Cyclomatic Complexity

Lecture Objectives:

1) Explain the different levels of testing and the number of test cases required to meet the criteria.
2) Calculate the Cyclomatic complexity for a source code module based on a control flow graph.
3) Define the term basis path.
4) Construct test cases from the control flow graph which fully exercise the software module.
5) Critique Control flow testing, listing its advantages and disadvantages versus other testing techniques.
6) Visualize the relationship between cyclomatic complexity and the probability of fault manifestation.

Structured Testing

- Also referred to as basis path testing
- Uses the topology of the control flow graph to identify test cases

map

- Steps
  1. Derive the control flow graph
  2. Compute the Cyclomatic Complexity of the graph
  3. Select a set of C basis paths
  4. Create a test case for each basis path
  5. Execute these tests

Yes we can do it!

Happy ever after . . .

Basic Path Set

- An execution path is a set of nodes and directed edges in a flow graph that connects (in a directed fashion) the start node to a terminal node.
- Two execution paths are said to be independent if they do not include the same set of nodes and edges.
- A basic set of execution paths for a flow graph is an independent maximum set of paths in which all nodes and edges of the graph are included at least once.

Cyclomatic Complexity

The maximum size of a set of independent paths is unique for a given graph and is called the cyclomatic number.

MCC = McCabe's Cyclomatic Complexity

$$v(G) = e - n + 2$$

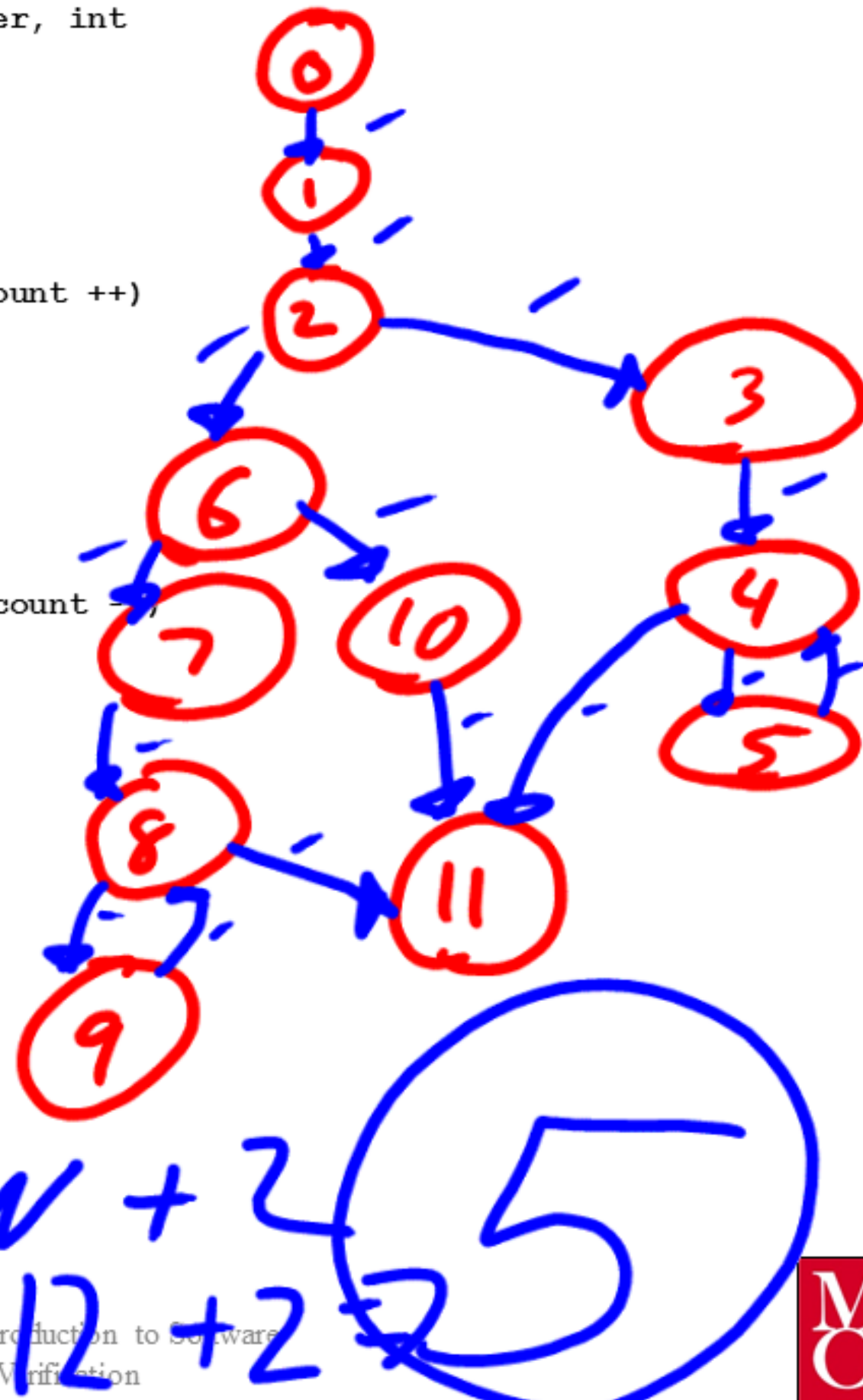
Where $v(G)$ denotes the cyclomatic number of graph G, n is the number of vertices in G, e is the number of edges.

Cyclomatic Complexity Criterion

- A set *P* of execution paths satisfies *cyclomatic number criterion* if and only if *P* contains at least one set of *v* independent paths, where $v = e - n + 2$ is the cyclomatic number of the flow graph.

Example

```
0  public static double power(double number, int
   power)
   {
1    double retVal = 0;
2    if (power > 0)
     {
3      retVal = number;
4      for (int count=1; count < power; count ++)
       {
5        retVal *= number;
       }
     }
6    else if (power < 0)
     {
7      retVal = (1.0 / number);
8      for (int count=-1; count > power; count --)
       {
9        retVal /= number;
       }
     }
     else
     {
10     retVal = 1.0;
     }
11   return retVal;
   }
```

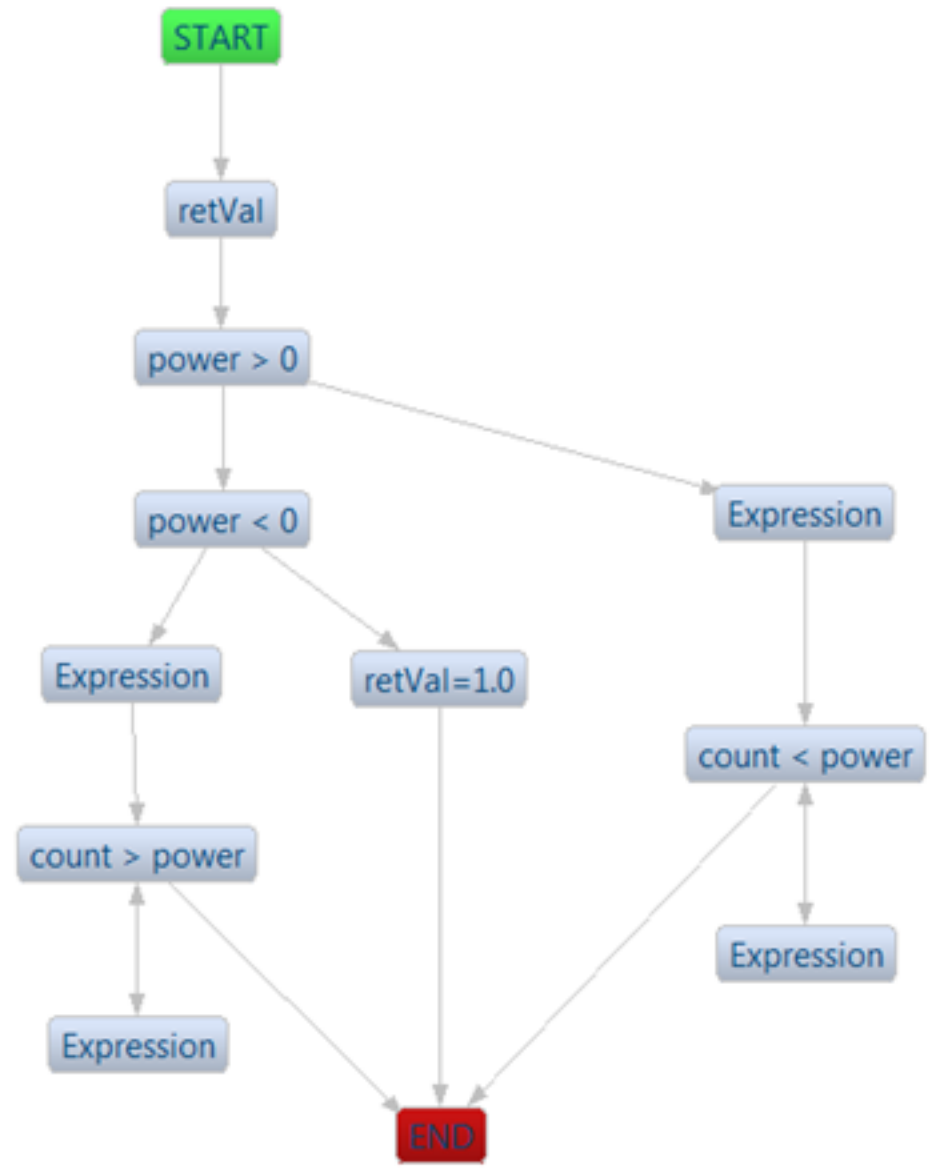


$$MCL = E - N + 2$$
$$= 15 - 12 + 2$$
$$= 5$$

MSOE

Example

```
public static double power(double number, int
power)
{
  double retVal = 0;
  if (power > 0)
  {
    retVal = number;
    for (int count=1; count < power; count ++)
    {
      retVal *= number;
    }
  }
  else if (power < 0)
  {
    retVal = (1.0 / number);
    for (int count=-1; count > power; count --)
    {
      retVal /= number;
    }
  }
  else
  {
    retVal = 1.0;
  }
  return retVal;
}
```



Eclipse Flowchart Generator.

Flow Chart Generator

MacCabe results
15 - 12 + 2 = 5
*Satisfied : true

Nodes:	12
Connectio...	15
Limit:	7

MSOE

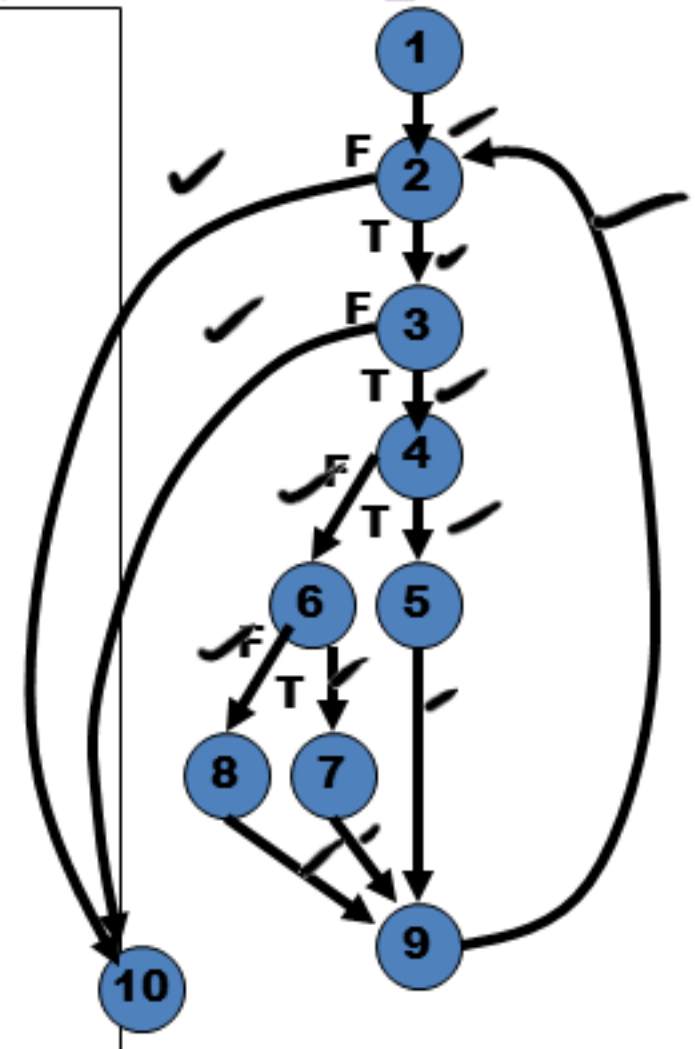
Binary Search CFG

$$MCC = E - N + 2$$

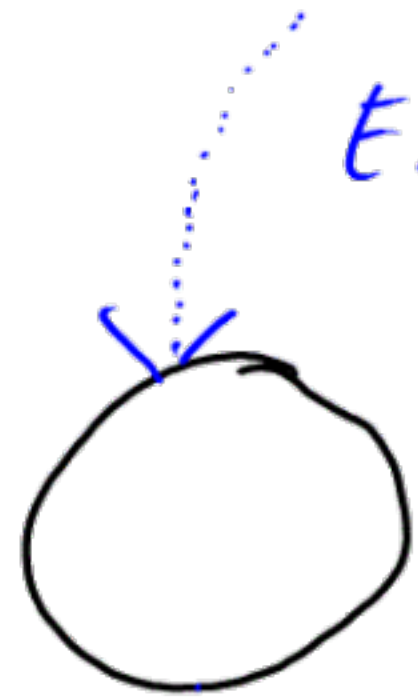
$$13 - 10 + 2 = 5$$

```
public static int binarySearch( int key, int[] sequence ) {
  int bottom = 0;
  int top = sequence.length - 1;
  int mid = 0;
  int keyPosition = -1;

  while( bottom <= top && keyPosition == -1 ) {
    mid = ( top + bottom ) / 2;
    if( sequence[ mid ] == key ) {
      keyPosition = mid;
    }
    else {
      if( sequence[ mid ] < key ) {
        bottom = mid + 1;
      }
      else {
        top = mid - 1;
      }
    }
  }
  return keyPosition;
}
```



// : return the number of ST's graduating
int test(){ instead of 4 @ most:
{
return 0; Else going in
}



CL = E − N + 2 ; Else going out

= 0 − 1 ~~+2~~

⇒ 1 ⇒ −1

boolean isODD(int)

{

if(x%2==0)

{ return false

}

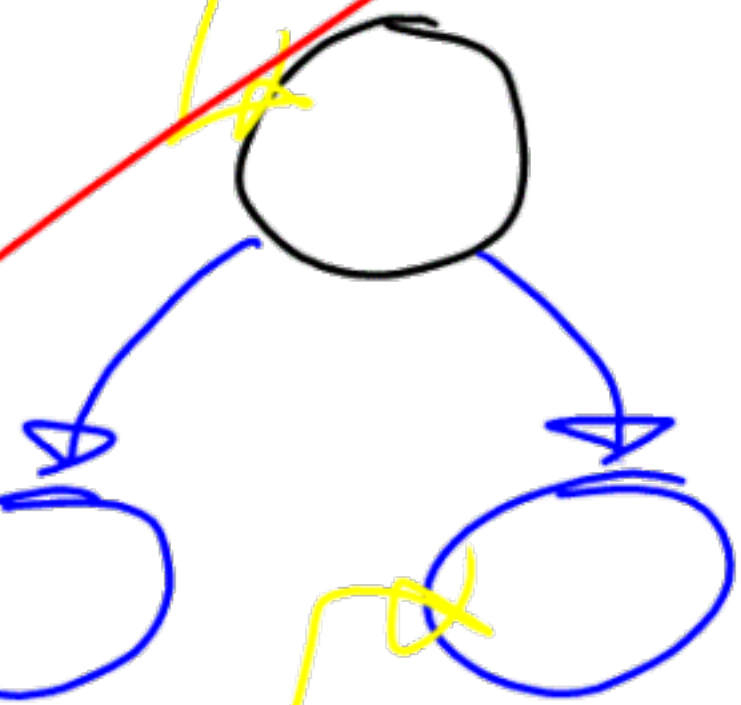
else

{ return true;}

}

E-N

2-3+2 ⇒



boolean isEven(int x)

#define boolean bool

{ boolean retV;

if(x % 2 == 0)

{ retV = true; }

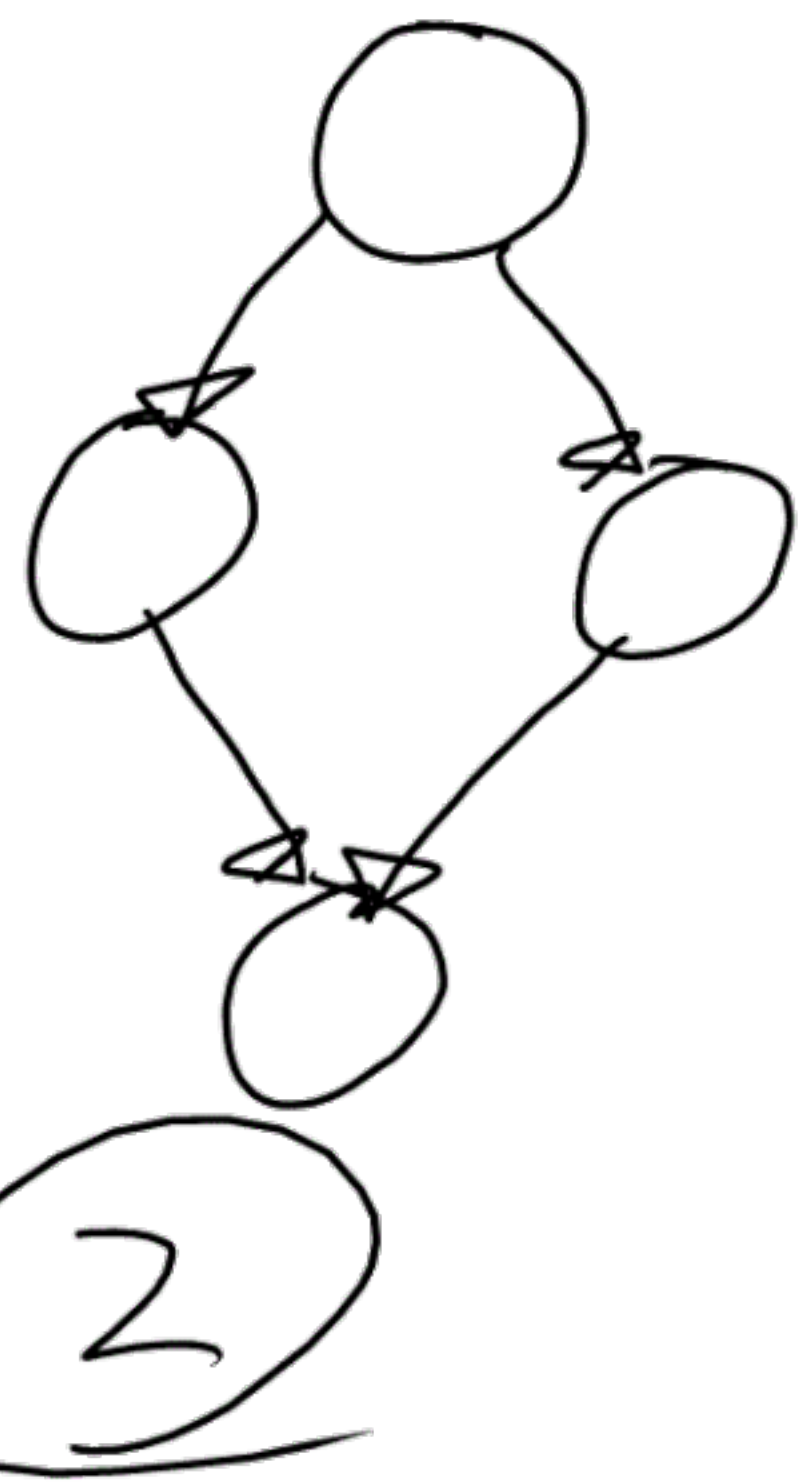
else { retV = false; }

return retV;

}

E - N + 2

4 - 4 + 2 =>

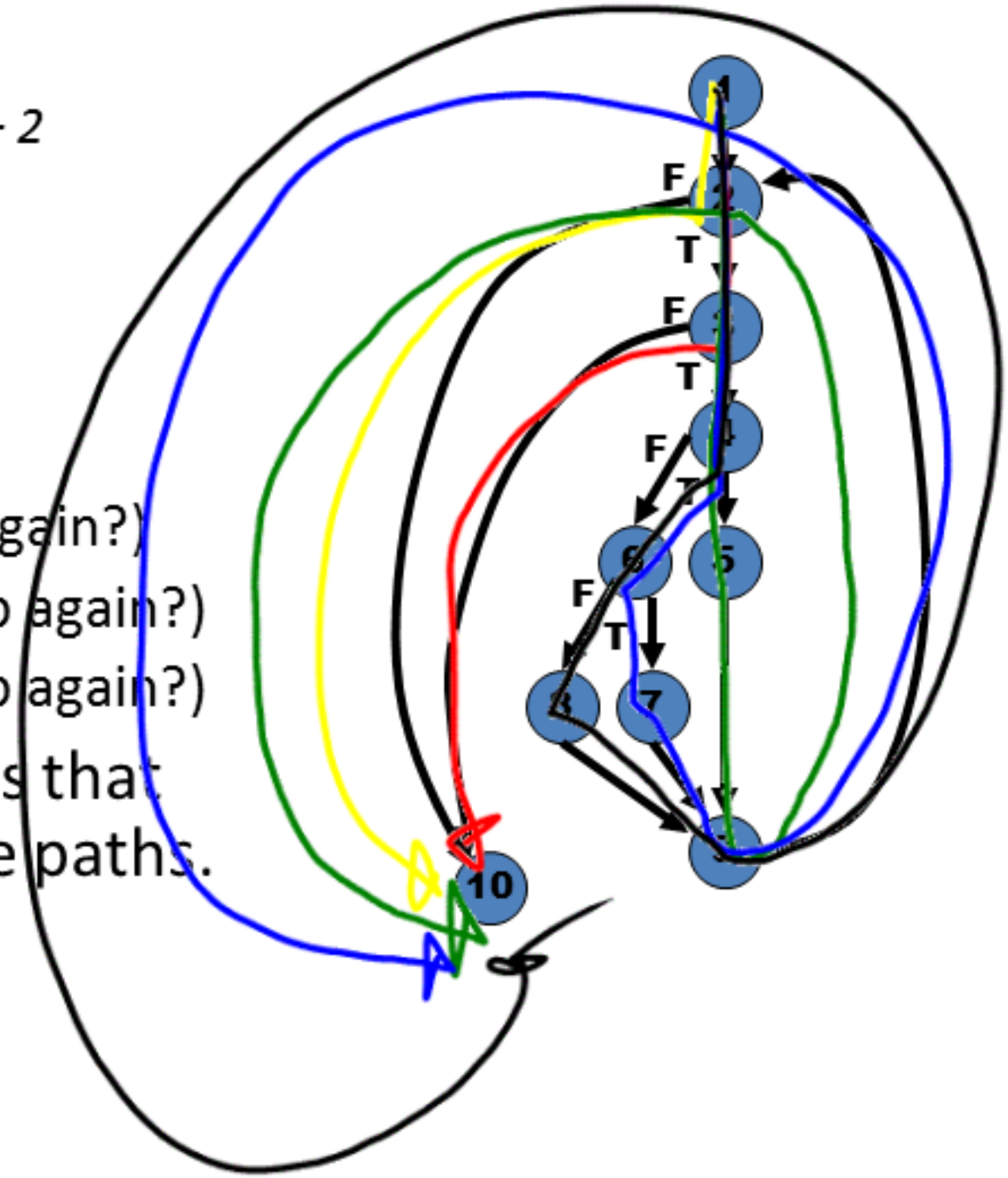


Creating Test Cases

- Work out the number of distinct paths.
  - Cyclomatic Complexity

    CC = noEdges – noNodes + 2

    CC = 13 – 10 + 2 = 5
- List the distinct paths.
  - 1, 2, 10
  - 1, 2, 3, 10
  - 1, 2, 3, 4, 5, 9, 2… (loop again?)
  - 1, 2, 3, 4, 6, 7, 9, 2… (loop again?)
  - 1, 2, 3, 4, 6, 8, 9, 2… (loop again?)
- Figure out the conditions that cause execution of these paths.



Your exercise:

Draw a flow graph for this piece of source code

Determine the cyclomatic complexity of the code

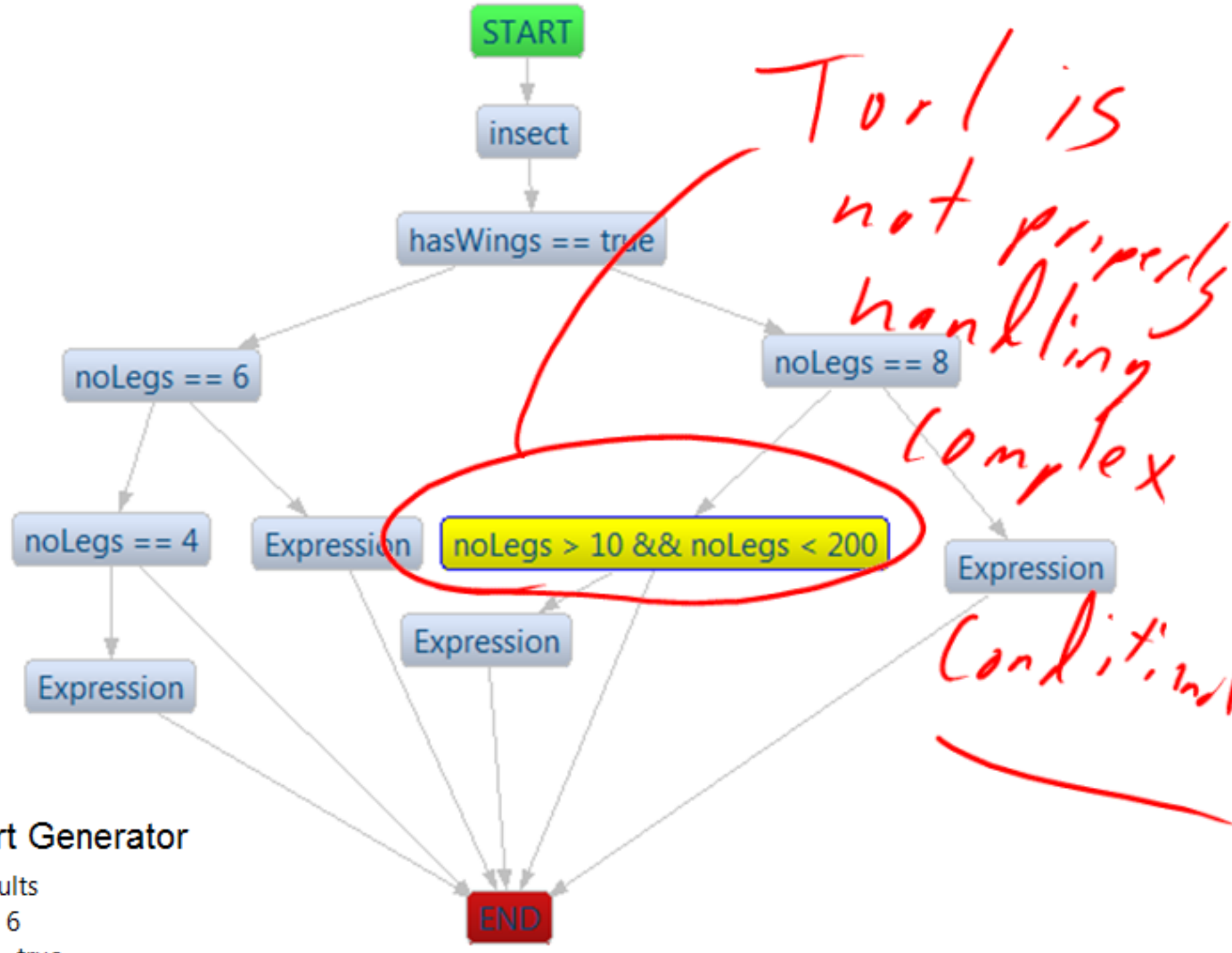
```
String identifyInsect(boolean  hasWings, int noLegs) {
    String insect = "Unknown";
    if (hasWings == true) {
        if (noLegs == 6)
            insect = "Fly";
        else if (noLegs == 4)
            insect = "Grass Hopper";
    } else {
        if (noLegs  == 8)
            insect = "Spider";
        else if (noLegs > 10 && noLegs < 200)
            insect = "Millipede";
    }
    return insect;
}
```

4 1111

6 1

7 1

Tool Answer



Flow Chart Generator

MacCabe results

16 - 12 + 2 = 6

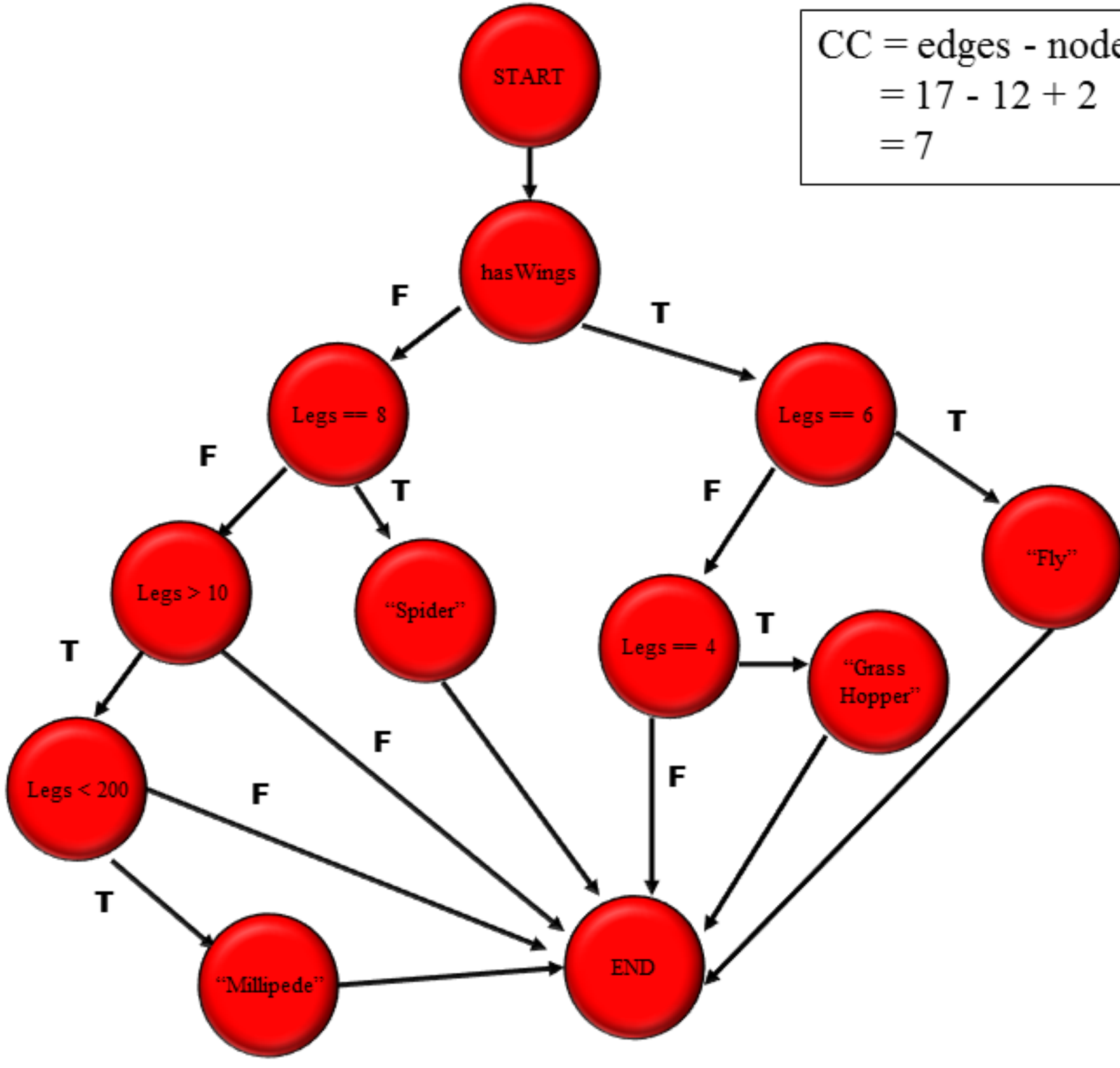
*Satisfied : true

Nodes:	12
Connectio...	16
Limit:	

MSOE

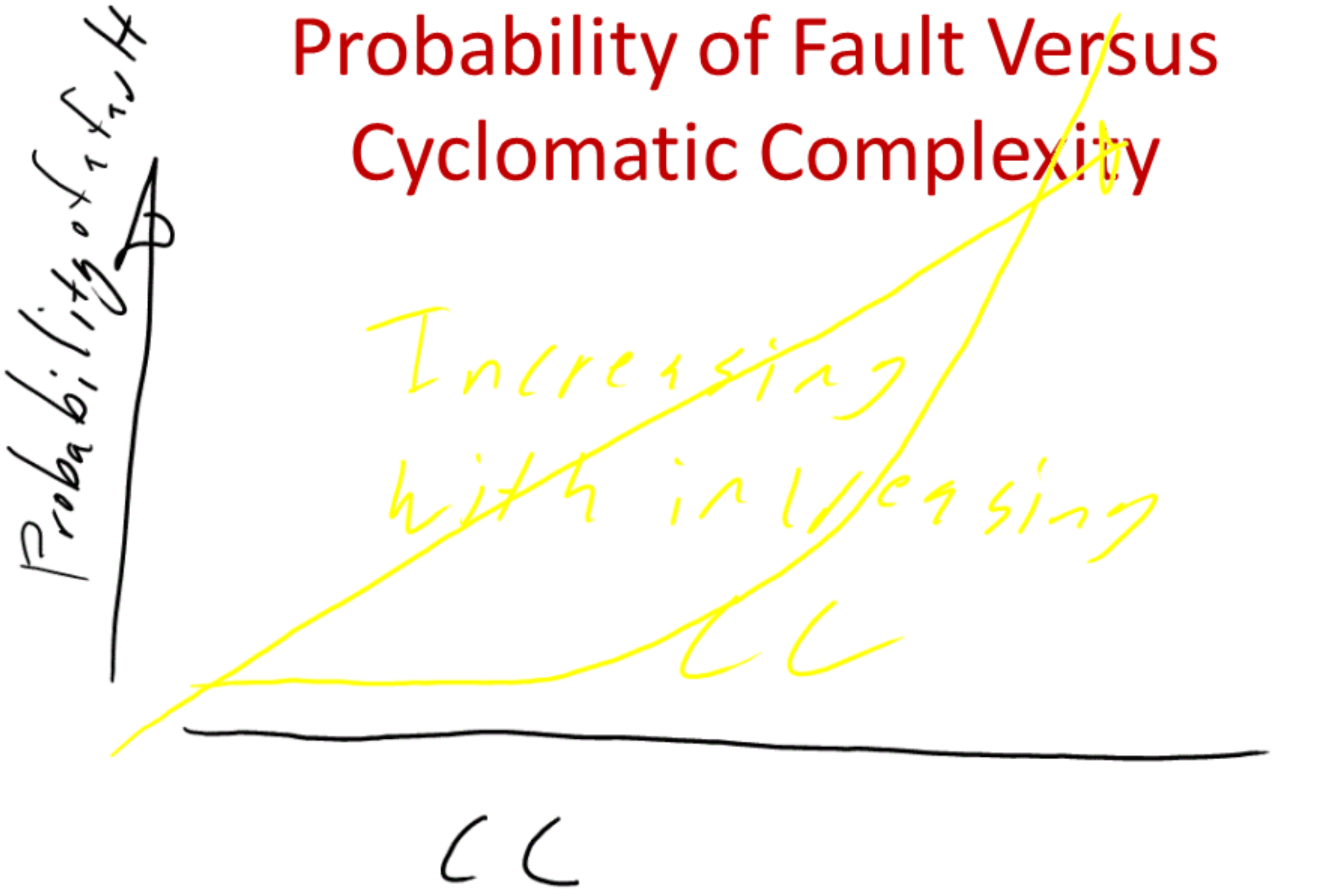
Answer

CC = edges - nodes + 2
= 17 - 12 + 2
= 7

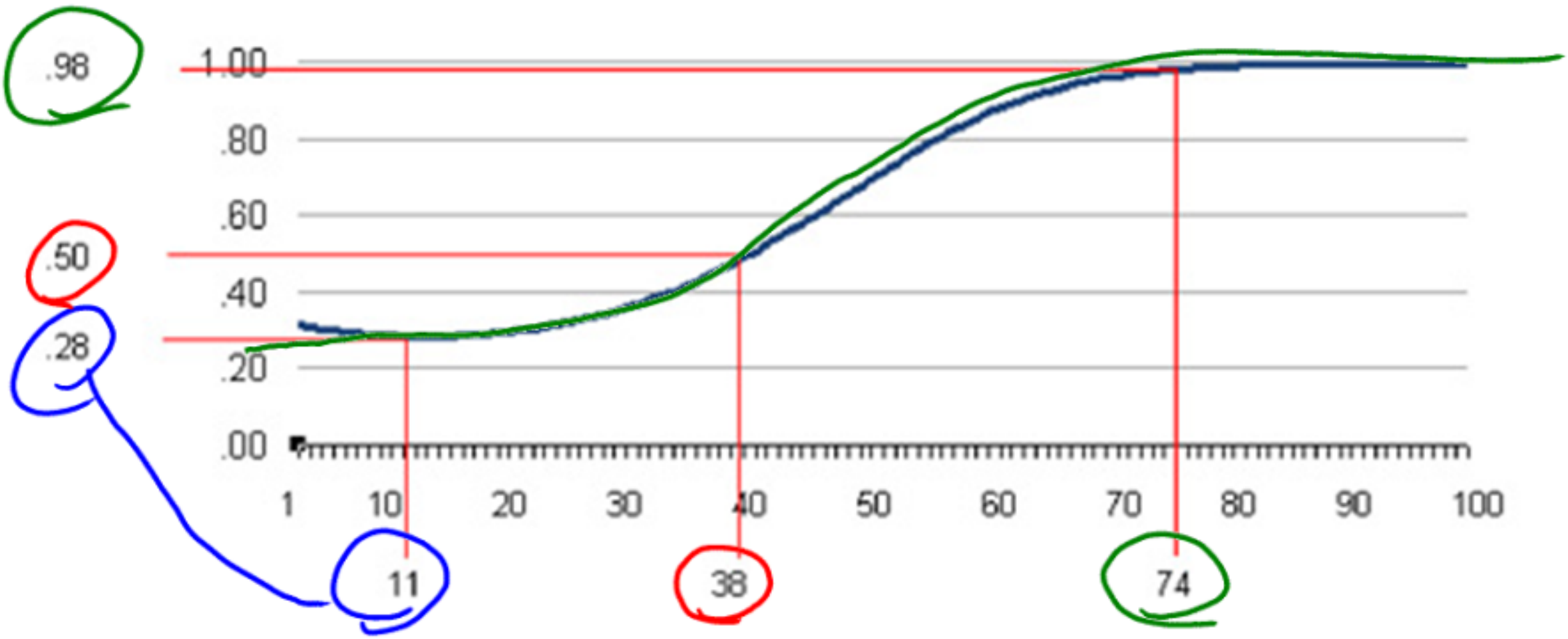


Lets look at some of your code from last lab…

Probability of Fault Versus Cyclomatic Complexity



Probability of Fault Versus Cyclomatic Complexity



Cyclomatic Complexity and Testability

Cyclomatic Complexity	Testability
1-10	A Simple program without much risk
11-20	More complex, moderate risk
21-50	Complex, High Risk
51+	Untestable, very high risk

Issues with Cyclomatic Complexity

- Are all decisions equal?
  - Case statement versus if statements?
  - Nested logic versus multiple conditions?