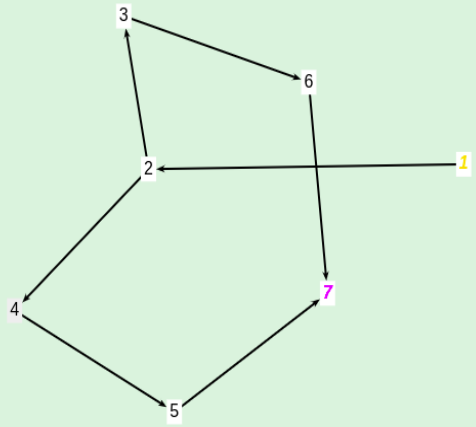

DFG
1
def (original, revised)
2
use (original)
original == 0
original != 0
3
def (exception)
4
def (result)
use (original, revised)
6
use (exception)
5
use (result)
7
terminated

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

DU Pairs for all variables are:

Variable	DU Pairs
original	[1,2] [1,4]
revised	[1,4]
exception	[3,6]
result	[4,5]

Node color: Initial Node, Final Node



DU Paths for all variables are:

Variable	DU Paths
original	[1,2] [1,2,4]
revised	[1,2,4]
exception	[3,6]
result	[4,5]

All DU Path Coverage for all variables are:

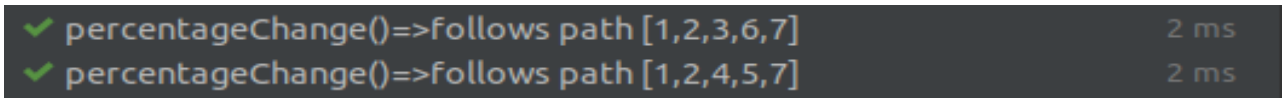
Variable	All DU Path Coverage
original	[1,2,3,6,7] [1,2,4,5,7]
revised	[1,2,4,5,7]
exception	[1,2,3,6,7]
result	[1,2,4,5,7]

We have total 2 unique paths here:-

1. [1,2,3,6,7]
   Input:- original = 0, revised = 10
   Output:- Invalid original value.

2. [[1,2,4,5,7]
   Input:- original = 10, revised = 15
   Output:- 50.0

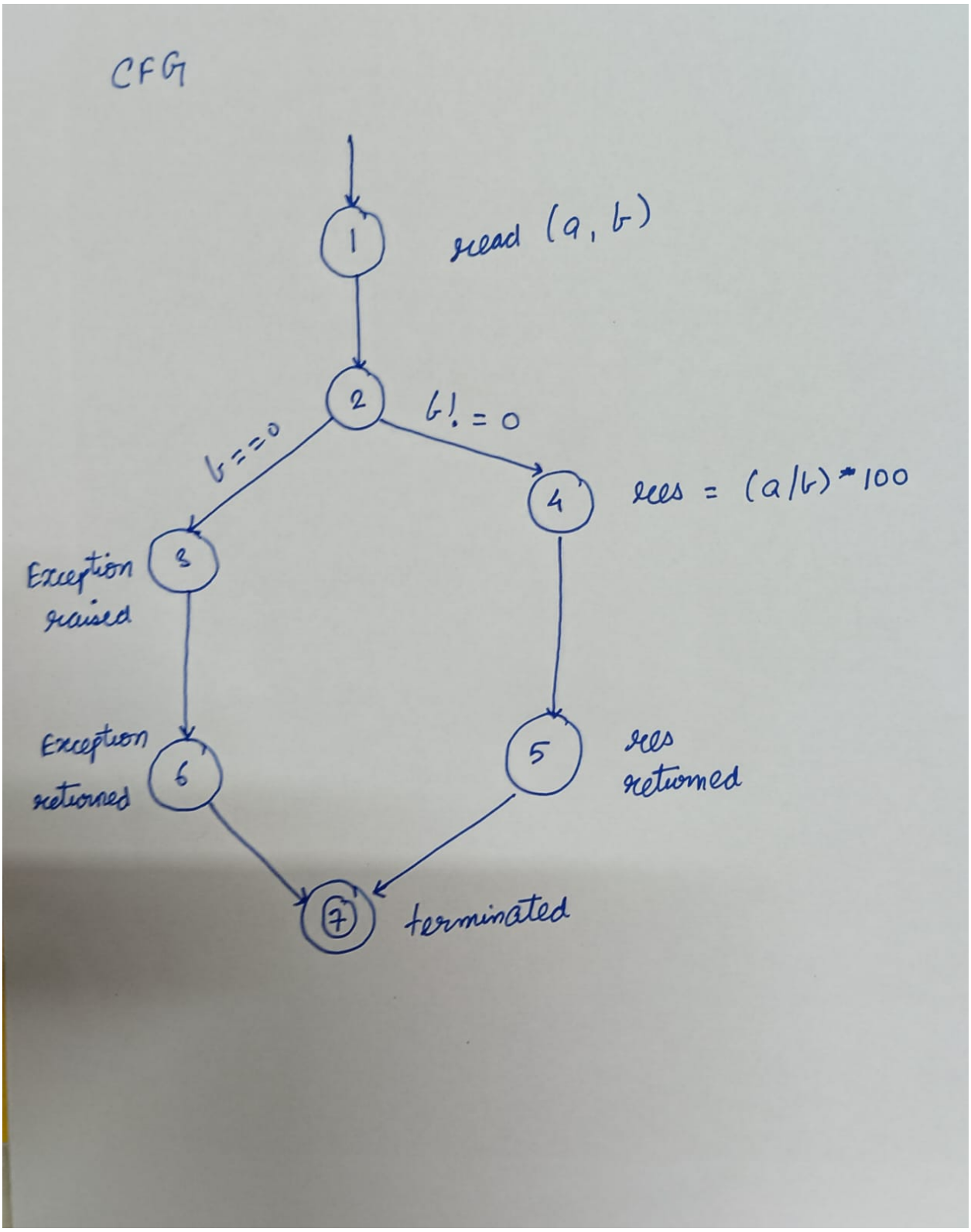
Result



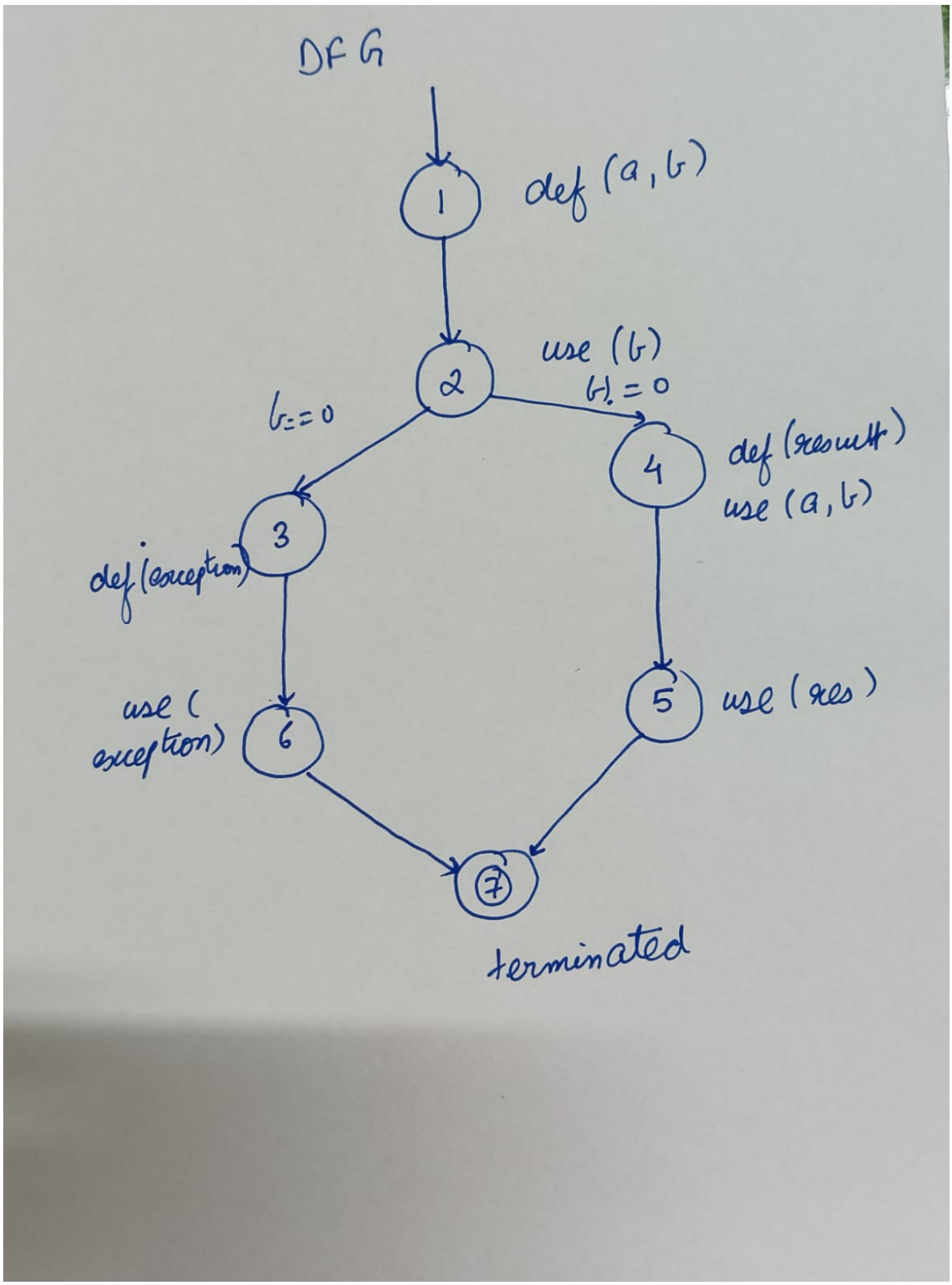
2. Fraction to percentage converter:-

```java
public String fractionToPercentageConverter(double a, double b)
{
    try
    {
        if(b==0)
        {
            throw new Exception();
        }
        else
        {
            double res = (a/b)*100;
            return Double.toString(res);
        }
    }catch(Exception e)
    {
        return "Value of b cannot be zero";
    }
}
```

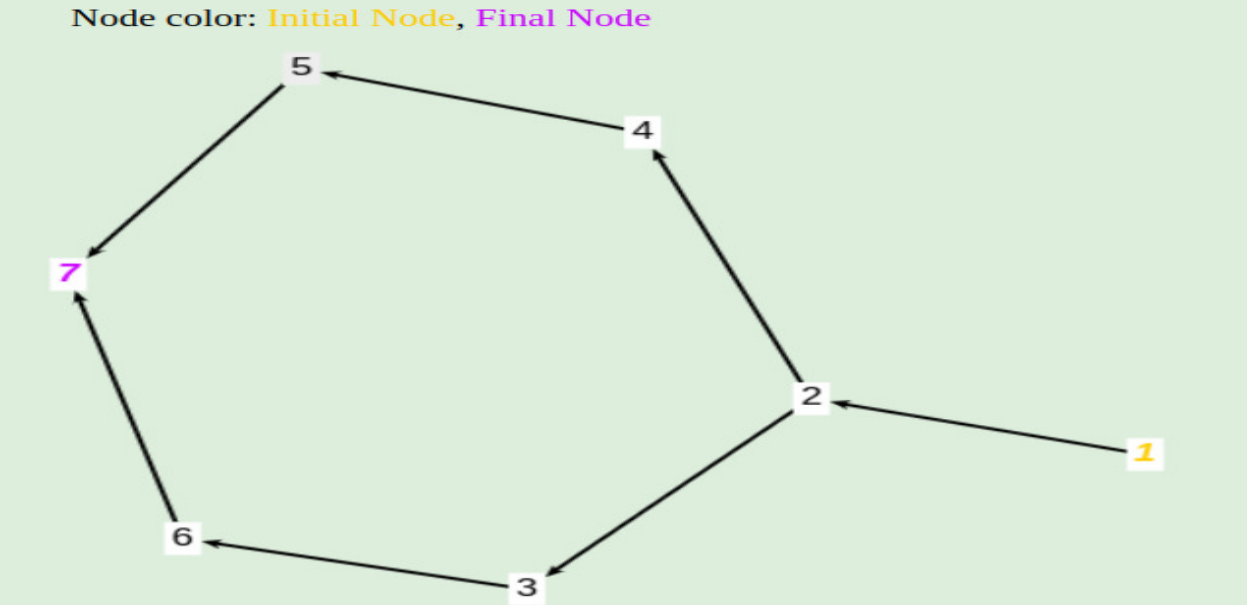
CFG


CFG
1
read (a, b)
2
b != 0
b == 0
4
res = (a/b) * 100
3
Exception raised
5
res returned
Exception returned
6
7
terminated

DFG


DFG
1
def (a, b)
2
use (b)
b != 0
b == 0
4
def (result)
use (a, b)
3
def (exception)
5
use (res)
use (exception)
6
7
terminated

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



DU Paths for all variables are:

Variable	DU Paths
a	[1,2,4]
b	[1,2] [1,2,4]
exception	[3,6]
res	[4,5]

DU Pairs for all variables are:

Variable	DU Pairs
a	[1,4]
b	[1,2] [1,4]
exception	[3,6]
res	[4,5]

All DU Path Coverage for all variables are:

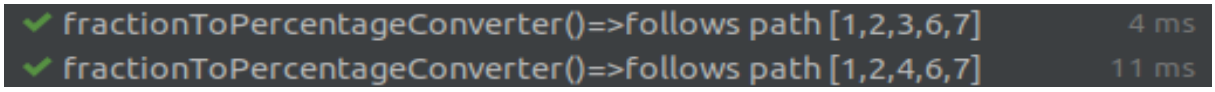
Variable	All DU Path Coverage
a	[1,2,4,5,7]
b	[1,2,4,5,7]
exception	[1,2,3,6,7]
res	[1,2,4,5,7]

We have total 2 unique paths here:-

1. [1,2,3,6,7]
   Input:- a = 10, b =0
   Output:- value of b cannot be zero

2. [[1,2,4,5,7]
   Input:- original = 20, revised = 40
   Output:- 50.0

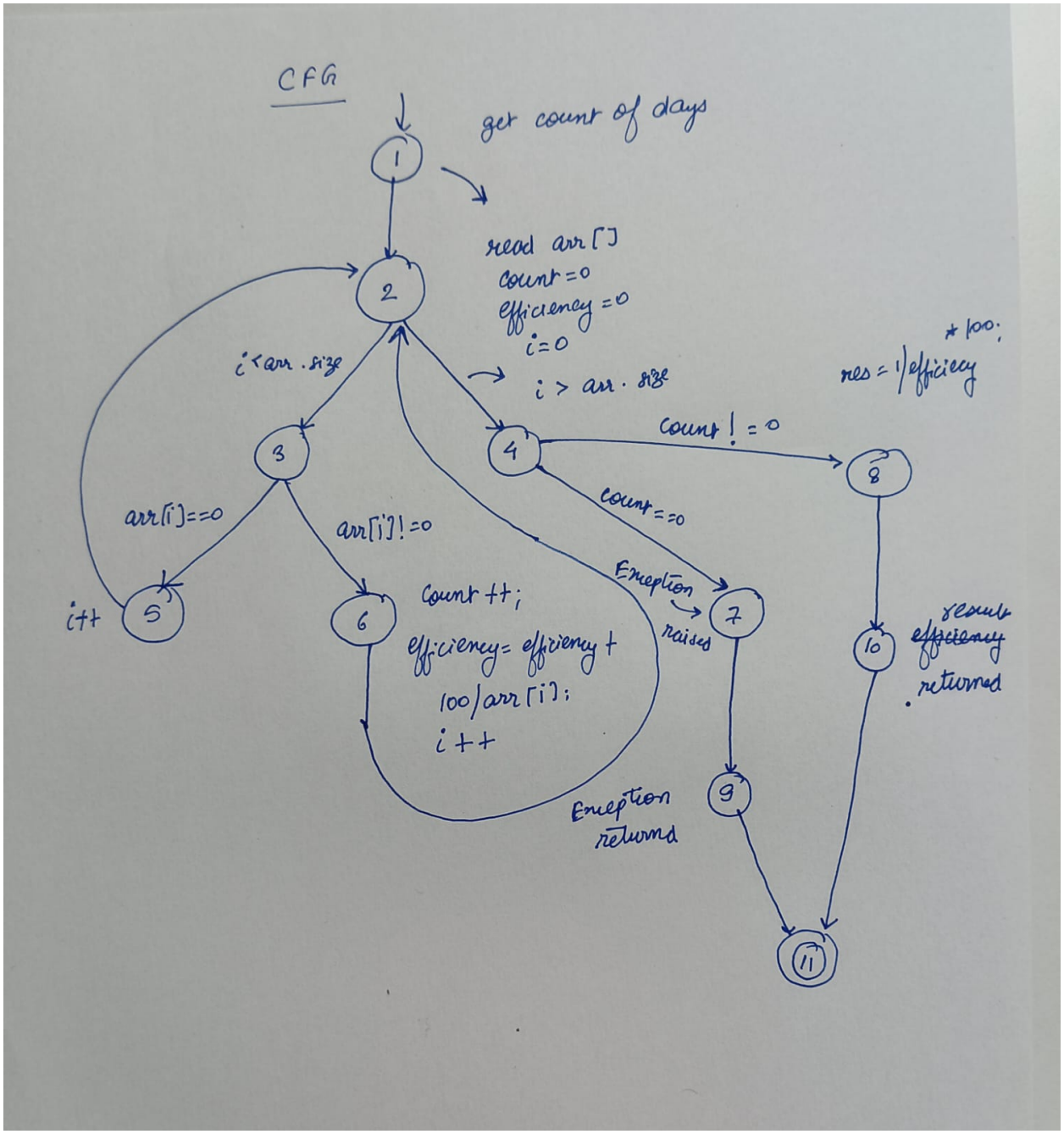
Result



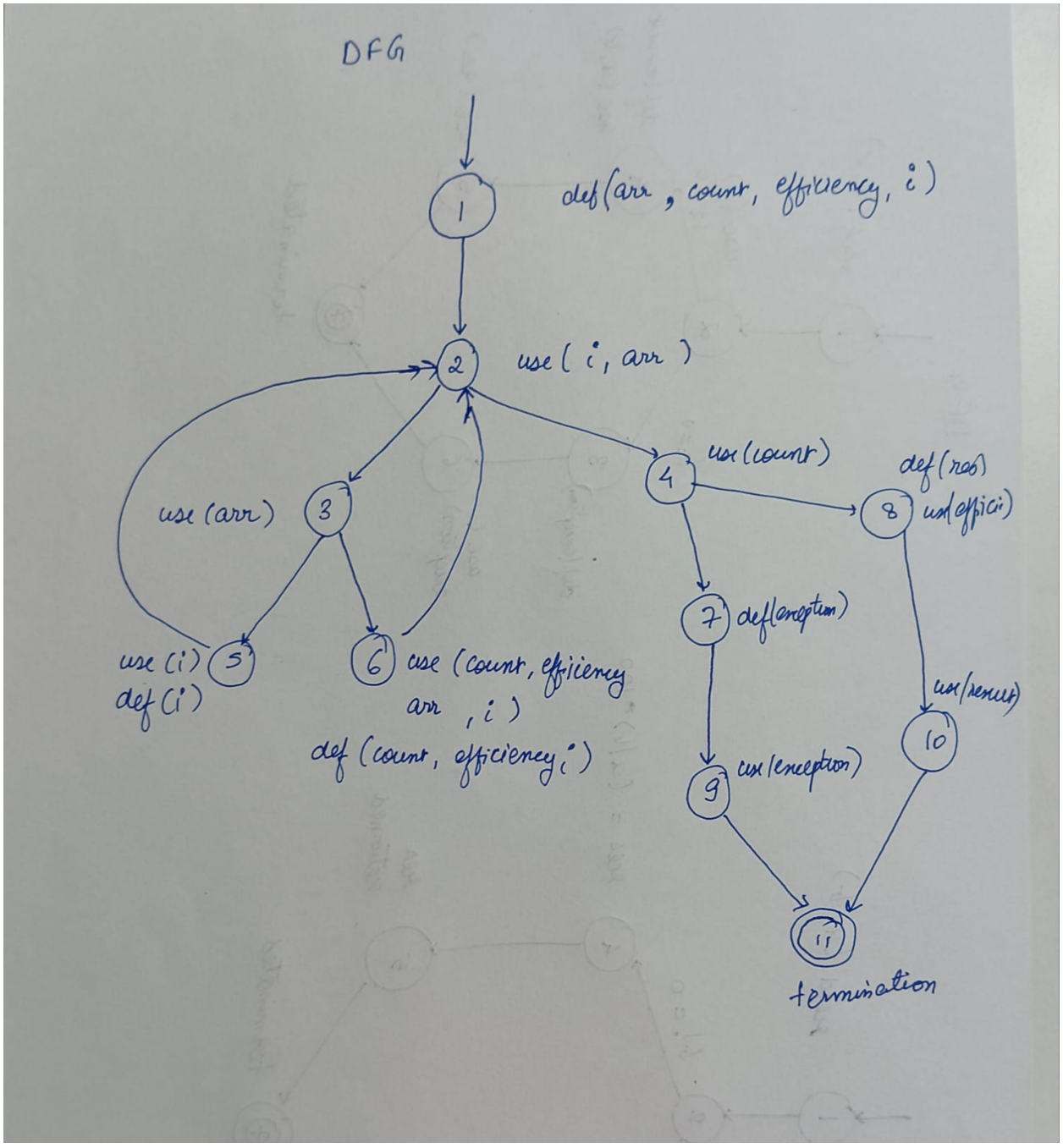
3. Count of Days:-

```java
public String getCountOfDays(int[] arr)
{
    int count = 0;
    double efficiency=0;
    for(int i=0;i<arr.length;i++)
    {
        if(arr[i]!=0)
        {
            count++;
            efficiency = efficiency + 100/arr[i];
        }
    }
    try
    {
        if(count==0)
        {
            throw new Exception();
        }
        double res = (1/efficiency)*100;
        return Double.toString(res);
    }catch (Exception e)
    {
        return "Invalid input";
    }
}
```

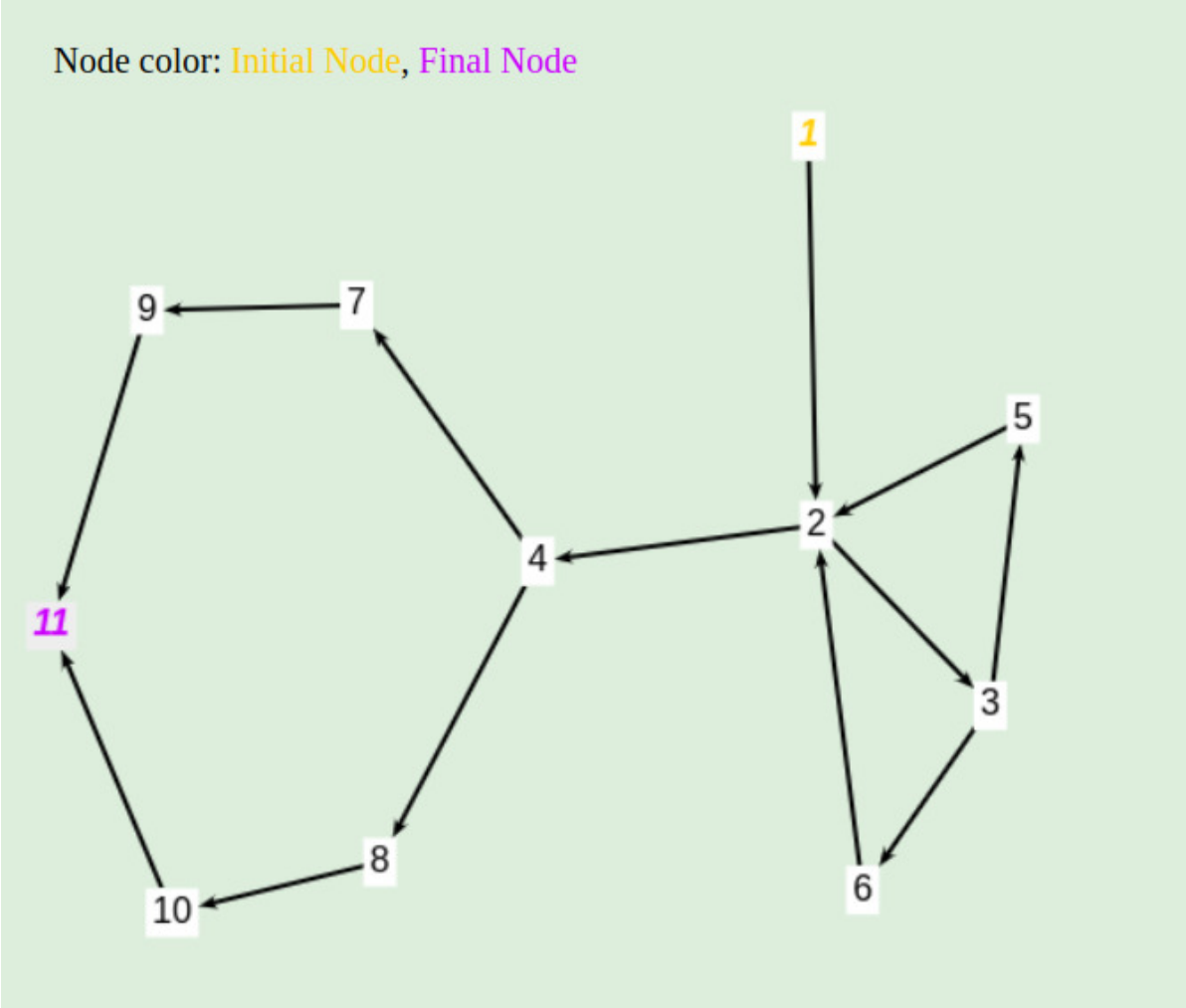
CFG


CFG
get count of days
1
read arr[]
count = 0
efficiency = 0
i = 0
2
i < arr.size
i > arr.size
3
4
count != 0
count == 0
8
res = 1/efficiency
* 100;
arr[i] == 0
arr[i] != 0
i++
5
6
count ++;
efficiency = efficiency + 100/arr[i];
i++
Exception raised
7
Exception returned
9
10
result efficiency returned
11

DFG


DFG
1
def(arr , count, efficiency, i)
2
use (i, arr)
3
use (arr)
4
use(count)
8
def(res)
use(effici)
use (i)
def (i)
5
6
use (count, efficiency
arr , i)
def (count, efficiency, i)
7
def(exception)
9
use/exception
10
use/result
11
termination

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



DU Paths for all variables are:

Variable	DU Paths
arr	[1,2] [1,2,3] [1,2,3,6]
count	[1,2,4] [1,2,3,6] [6,2,4] [6,2,3,6]
efficiency	[1,2,4,8] [1,2,3,6] [6,2,4,8] [6,2,3,6]
i	[1,2] [1,2,3,5] [1,2,3,6] [5,2] [5,2,3,6] [5,2,3,5] [6,2] [6,2,3,6] [6,2,3,5]
exception	[7,9]
result	[8,10]

DU Pairs for all variables are:

Variable	DU Pairs
arr	[1,2] [1,3] [1,6]
count	[1,6] [1,4] [6,6] [6,4]
efficiency	[1,6] [1,8] [6,6] [6,8]
i	[1,2] [1,5] [1,6] [5,2] [5,5] [5,6] [6,2] [6,5] [6,6]
exception	[7,9]
result	[8,10]

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
arr	[1,2,4,7,9,11] [1,2,3,5,2,4,7,9,11] [1,2,3,6,2,4,7,9,11]
count	[1,2,4,7,9,11] [1,2,3,6,2,4,7,9,11] [1,2,3,6,2,3,6,2,4,7,9,11]
efficiency	[1,2,4,8,10,11] [1,2,3,6,2,4,7,9,11] [1,2,3,6,2,4,8,10,11] [1,2,3,6,2,3,6,2,4,7,9,11]
i	[1,2,4,7,9,11] [1,2,3,5,2,4,7,9,11] [1,2,3,6,2,4,7,9,11] [1,2,3,5,2,3,6,2,4,7,9,11] [1,2,3,5,2,3,5,2,4,7,9,11] [1,2,3,6,2,3,6,2,4,7,9,11] [1,2,3,6,2,3,5,2,4,7,9,11]
exception	[1,2,4,7,9,11]
result	[1,2,4,8,10,11]

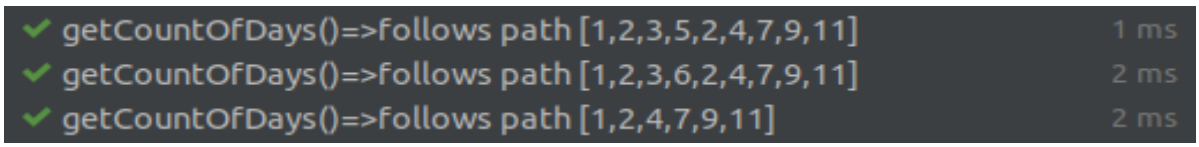
Here we have total 3 unique paths:-

1. [1,2,3,5,2,4,7,9,11]
   Input:- arr[]={}
   Output:-Invalid input

2. [1,2,3,6,2,4,7,9,11]
   Input:- arr[]={1}
   Output:- 1.0

3. [1,2,4,7,9,11]
   Input:- arr[]={}
   Output:- Invalid input

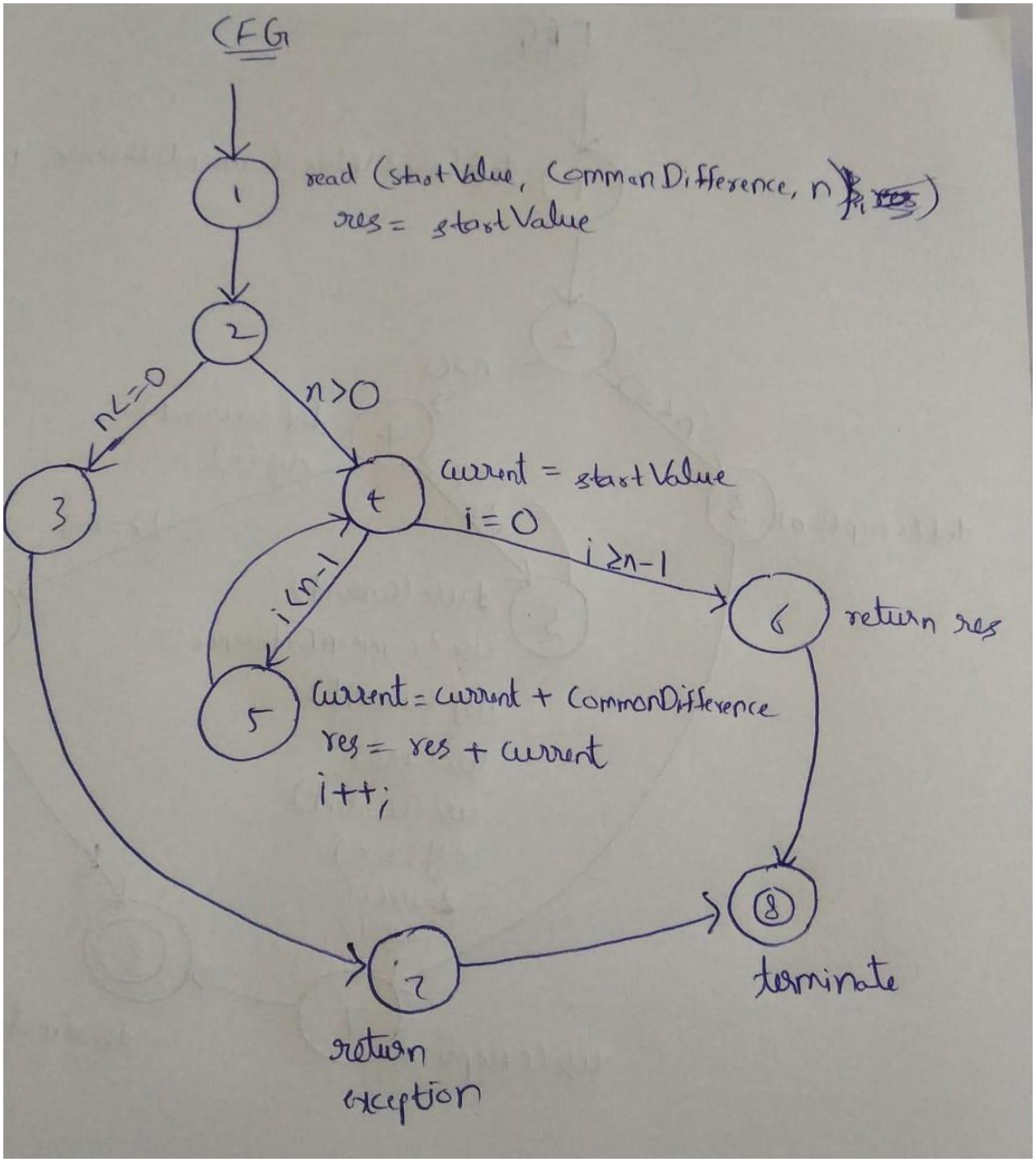
Result



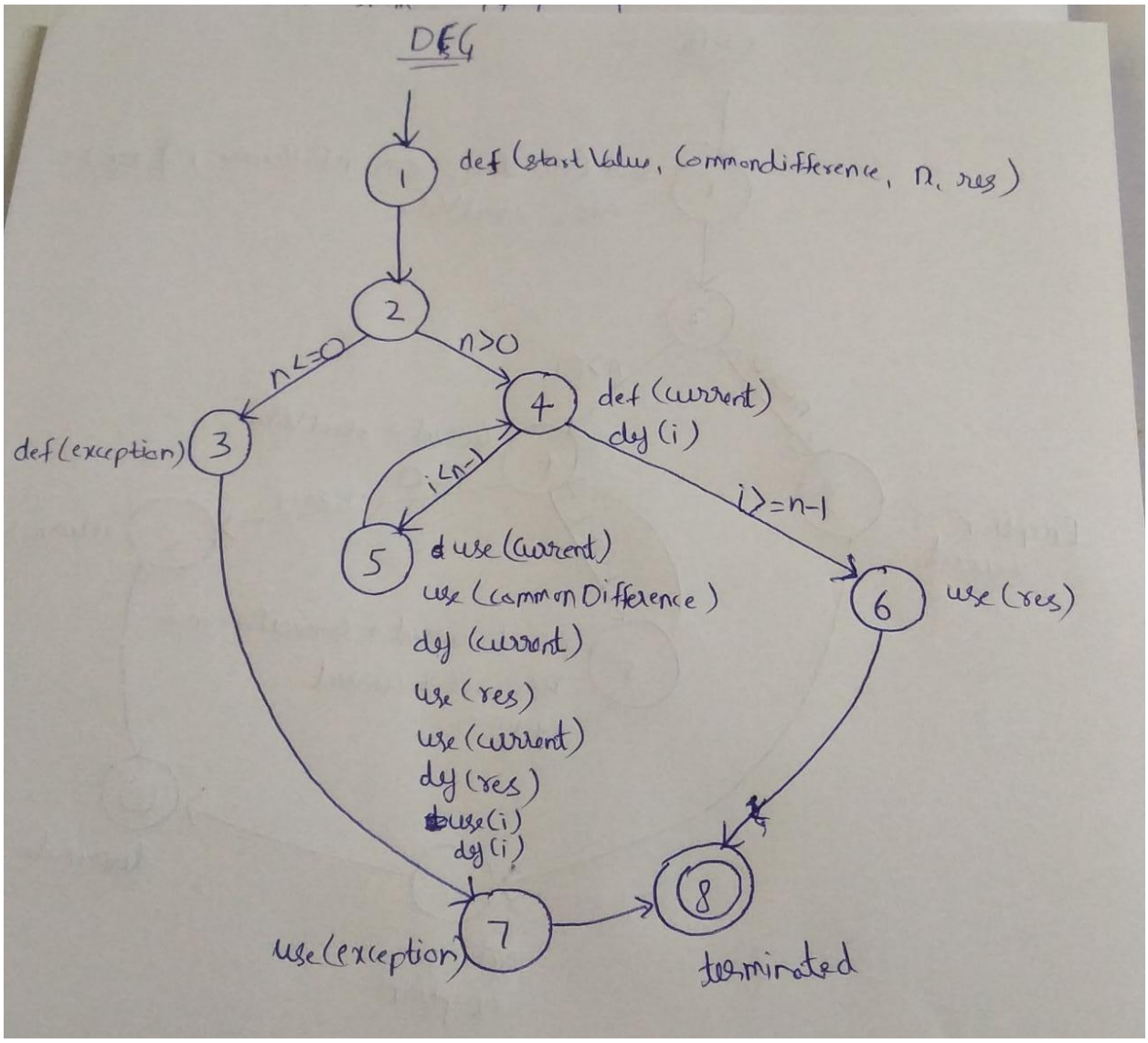
4. Find the first N terms of AP

```java
public String findFirstNTermsAP(double startValue, double commonDifference, int n)
{
    String res = Double.toString(startValue);
    try
    {
        if(n<=0)
        {
            throw new Exception();
        }
        double current = startValue;
        for(int i=0;i<n-1;i++)
        {
            current = current + commonDifference;
            res = res + "," + Double.toString(current);
        }
        return res;
    }catch(Exception e)
    {
        return "Invalid count input";
    }
}
```

CFG


CFG
1
read (startValue, commonDifference, n)
res = startValue
2
n<=0
n>0
3
4
current = startValue
i = 0
i<n-1
i≥n-1
5
6
return res
current = current + commonDifference
res = res + current
i++;
7
8
return exception
terminate

DFG

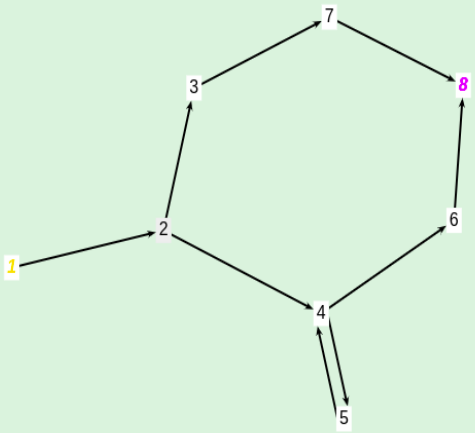

DFG
1
def (start Value, Commondifference, n, res)
2
n<=0
n>0
3
def(exception)
4
def (current)
def (i)
i<n-1
i>=n-1
5
use (current)
use (common Difference)
def (current)
use (res)
use (current)
def (res)
use(i)
def(i)
6
use (res)
7
use(exception)
8
terminated

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

DU Pairs for all variables are:

Variable	DU Pairs
startValue	[1,1] [1,4]
commonDifference	[1,5]
n	[1,2] [1,4]
res	[1,5] [1,6] [5,5] [5,6]
exception	[3,7]
current	[4,5] [5,5]
i	[4,4] [4,5] [5,4] [5,5]

Node color: Initial Node, Final Node



DU Paths for all variables are:

Variable	DU Paths
startValue	[1,2,4]
commonDifference	[1,2,4,5]
n	[1,2] [1,2,4]
res	[1,2,4,6] [1,2,4,5] [5,4,6] [5,4,5]
exception	[3,7]
current	[4,5]
i	[4,5] [5,4]

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
startValue	[1,2,4,6,8]
commonDifference	[1,2,4,5,4,6,8]
n	[1,2,3,7,8] [1,2,4,6,8]
res	[1,2,4,6,8] [1,2,4,5,4,6,8] [1,2,4,5,4,5,4,6,8]
exception	[1,2,3,7,8]
current	[1,2,4,5,4,6,8]
i	[1,2,4,5,4,6,8]

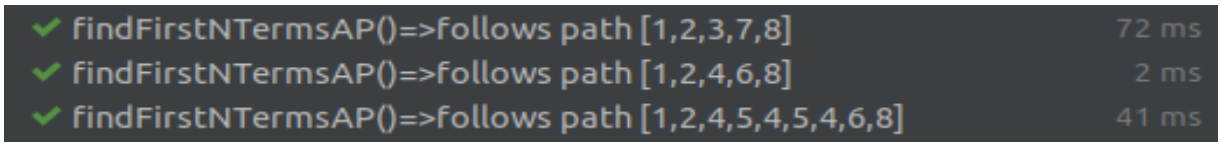
Here we have 3 unique paths:-

1. [1,2,3,7,8]
   Input:- startValue=1, commonDifference=3, n=-1
   Output:- Invalid count Input

2. [1,2,4,6,8]
   Input:- startValue=1, commonDifference=3, n=1
   Output:- 1.0

3. [1,2,4,5,4,5,4,6,8]
   Inputs:- startValue=1, commonDifference=3, n=3
   Output:- 1.0, 4.0, 7.0

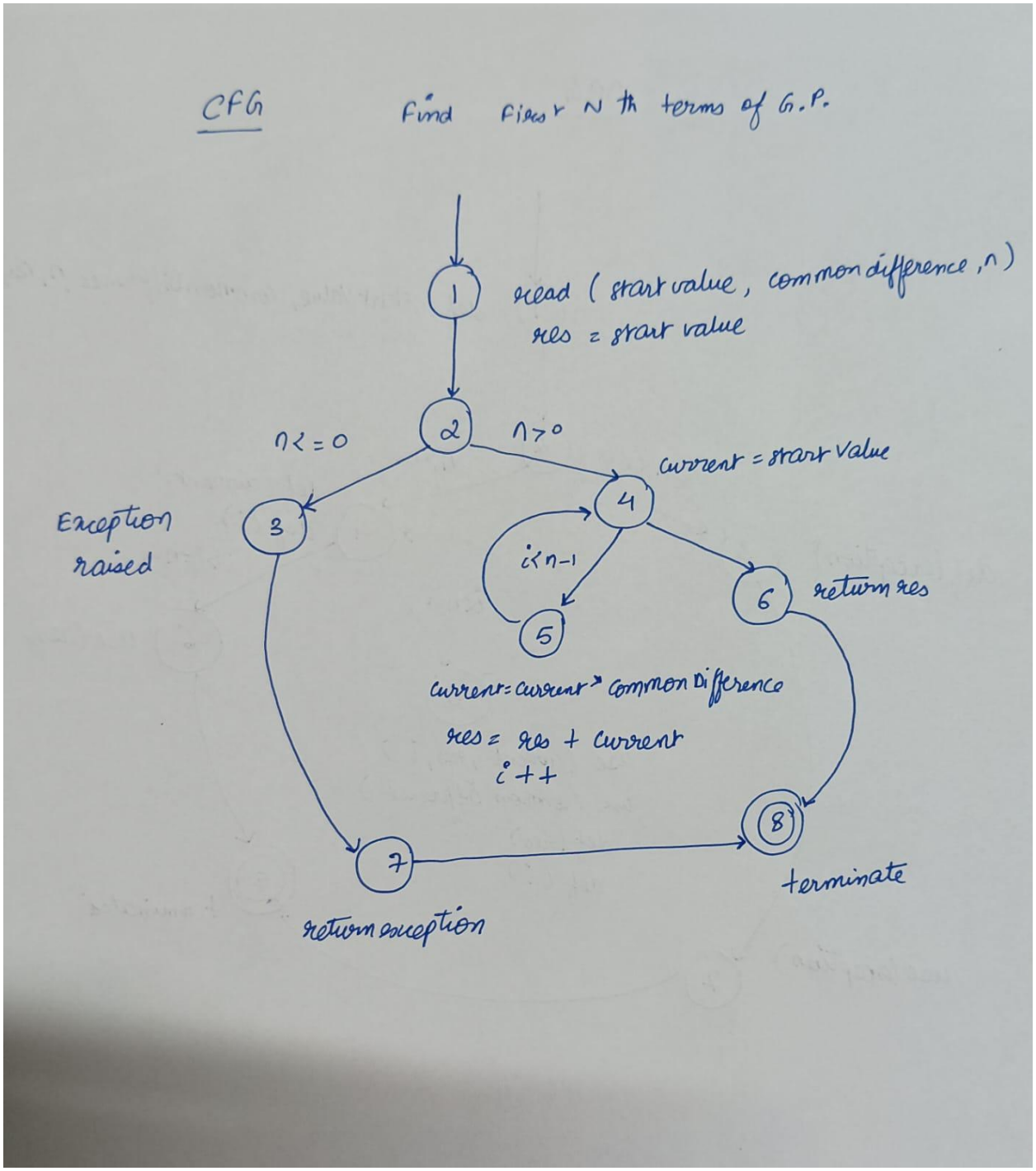
Result:-



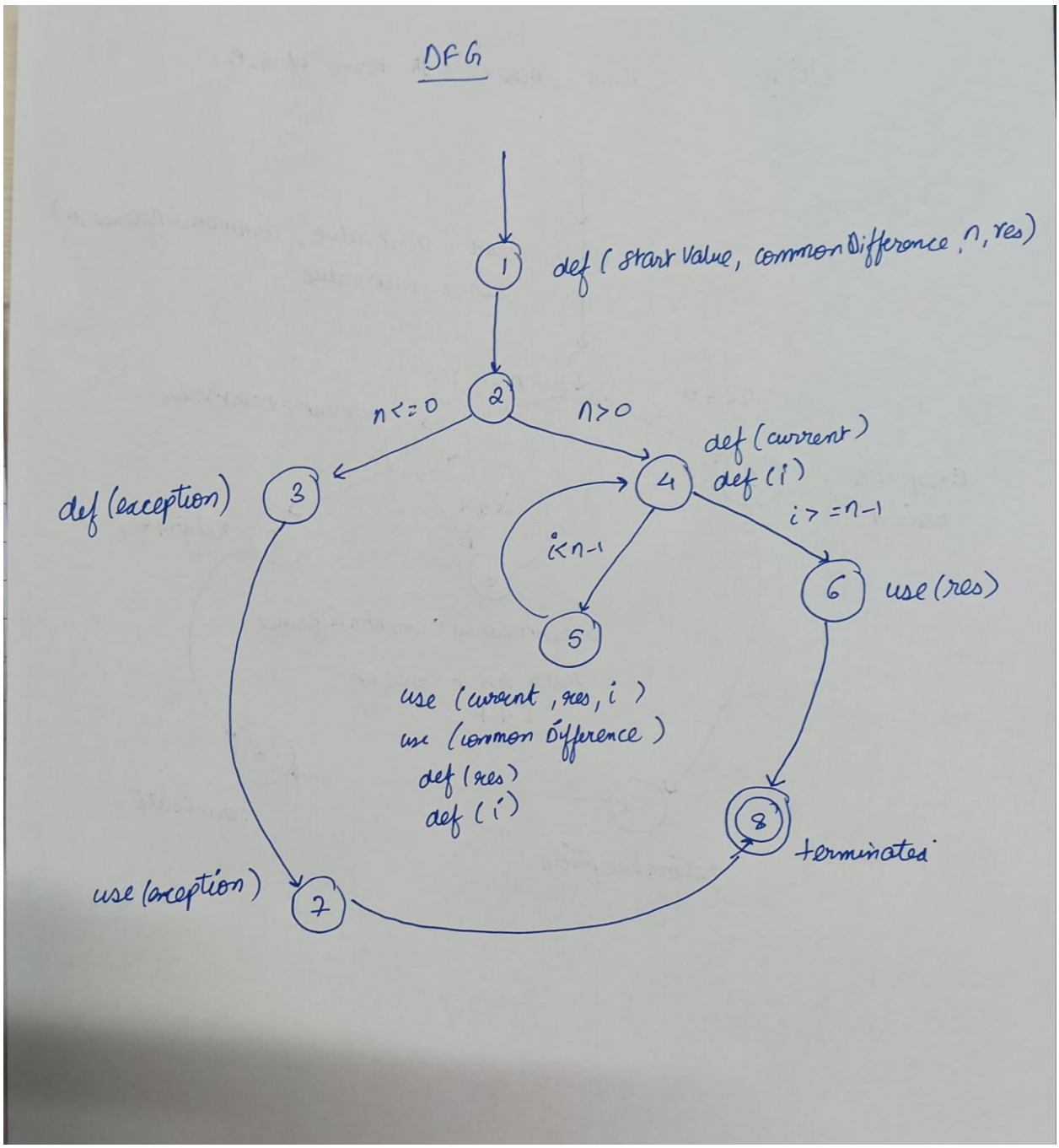
5. Find the First nth Terms of GP

```java
public String findFirstNTermsGP(double startValue, double commonDifference, int n)
{
    String res = Double.toString(startValue);
    try
    {
        if(n<=0)
        {
            throw new Exception();
        }
        double current = startValue;
        for(int i=0;i<n-1;i++)
        {
            current = current*commonDifference;
            res = res + "," + Double.toString(current);
        }
        return res;
    }catch(Exception e)
    {
        return "Invalid value of n";
    }
}
```

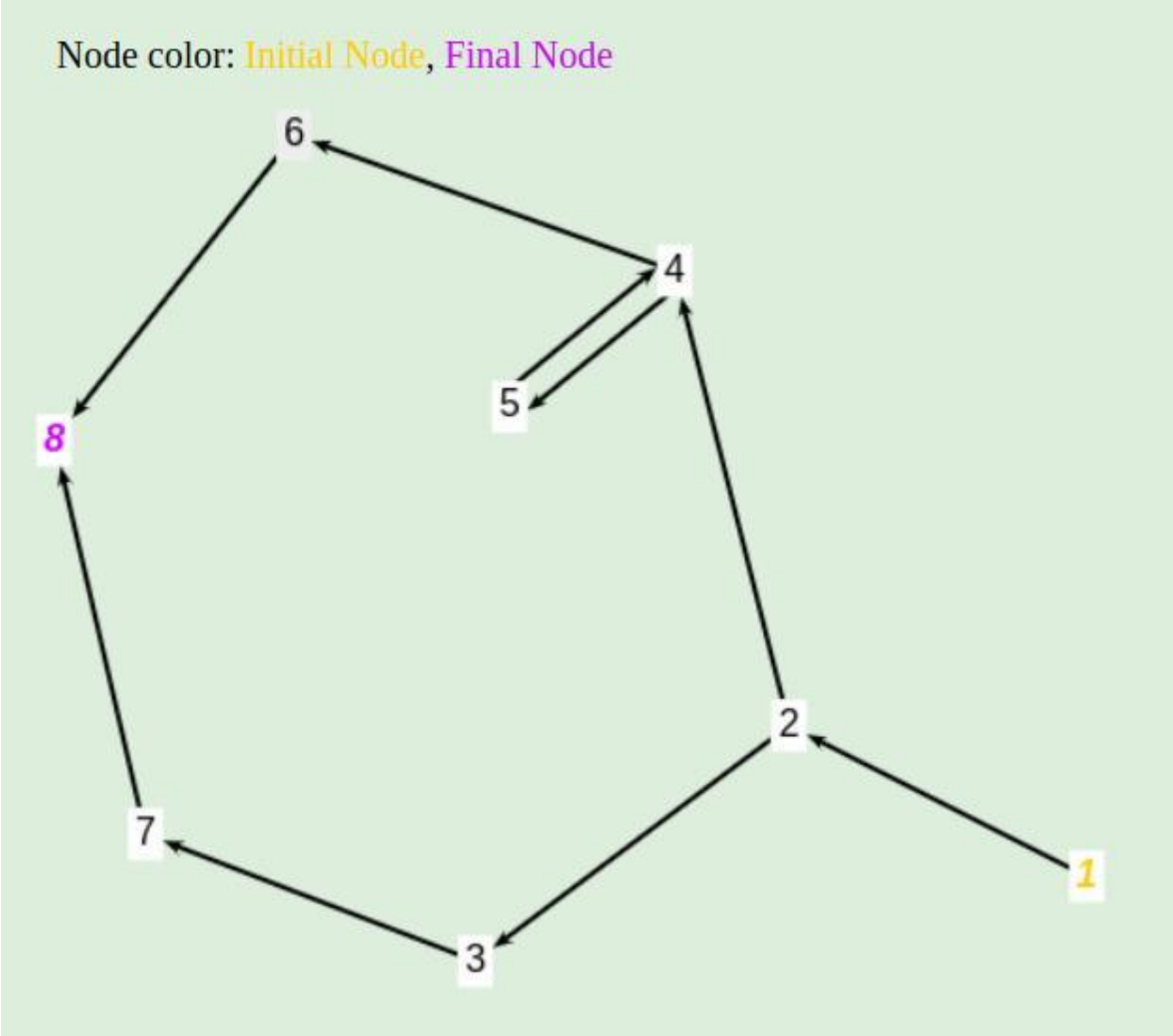
CFG


CFG
Find First N th terms of G.P.
1
read (start value, common difference, n)
res = start value
2
n<=0
n>0
3
Exception raised
4
current = start value
i<n-1
5
current = current + common Difference
res = res + current
i++
6
return res
7
return exception
8
terminate

DFG


DFG
1
def (start value, common Difference, n, res)
2
n<=0
n>0
3
def (exception)
4
def (current)
def (i)
i >=n-1
i<n-1
5
use (current, res, i)
use (common Difference)
def (res)
def (i)
6
use (res)
7
use (exception)
8
terminates

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



DU Paths for all variables are:

Variable	DU Paths
startvalue	[1,2,4]
commonDifference	[1,2,4,5]
n	[1,2]
current	[4,5]
i	[4,5]
res	[1,2,4,6] [1,2,4,5] [5,4,6] [5,4,5]
exception	[3,7]

DU Pairs for all variables are:

Variable	DU Pairs
startvalue	[1,4]
commonDifference	[1,5]
n	[1,2]
current	[4,5] [5,5]
i	[4,5] [5,5]
res	[1,6] [1,5] [5,6] [5,5]
exception	[3,7]

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
startvalue	[1,2,4,6,8]
commonDifference	[1,2,4,5,4,6,8]
n	[1,2,4,6,8]
current	[1,2,4,5,4,6,8]
i	[1,2,4,5,4,6,8]
res	[1,2,4,6,8] [1,2,4,5,4,6,8] [1,2,4,5,4,5,4,6,8]
exception	[1,2,3,7,8]

Here we have 3 unique paths:-

1. [1,2,4,5,4,6,8]

Input:- startValue=10, commonDifference=10, n=3
Output:- `10.0,100.0,1000.0`

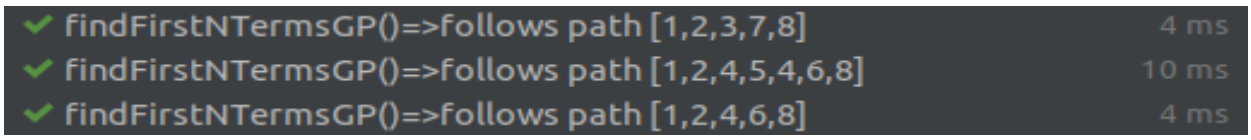
2. [1,2,4,6,8]

Input:- startValue=10, commonDifference=10, n=1
Output:- 10.0

3. [1,2,3,7,8]

Input:- startValue=10, commonDifference=10, n=0
Output:- `Invalid value of n`

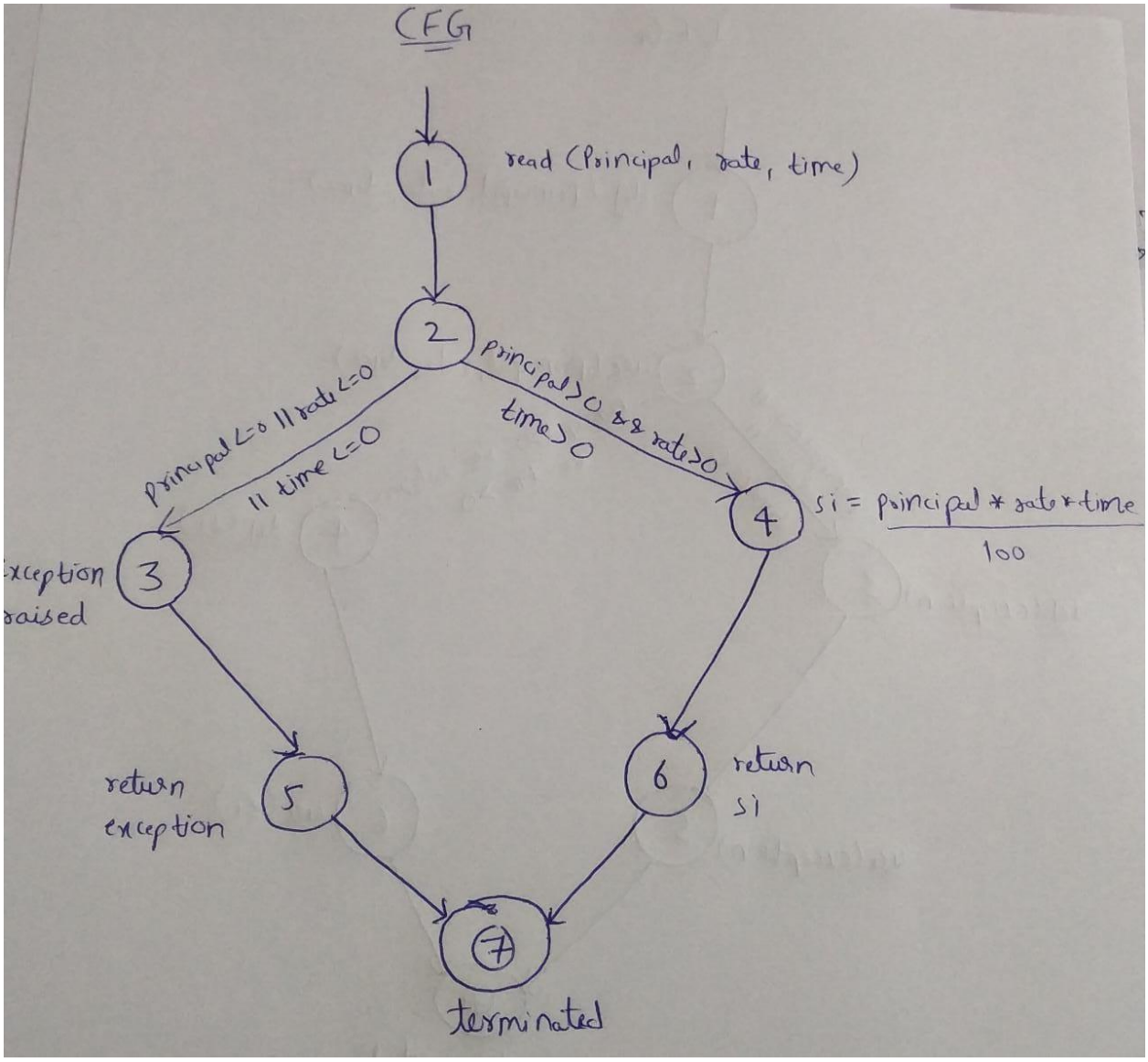
Result



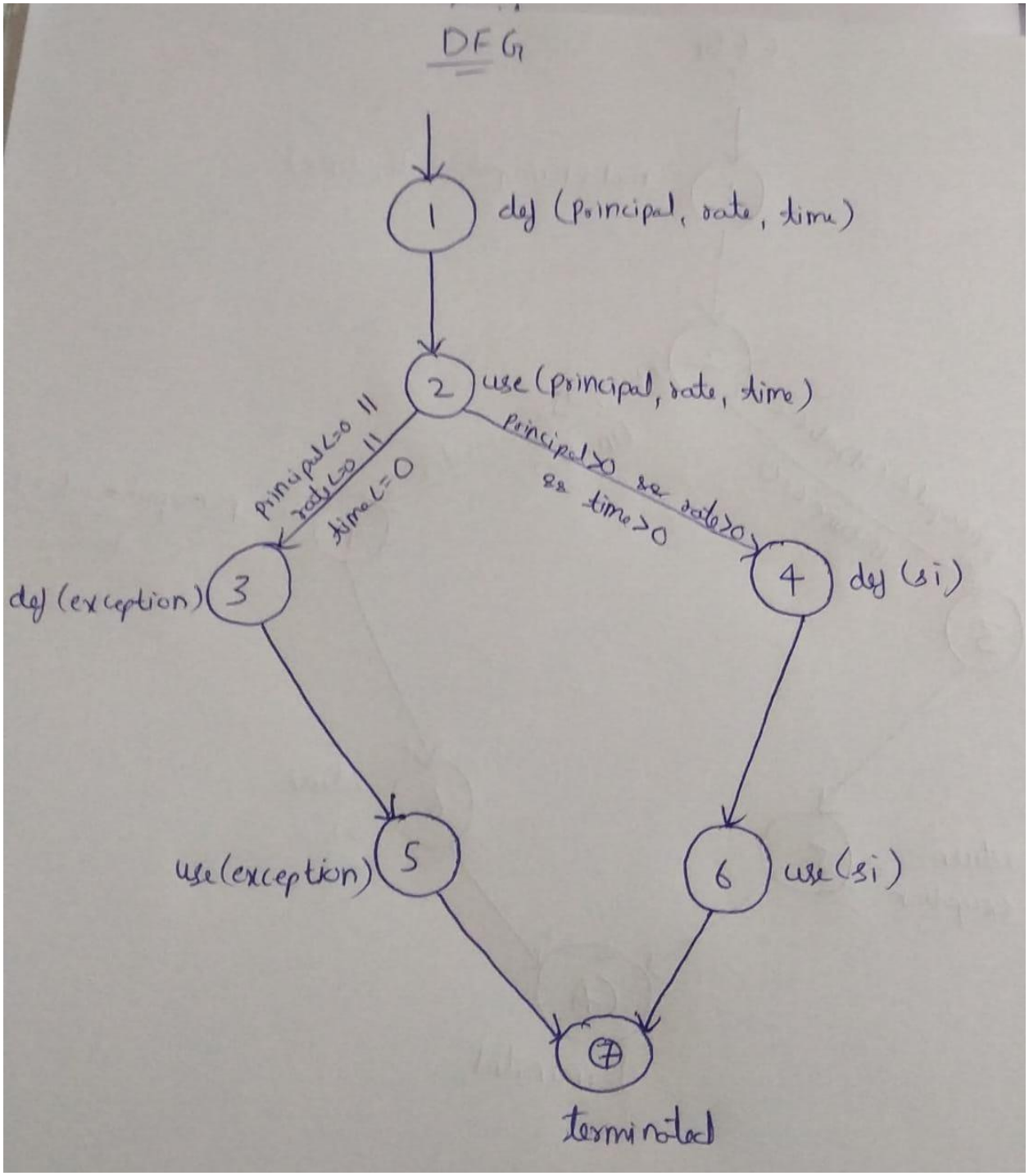
6. Find Simple Interest:-

```java
public String simpleInterest(double principal, double rate, double time)
{
    try {
        if (principal <= 0 || rate<=0 || time<=0) {
            throw new Exception();
        }
        double si = (principal * rate * time)/ 100;
        return Double.toString(si);
    }catch (Exception e)
    {
        return "Either principal value or rate value or time value is invalid";
    }
}
```

CFG


CFG
1
read (Principal, rate, time)
2
principal <=0 || rate <=0
|| time <=0
principal >0 && rate >0
time >0
4
si = principal * rate * time / 100
3
exception raised
5
return exception
6
return si
7
terminated

DFG

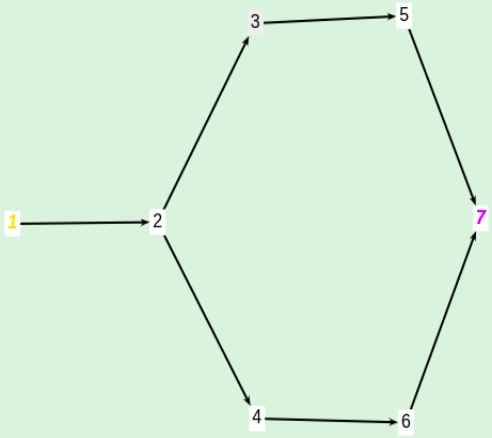

DFG
1
def (principal, rate, time)
2
use (principal, rate, time)
principal<=0 ||
rate<=0 ||
time<=0
Principal>0 && rate>0
&& time>0
3
def (exception)
4
def (si)
5
use (exception)
6
use (si)
7
terminated

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

DU Pairs for all variables are:

Variable	DU Pairs
principal	[1,2] [1,4]
rate	[1,2] [1,4]
time	[1,2] [1,4]
si	[4,6]
exception	[3,5]

Node color: Initial Node, Final Node



DU Paths for all variables are:

Variable	DU Paths
principal	[1,2] [1,2,4]
rate	[1,2] [1,2,4]
time	[1,2] [1,2,4]
si	[4,6]
exception	[3,5]

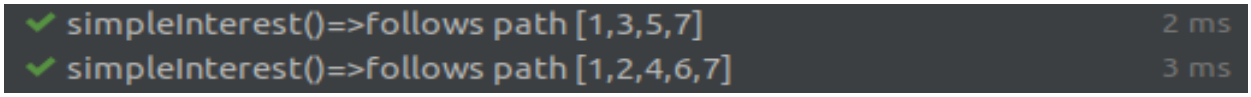
All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
principal	[1,2,3,5,7] [1,2,4,6,7]
rate	[1,2,3,5,7] [1,2,4,6,7]
time	[1,2,3,5,7] [1,2,4,6,7]
si	[1,2,4,6,7]
exception	[1,2,3,5,7]

Here we have two unique paths:-

1. [1,3,5,7]
   Input:- principal=10, rate=2, time=0
   Output:- Either the value of principal or the value of rate or the value of time is invalid.
2. [1,2,4,6,8]
   Input:- principal=100, rate=10, time=2
   Output:- 20.0

Result:-

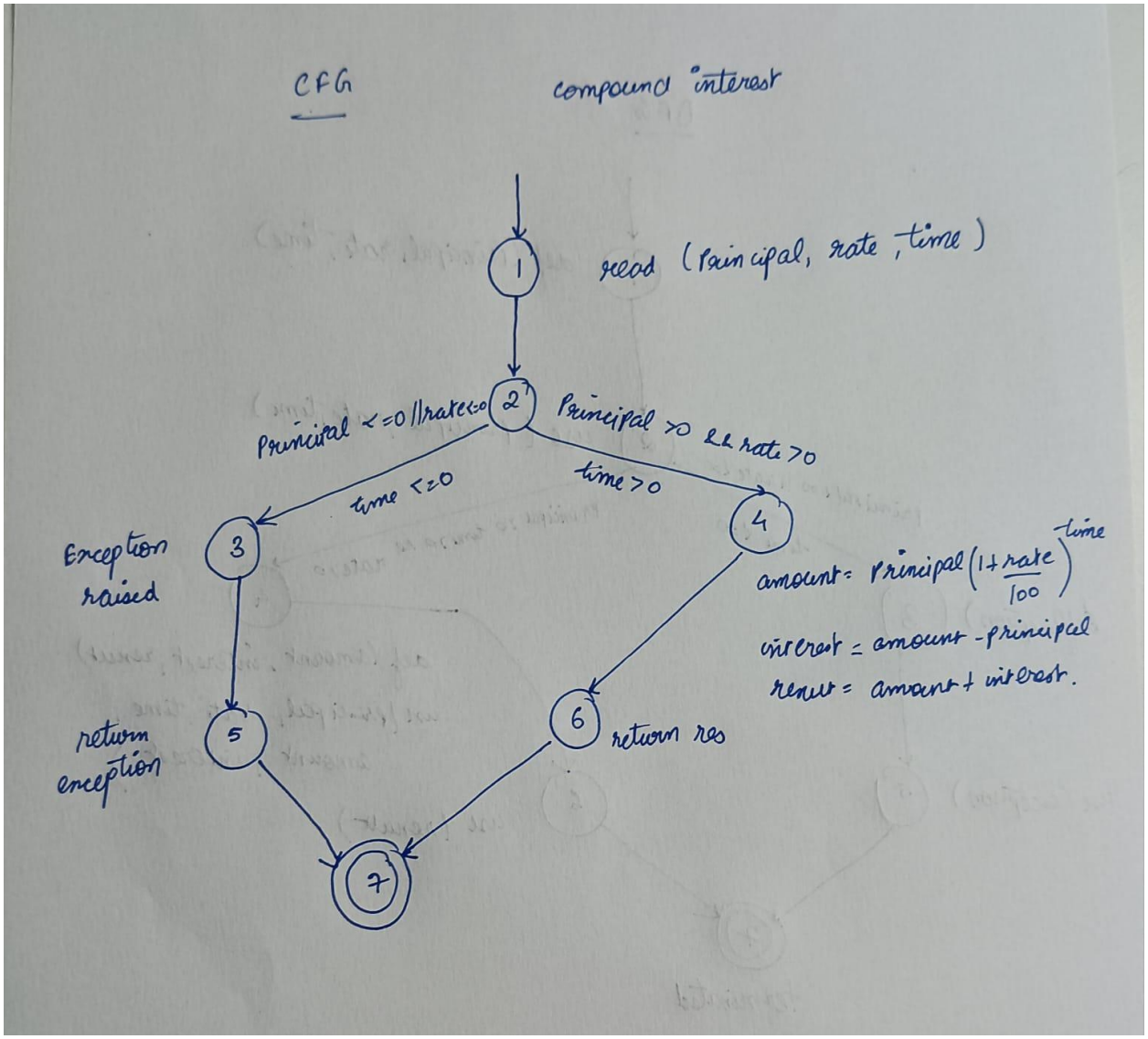


7. Compound Interest :-

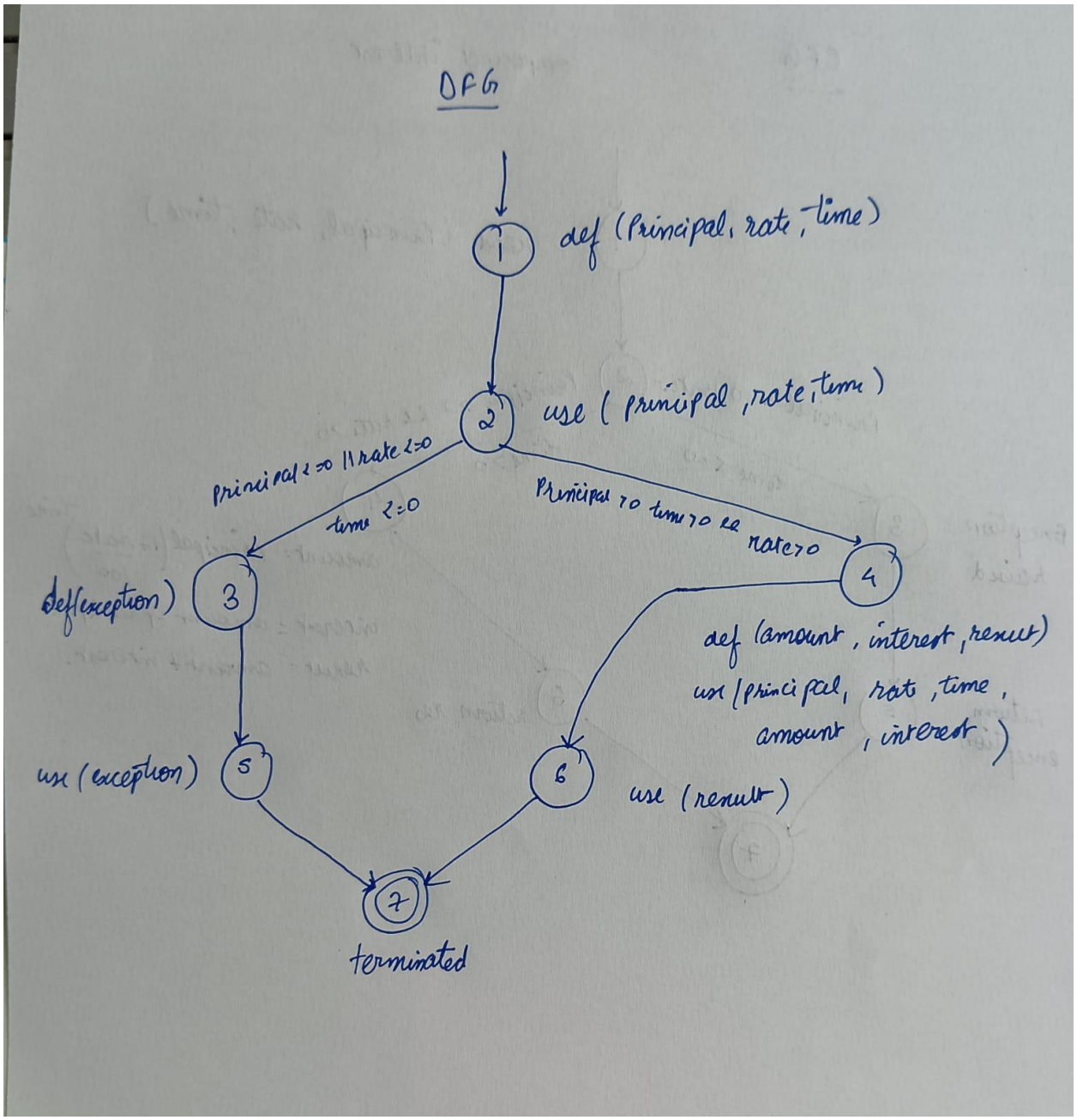
```java
public String compoundInterest(double principal, double rate, double time)
{
    String result;
    try {
        if (principal == 0 || rate==0 || time==0) {
            throw new Exception();
        }
        double amount = principal * (Math.pow((1 + rate/100), (time)));
        double interest = amount - principal;
        return result = "Amount is: " + amount + " Interest is: " + interest;
    }catch (Exception e)
    {
        return "Either principal value or rate value or time value is invalid";

    }
}
```

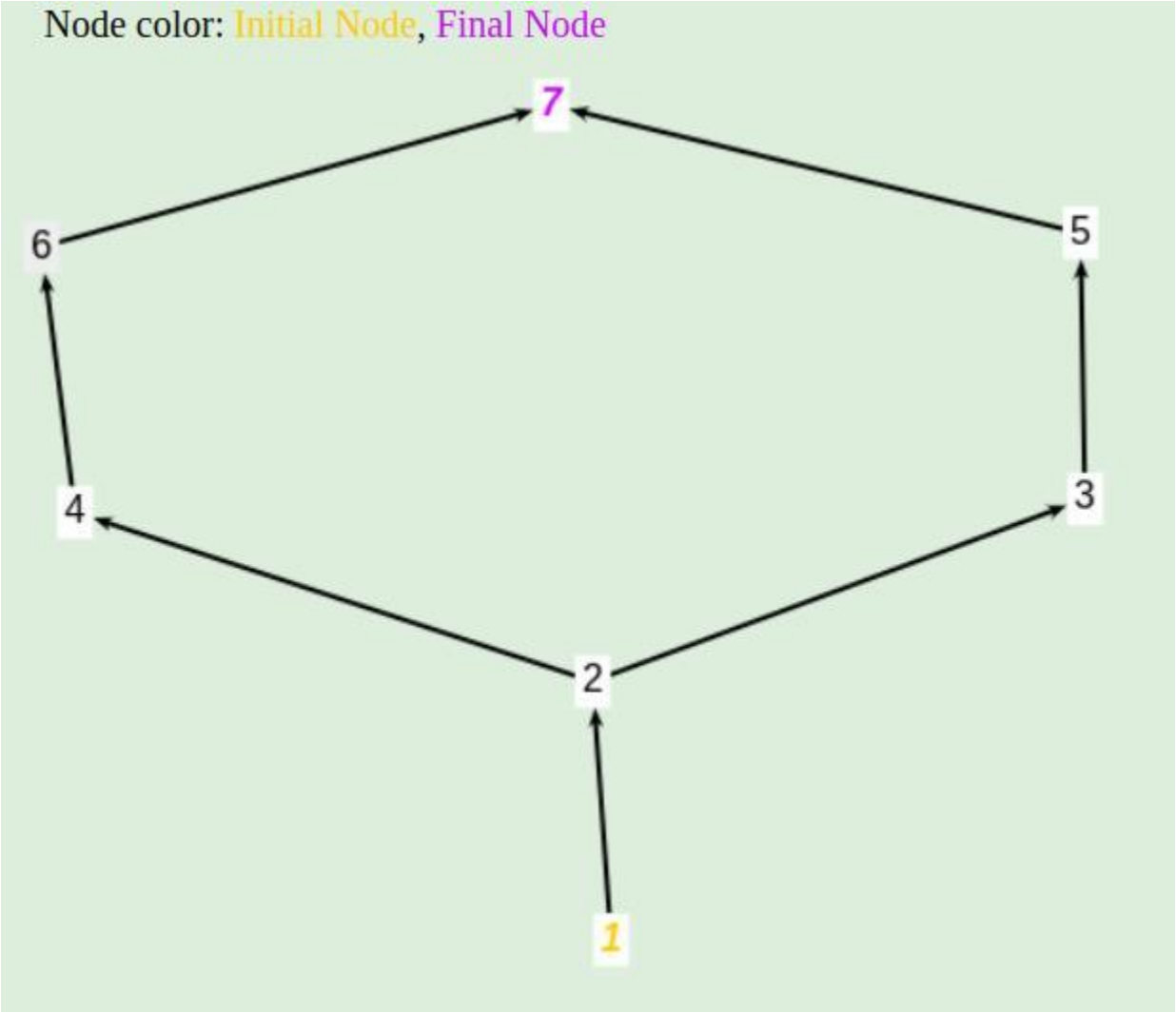
CFG


CFG
compound interest
1
read (principal, rate, time)
2
Principal <=0 || rate<=0
Principal >0 && rate >0
time <=0
time >0
3
Exception raised
4
amount = principal (1 + rate/100)^time
interest = amount - principal
return = amount + interest.
5
return exception
6
return res
7

DFG


DFG
1
def (Principal, rate, time)
2
use (principal, rate, time)
Principal<=0 || rate<=0
time<=0
Principal>0 time>0 && rate>0
3
def(exception)
4
def (amount, interest, result)
use (principal, rate, time,
amount, interest)
5
use (exception)
6
use (result)
7
terminated

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



DU Paths for all variables are:

Variable	DU Paths
principal	[1,2] [1,2,4]
rate	[1,2] [1,2,4]
time	[1,2] [1,2,4]
exception	[3,5]
amount	No path or No path needed
interest	No path or No path needed
result	[4,6]

DU Pairs for all variables are:

Variable	DU Pairs
principal	[1,2] [1,4]
rate	[1,2] [1,4]
time	[1,2] [1,4]
exception	[3,5]
amount	[4,4]
interest	[4,4]
result	[4,6]

All DU Path Coverage for all variables are:

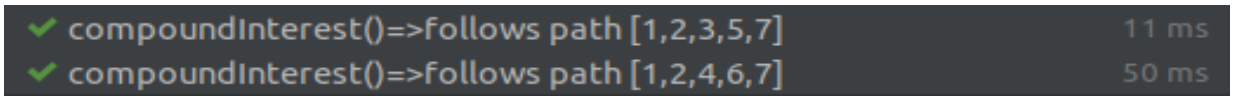
Variable	All DU Path Coverage
principal	[1,2,4,6,7]
rate	[1,2,4,6,7]
time	[1,2,4,6,7]
exception	[1,2,3,5,7]
amount	No path or No path needed
interest	No path or No path needed
result	[1,2,4,6,7]

Here we have two unique paths:-

1. [1,2,3,5,7]
   Input:- principal=0, rate=10, time=20
   Output:- Either the value of principal or the value of rate or the value of time is invalid.

2. [1,2,4,6,7]
   Input:- principal=1000, rate=10, time=5
   Output:- Amount is: 1610.5100000000004 Interest is: 610.5100000000004

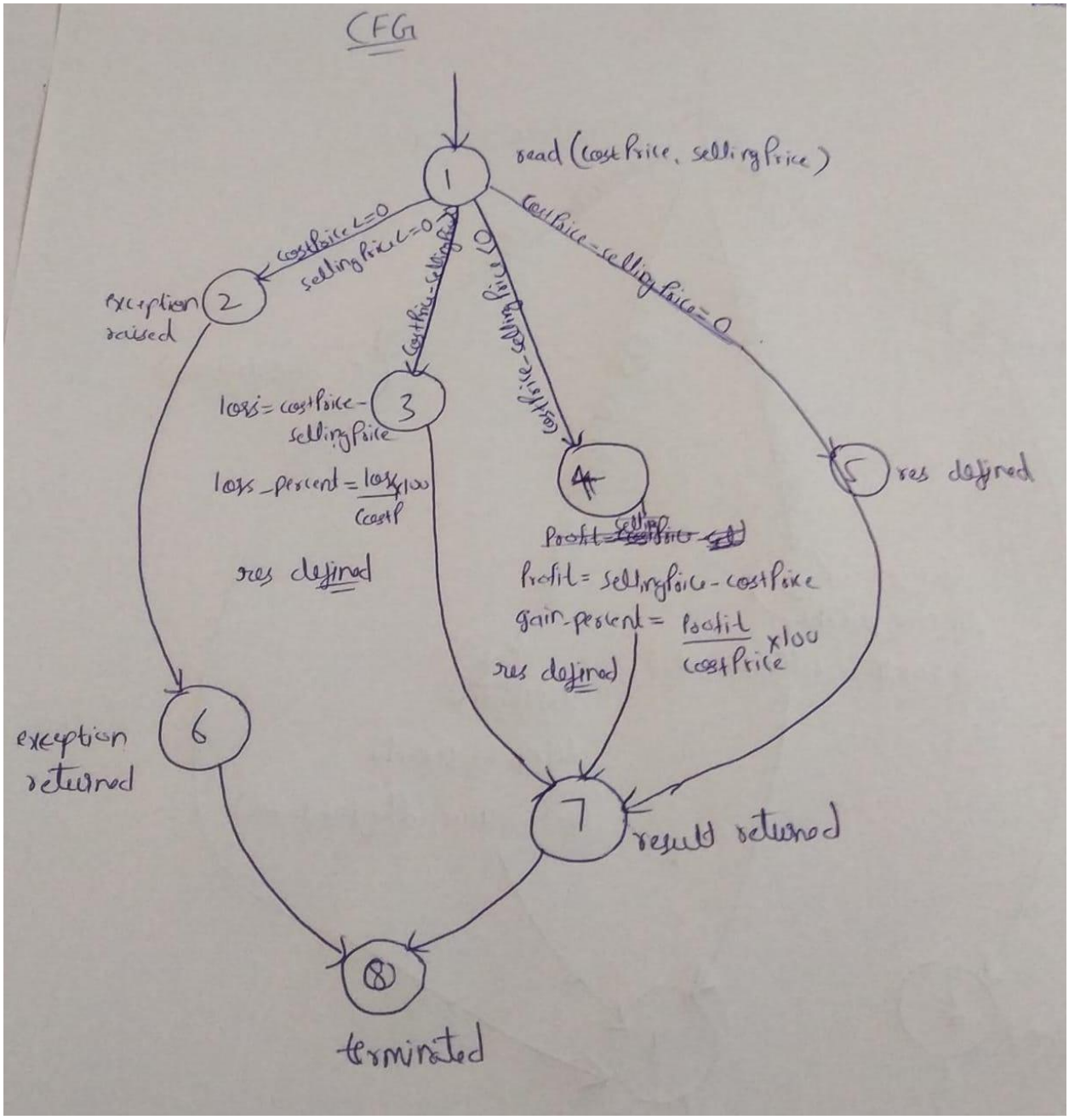
Result



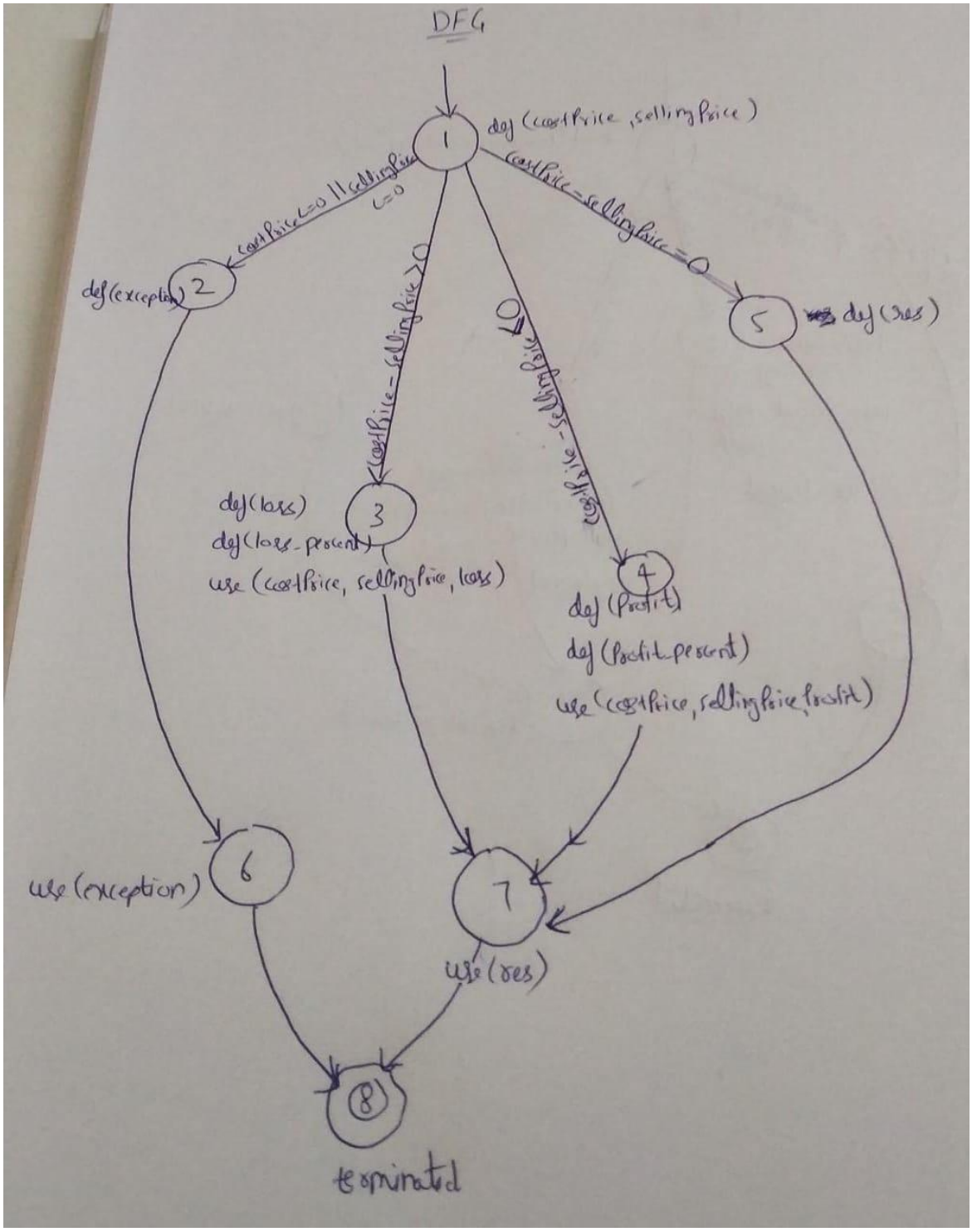
8. Find Profit and Loss:-

```java
public String compareProfitLoss(double costPrice,double sellingPrice)
{
    String res;
    try {
        if (costPrice <= 0 || sellingPrice<=0) {
            throw new Exception();
        }
        else if(costPrice-sellingPrice>0) {
            double loss=costPrice-sellingPrice;
            double loss_percent=(loss/costPrice)*100;
            res="loss: " + loss + ", loss percent: " +loss_percent+"%";
        }
        else if(costPrice-sellingPrice<0) {
            double gain=costPrice-sellingPrice;
            double gain_percent=(1 * gain/costPrice)*100;
            res="profit: "+ gain +", profit percent: " +gain_percent+"%";
        }
        else {
            res="No profit, No loss";
        }
        return res;
    }
    catch (Exception e)
    {
        return "Either costPrice value or sellingPrice value is invalid";
    }
}
```

CFG


CFG
read (costPrice, sellingPrice)
1
costPrice <= 0
sellingPrice <= 0
costPrice - sellingPrice < 0
costPrice - sellingPrice > 0
costPrice - sellingPrice = 0
exception raised
2
loss = costPrice - sellingPrice
loss_percent = loss x100 / costP
res defined
3
4
Profit = sellingPrice - costPrice
gain_percent = Profit / costPrice x100
res defined
5
res defined
exception returned
6
7
result returned
8
terminated

DFG

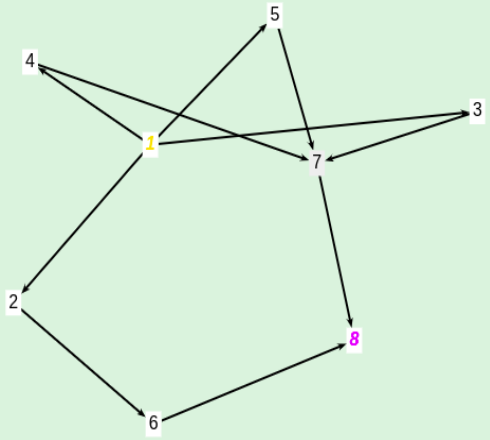

DFG
def (costPrice, sellingPrice)
1
costPrice<=0 || sellingPrice <=0
costPrice - sellingPrice = 0
costPrice - sellingPrice > 0
costPrice - sellingPrice < 0
def(exception)
2
5
def (res)
def(loss)
def(loss_precent)
use (costPrice, sellingPrice, loss)
3
4
def (Profit)
def (Profit_precent)
use (costPrice, sellingPrice, Profit)
use (exception)
6
7
use (res)
8
terminated

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

DU Pairs for all variables are:

Variable	DU Pairs
costPrice	[1,1] [1,3] [1,4]
sellingPrice	[1,1] [1,3] [1,4]
exception	[2,6]
loss	[3,3]
losspercent	[3,3]
profit	[4,4]
profitpercent	[4,4]
res	[3,7] [4,7] [5,7]

Node color: Initial Node, Final Node



DU Paths for all variables are:

Variable	DU Paths
costPrice	[1,3] [1,4]
sellingPrice	[1,4] [1,3]
exception	[2,6]
loss	No path or No path needed
losspercent	No path or No path needed
profit	No path or No path needed
profitpercent	No path or No path needed
res	[3,7] [4,7] [5,7]

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
costPrice	[1,3,7,8] [1,4,7,8]
sellingPrice	[1,4,7,8] [1,3,7,8]
exception	[1,2,6,8]
loss	No path or No path needed
losspercent	No path or No path needed
profit	No path or No path needed
profitpercent	No path or No path needed
res	[1,3,7,8] [1,4,7,8] [1,5,7,8]

Here we have total 4 unique paths:-

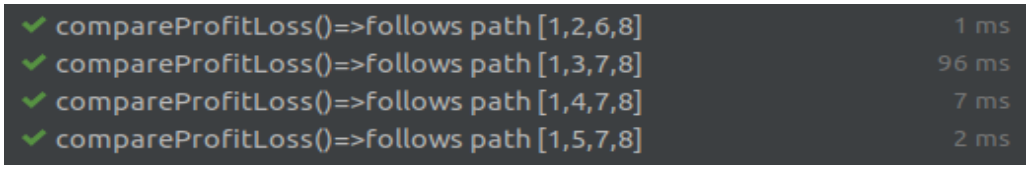
1. [1,2,6,8]
   Input:- costPrice=-10, sellingPrice=100
   Output:- Either costPrice value or sellingPrice value is invalid.

2. [1,3,7,8]
   Input:- costPrice=100, sellingPrice=80
   Output:- loss: 20.0, loss percent: 20.0%

3. [1,4,7,8]
   Input:- costPrice=100, sellingPrice=120
   Output:- profit: 20.0, profit percent: 20.0%

4. [1,5,7,8]
   Input:- costPrice=100, sellingPrice=100
   Output:- No profit, No loss

Result



Contributions:-

1. **Code, DFGs & CFGs, Du pairs, Du paths, and All Du path coverage TR and Test case design of percentageChange(), findFirstNTermsAP(), simpleInterest(), and compareProfitLoss() done by Gopal Goyal(MT2021048).**

2. **Code, DFGs & CFGs, Du pairs, Du paths, and All Du path coverage TR and Test case design of fractionToPercentageConverter(), compoundInterest(), findFirstNTermsGP(), and getCountOfDays() done by Pranay Jain(MT2021098).**

****