Path Testing (cont.)

Dr. John H Robb
UTA Computer Science and Engineering

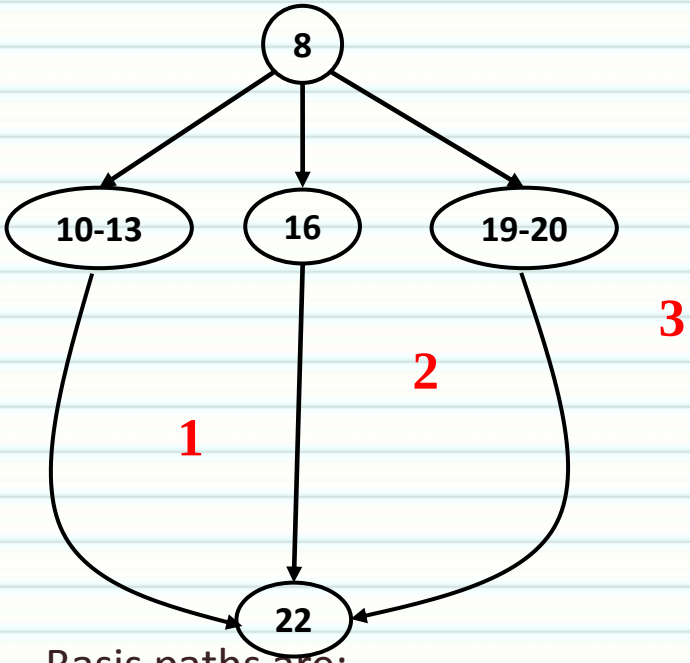
Switch Statement Exercise

```
1   private enum Day
2   {
3    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
4   }
5
6    public static void workingWeek(Day today)
7    {
8     switch (today)
9     {
10      case MONDAY:
11      case TUESDAY:
12      case WEDNESDAY:
13      case THURSDAY:  System.out.println("Workday");
14                  break;
15
16      case FRIDAY:    System.out.println(“Last workday!");
17                  break;
18
19      case SATURDAY:
20      case SUNDAY:    System.out.println("Weekend!");
21                  break;
22
23    }
24   }
```

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity**
5. **Develop the test cases using required input values to achieve boundary value coverage**
6. **Determine coverage achieved**
7. **Do the test cases and outputs refute or support the code functional description?**

Switch Statement Exercise (answer)

1. The code is writing the Strings “Workday” four times, followed by “Last Workday!” and then “Weekend!” twice – all on a successive line



4. Basis paths are:
   a) 8, 10-13, 22
   b) 8, 16, 22
   c) 8, 19-20, 22
5. Test cases are:
   a) theDay=Thursday, EO=“Workday”
   b) theDay=Friday, EO=“Last Workday!”
   c) theDay = Saturday, EO=“Weekend!”

Is this all I should do?

Switch Statement Example - Updated

```
1   private enum Day
2   {
3    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
4   }
5
6    public static void workingWeek(Day today)
7    {
8     switch (today)
9     {
10      case MONDAY:
11
12      case WEDNESDAY:
13      case THURSDAY:  System.out.println("Workday");
14                  break;
15
16      case FRIDAY:    System.out.println(“Last workday!");
17                  break;
18
19      case SATURDAY:
20      case SUNDAY:    System.out.println("Weekend!");
21                  break;
22
23    }
24   }
```

1. How are the basis paths and cyclomatic complexity changed for this example of code?
2. Are the test cases different? Do they expose the problem?
3. How should I test this?
4. If I am not careful I may never see that Tuesday is missed.

Switch Statement Weaknesses

- Mitre.org who are responsible for the National Security Engineering Center maintains a list of common code vulnerabilities (security related) call the Common Weakness Enumeration (CWE)

- CWE is co-sponsored by the office of Cybersecurity and Communications at the U.S. Department of Homeland Security.

- 2 CWE weaknesses associated with a switch statement
  - CWE-478: Missing Default Case in Switch Statement ...
  - CWE-484: Omitted Break Statement in Switch (2.8)

- We'll look at examples of the former – we know the latter is a weakness

Switch Default Example (cont.)

- From Mitre.org
- This code assumes that the value of the points input parameter will always be 0, 1 or 2 and does not check for other incorrect values passed to the method.
- This can be easily accomplished by providing a default label in the switch statement that outputs an error message indicating an invalid value for the points input parameter and returning a null value.
- The following slide shows the updated switch

Switch Default Example

- Adopted from Mitre.org

```
public static double getInterestRate(int points) {
        double result;
        switch (points) {
                case 0:
                    result = 0.05;
                    break;
                case 1:
                    result = 0.0475;
                    break;
                case 2:
                    result = 0.045;
                    break;
                default:
                    System.err.println("Invalid value for points, must be 0, 1 or 2");
                    System.err.println("Returning null value for interest rate");
                    result = 0.0;
        }
return result;}
```

What assumptions are we making about the default?

Switch Statement Example - Updated

```
1   public class WorkingDayClass {

    public enum Day
    {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}

     public static void workingWeek(Day theDay)
     {
       switch (theDay)
       {
         case MONDAY:

         case WEDNESDAY:
         case THURSDAY:  System.out.println(theDay+" is a workday");
                         break;
         case FRIDAY:    System.out.println(theDay+" is the last Workday!");
                         break;
         case SATURDAY:
         case SUNDAY:    System.out.println(theDay+" is on the Weekend!");
//                       break;
         default:                System.out.println(theDay+" None of the above");
       }
     }
                 }
}
```

1. How does the default here affect the execution of this?
2.What do we expect to see if we execute this method?

Test Strategy for Switch Statement

- Determine if the Switch contains a default with n unique cases.

- If a default exists then
  1. The number of test inputs = n+1 which includes the default
  2. Determine the software response especially to the default – does this provide the correct response?
  3. Do the requirements address what the default does?

- If no default
  1. Determine the correct strategy for the requirements/software if the default is executed – update as rqmts/software accordingly
  2. Try number of test cases = n+1 and try to go out of range of the switch – may not be possible for enums

- Test cases vs. test cases + inspection
  - From our previous example we can look at the switch statement
  - We are looking for cases in the wrong spot and can visually spot those
  - Spotting an error of omission is harder (e.g., Tuesday missing)
  - Best strategy is to test all cases, but when expensive then test all unique cases and inspect each equivalence

Code Coverage Criteria and Structured Code

- Code coverage leads to the sometimes controversial statement to avoid using these constructs in code
  - Goto (never use)
  - Break (except in switches)
  - Continue
  - Return (only use it once in a single method)

- When we avoid using these constructs
  - The code has only one entry and one exit and is also easier to draw the CFG and test
  - Like other engineering disciplines we are developing a product where testability is a central consideration

Examples

```
public static boolean verify(int param1) {
   if (param1 < 0) return false;
   if (param1 > 31) return false;
   if ((param1 <30) && (param1>15)) return false;
   if (param1 == 15) return false;
   return true;
}
```

Try to draw the CFG for this as an exercise

```
while (loop_control) {
     if (loop_count > 1000) break;
     if (time_exect > 3600) break;
     if (this.data == "undefined") continue;
     if (this.skip == true) continue; ... }
```

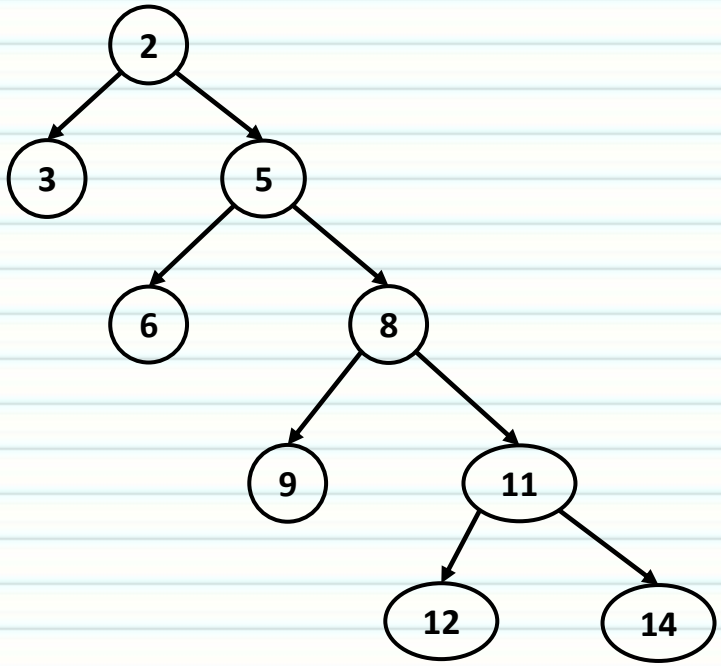
Try to draw the CFG for this as an exercise

- Most mature software organizations have programming standards that either disallow or highly restrict these constructs

Student Exercise

Make the following spaghetti code structured and develop the CFG. Assume temp is an integer and a range of 0 to 100 inclusive.

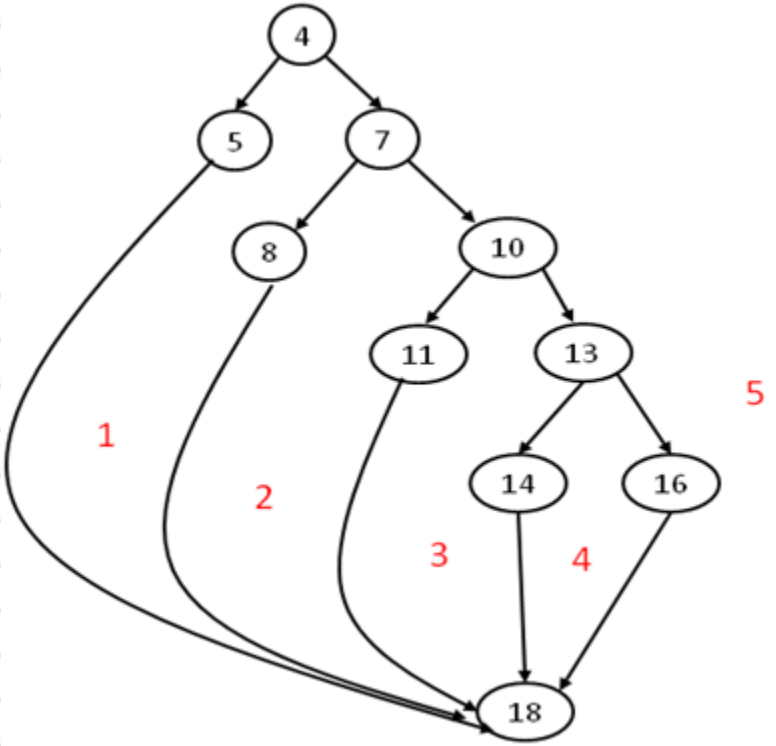
```
1   String getWeight(int temp) {
2         if (temp<= 32) {
3               return "freezing";
4         }
5         if (temp < 40) {
6               return "very cold";
7         }
8         if (temp < 50) {
9               return "cold";
10        }
11        if (temp < 80) {
12            return "nice";
13        }
14        return "warm";
15  }
```



Answer

The re-written code is as follows:

```
1   public String getTemp(int temp) {
2       String feels;
3
4       if (temp<=32)
5           feels="Freezing";
6       else {
7           if (temp<40)
8               feels="Very Cold";
9           else {
10              if (temp<50)
11                  feels="Cold";
12              else {
13                  if (temp<80)
14                      feels="Nice";
15                  else
16                      feels="Warm";
17              }}}
18      return feels;
19  }
```



Develop the Test Cases needed to achieve Statement coverage

Answer (cont.)

The basis paths are as follows:

Test Case	Inputs temp	Exp Output Return or feels	Basis Path coverage
1	32	"Freezing"	4, 5, 18
2	39	"Very Cold"	4, 7, 8, 18
3	49	"Cold"	4, 7, 10, 11, 18
4	79	"Nice"	4, 7, 10, 13, 14, 18
5	80	"Warm"	4, 7, 10, 13, 16, 18

Three BVs are left uncovered:

Boundary Value (x)	Fully Tested?	Condition missing
32	No	>32
40	No	=40
50	No	=50
80	Yes	n/a

Test Case	Inputs temp	Exp Output Return or feels	Basis Path coverage
6	33	"Very Cold"	n/a
7	40	"Cold"	n/a
8	50	"Nice"	n/a

ECPs look like the following (with the untested values in red)

temp	0	32	33	39	40	49	50	79	80	100

The additional tests required then are the following:

Test Case	Inputs temp	Exp Output Return or feels	Basis Path coverage
9	0	"Freezing"	n/a
10	100	"Warm"	n/a

Full decision, statement, ECP, and boundary value coverage is achieved with these 10 tests.

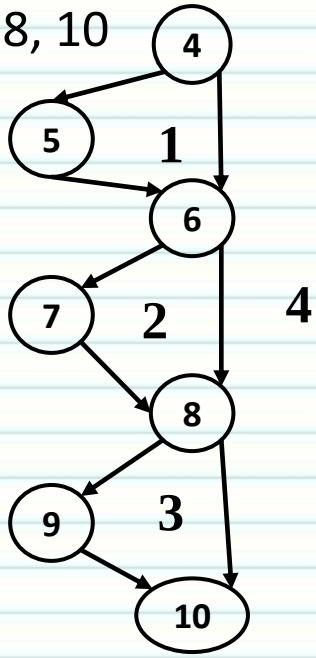
Demonstrate Code Based Coverage

```
1   public static int returnInput(boolean conditiona, boolean conditionb, boolean conditionc) {
2           int x=0;
3
4           if (conditiona)
5            x++;
6           if (conditionb)
7            x++;
8           if (conditionc)
9            x++;
10
11          return x;
12  }
```

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity**

Demonstrate Code Based Coverage (cont.)

1. The code is counting the number of the three input conditions that have the boolean value true
4. Basis paths are:
   a) 4, 5, 6, 7, 8, 9, 10
   b) 4, 6, 7, 8, 9, 10
   c) 4, 6, 8, 9, 10
   d) 4, 6, 8, 10



4

Coverage goal	Test Inputs	Number of combinations	What's measured?
Statement	TTT	1	Each node is reached
Branch	TTT,FFF	2	Each edge is reached
Decision	TTT,FFF	2	Each edge is reached
Condition	TTT,FFF	2	Decisions and Conditions are identical here
Basis Path	TTT,FTT,FFT,FFF	4	Each edge is reached by changing one decision at a time
Path	Truth table for 3 inputs	8	All possible paths

Note:

100 decision coverage implies 100% statement coverage.

100 % statement coverage does not imply 100 % decision coverage

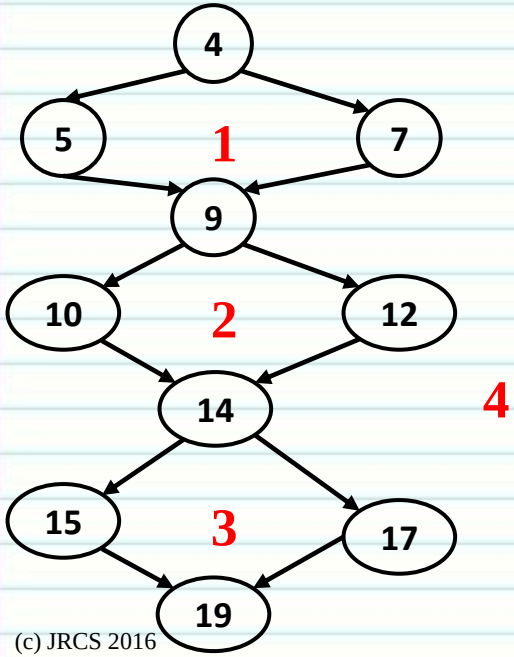
Another Example

```
1   public static int returnInput(boolean conditiona, boolean conditionb, boolean conditionc) {
2           int x=0;
3
4           if (conditiona)
5            x++;
6           else
7            x--;
8
9           if (conditionb)
10           x++;
11          else
12           x--;
13
14          if (conditionc)
15           x++;
16          else
17           x--;
18
19          return x;}
```

1. **Describe what the code seems to be doing**
2. **Draw the reduced CFG - use your previous CFG**
3. **Determine the Cyclomatic complexity**
4. **Develop the basis paths using the Cyclomatic complexity**
5. **Develop the coverage table for the different coverage levels**

Demonstrate Code Based Coverage (cont.)

1. The code is determining the sum of the number of True inputs minus the sum of the number of False inputs
4. Basis paths are:
   a) 4, 5, 9, 10, 14, 15, 19
   b) 4, 7, 9, 10, 14, 15, 19
   c) 4, 7, 9, 12, 14, 15, 19
   d) 4, 7, 9, 12, 14, 17, 19



Coverage goal	Test Inputs	Number of combinations	What's measured?
Statement	TTT, FFF	2	Each node is reached
Branch	TTT,FFF	2	Each edge is reached
Decision	TTT,FFF	2	Each edge is reached
Condition	TTT,FFF	2	Decisions and Conditions are identical here
Basis Path	TTT,FTT,FFT,FFF	4	Each edge is reached by changing one decision at a time
Path	Truth table for 3 inputs	8	All possible paths

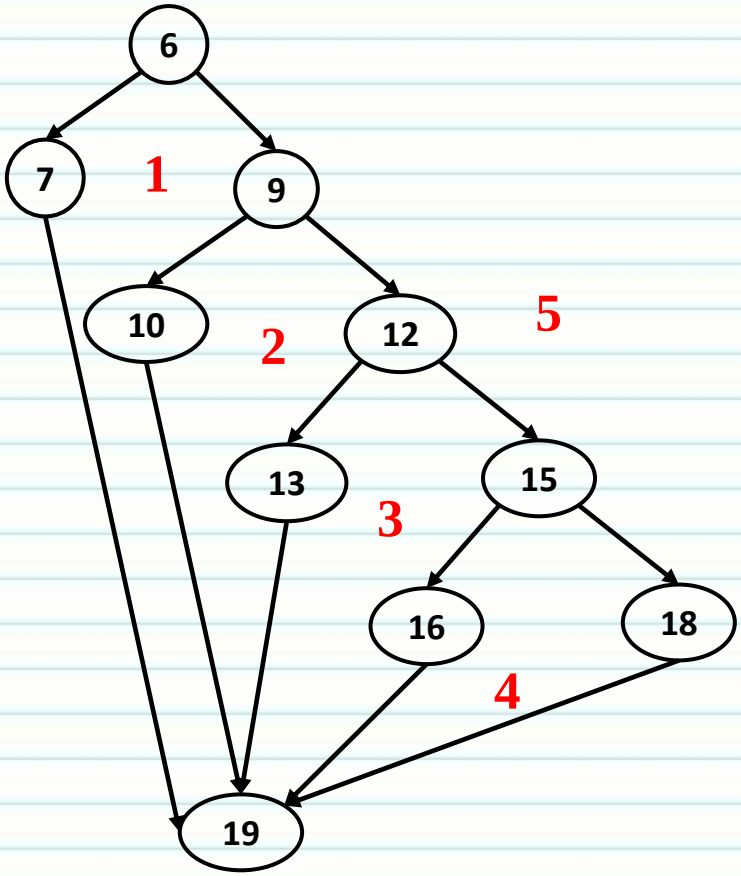
Do you see the difference between Statement and Decision coverage with these examples?

Mathematical Exercise

For the following snippet of code, utilize the in-class steps to analyze the problem. Assume a significance of 0.1 for this problem and a range of -4.0 to 10.0 inclusive for x.

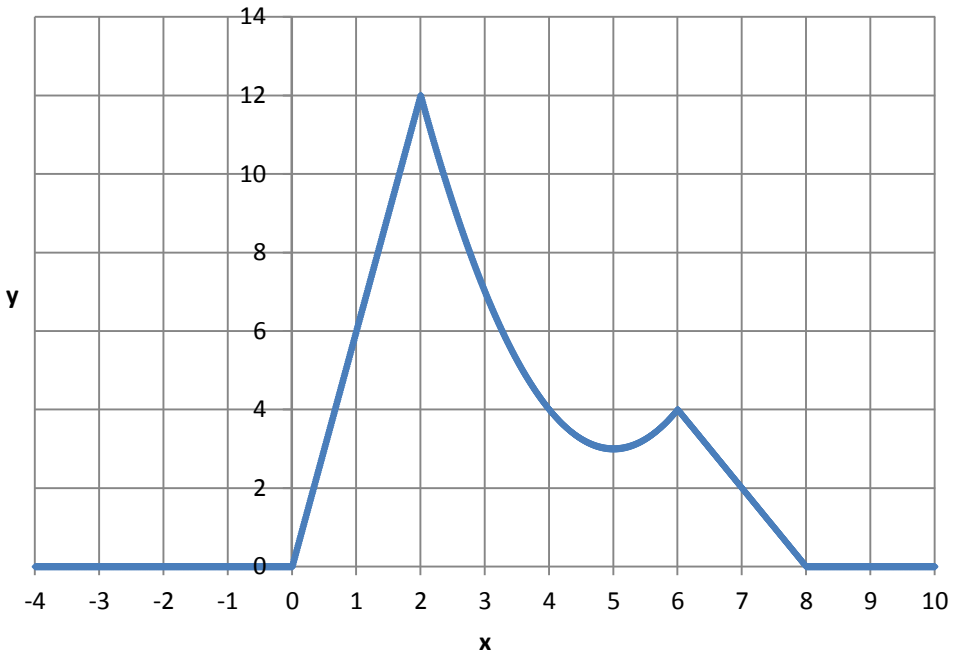
```
4  public double computeAnswer(double x) {
5    double y;
6    if (x < 0.0)
7      y = 0.0;
8    else
9      if (x < 2.0)
10         y = 6.0*x;
11     else
12         if (x > 8.0)
13           y = 0.0;
14         else
15           if (x > 6.0)
16             y = -2.0 * x + 16.0;
17           else
18             y=(x - 5.0) * (x - 5.0) + 3.0;
19 return y;
20 }
```

Answer



Answer (cont.)

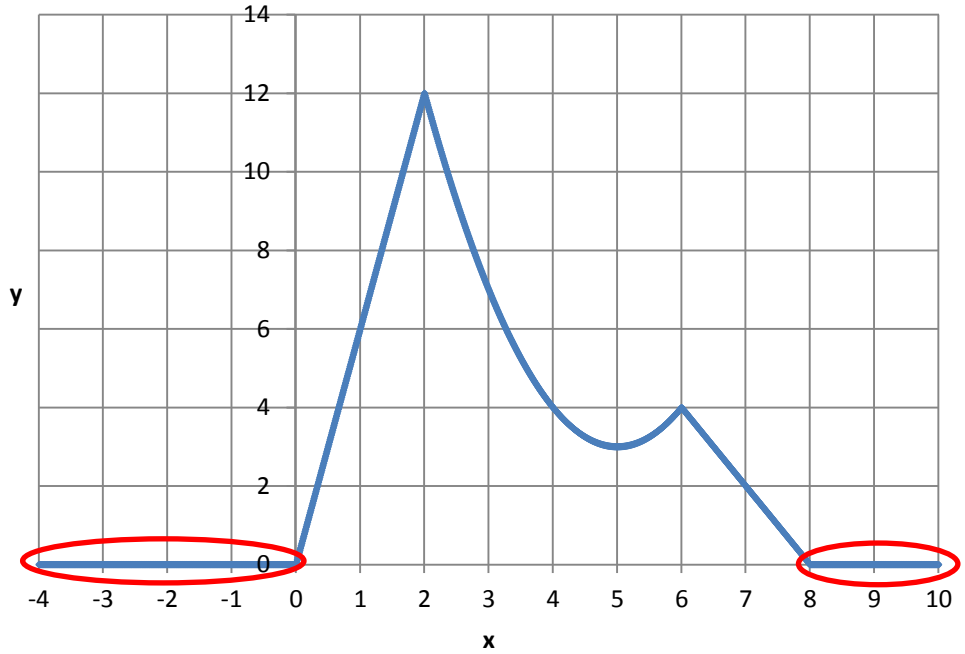
- Here is what the function looks like when graphed



- What problems are we going to have using our ECP/BVA approach here?

Answer (cont.)

- This mathematical function brings out an assumption that we have not specifically talked about - and this is the sole reason for the mathematical exercise



Up to this point we have implicitly talked about ECPs being uniform (a constant value) across the region

1. **interest rate = 5.0%**
2. **redLight = On**

We have used two points on each ECP (and since it's a constant we are safe doing this)

Since these other regions are non constant we need to add test points to ensure the equations are correct:

1. **linear functions - add one test point in the middle**
2. **parabolic - add two test points 1) at the max/min 2) midway between the max/min and the BV of the ECP (in terms of the x-value)**

Answer (cont.)

- Here are the basis paths

Test Case	Inputs x	Exp Output Return or y	Basis Path coverage
1	-0.1	0.0	6-7-19
2	1.9	11.4	6-9-10-19
3	8.1	0.0	6-9-12-13-19
4	6.1	3.8	6-9-12-15-16-19
5	6.0	4.0	6-9-12-15-18-19

- Which BVs are left unmapped?

Boundary Value (x)	Fully Tested?	Condition missing
0.0	No	=0.0
2.0	No	=2.0
6.0	Yes	n/a
8.0	No	=8.0

Test Case	Inputs x	Exp Output Return or y	Basis Path coverage
6	0.0	0.0	-
7	2.0	12.0	-
8	8.0	0.0	-

Answer (cont.)

- When we examine the ECPs we see that all ECPs are covered (two tests per ECP) except the two outside ones

x	-4.0 -0.1	0.0 1.9	2.0 6.0	6.1 8.0	8.1 10.0

- So, we add these tests as well:

Test Case	Inputs x	Exp Output Return or y	Basis Path coverage
9	-4.0	0.0	n/a
10	10.0	0.0	n/a

Answer (cont.)

- But there's more! We need to cover the regions that don't have uniform actions - we'll add test cases to correspond to the degree of the non-uniform function.

Test Case	Inputs x	Exp Output Return or y	Basis Path coverage
11	1.0	6.0	-
12	5.0	3.0	-
13	3.5	5.3	-
14	7.0	2.0	-
	Value could also be x=5.5 and y=3.3		

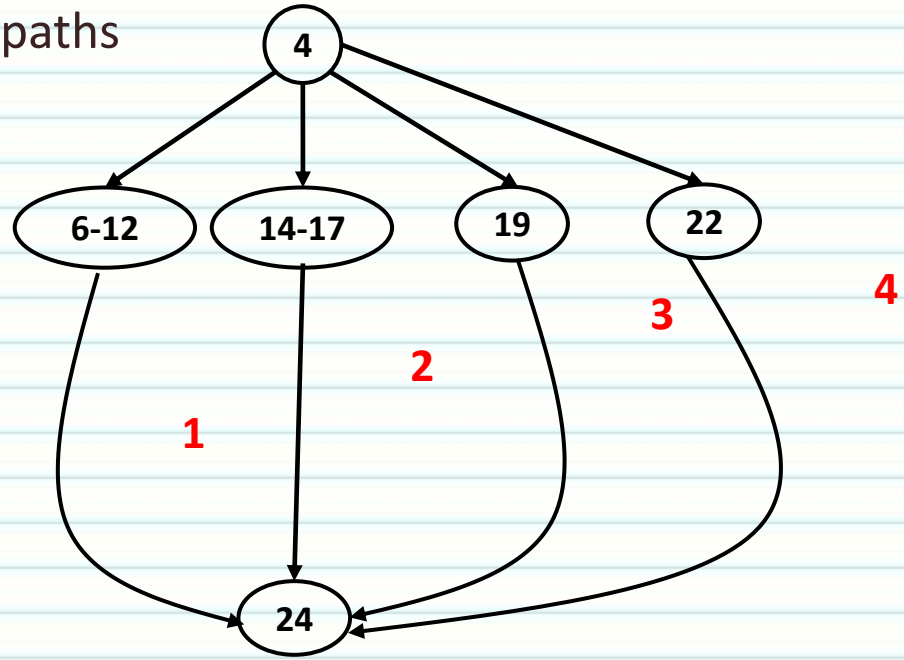
Switch Exercise

```
1   public static int daysinmonth(Month monthname) {
2           int x;
3
4           switch (monthname) {
5
6           case January:
7           case March:
8           case May:
9           case July:
10          case August:
11          case October:
12          case December: x=31; break;
13
14          case April:
15          case June:
16          case September:
17          case November: x=30; break;
18
19          case February: x=28; break;
20
21          //Just to cover the case where the enum is not captured in the case above
22          default: x=0; System.out.println("Error");
23          }
24          return x; }
```

Develop the CFG and ID the basis paths

Answer

- Here is the CFG and the basis paths

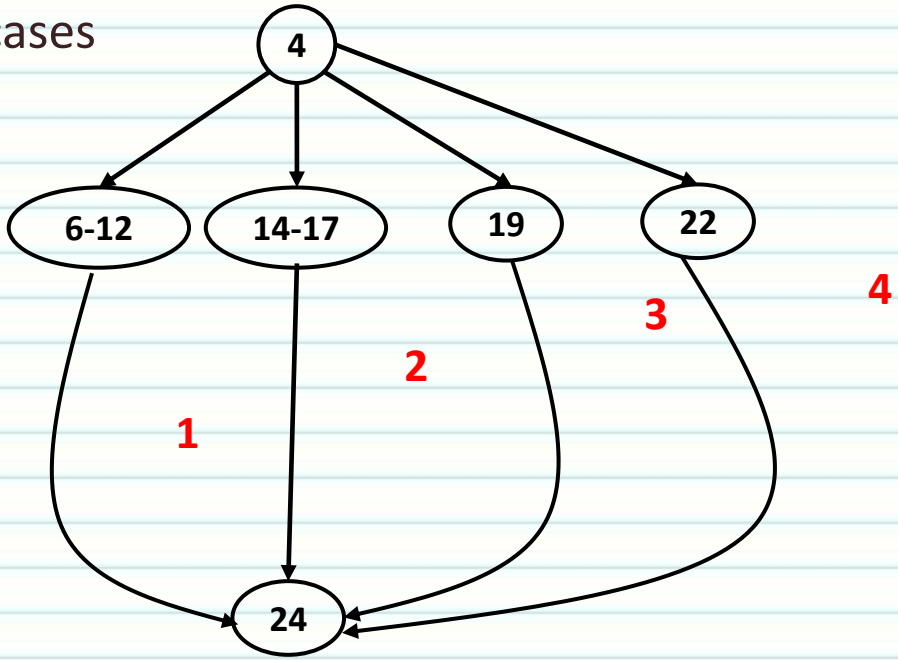


Test Case	Inputs Monthname	Exp Output Return or x	Basis Path coverage
1	January, March, May, July, August, October, or December	31	4, 6-12, 24
2	April, June, September, or November	30	4, 14-17, 24
3	February	28	4, 19, 24
4	n/a (not possible to test)	n/a	4, 22, 24

- What is missing?

Answer

- We have not fully verified all cases



Test Case	Inputs Monthname	Exp Output Return or x	Basis Path coverage
5	March (assuming January was chosen above)	31	4, 6-12, 24
6	May	31	4, 6-12, 24
7	July	31	4, 6-12, 24
8	August	31	4, 6-12, 24
9	October	31	4, 6-12, 24
10	December	31	4, 6-12, 24
11	June (assuming that April was chosen above)	30	4, 14-17, 24
12	September	30	4, 14-17, 24
13	November	30	4, 14-17, 24

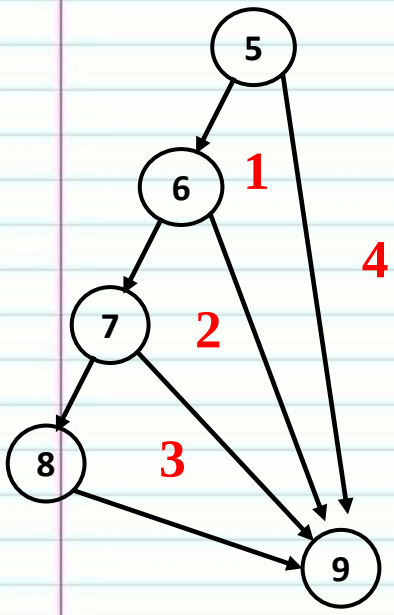
Multiple Variables with Boundary Conditions

```
1    public double applyDiscount (double balance, boolean prime,
                                  int years_prime, double discount) {
2
3    double invoice;
4    invoice=balance;
5    if (balance>3000.00)
6        if (prime)
7            if (years_prime>5)
8                invoice=(1-discount)*balance;
9    return invoice;
10   }
```

Assume:

1. The parameter balance has a range of 0 to 10,000.00 inclusive.
2. Years Prime has a range of 0 100 inclusive.
3. Assume Discount has a range of 0.0 to 1.0 inclusive - 0.15 means a 15 % percent discount is applied. Use this value
4. All currency values are truncated to and significant to the cent.

Multiple Variables with Boundary Conditions (cont.)



Basis path tests are:

Test Case	Inputs				Exp Out Result	Basis Path
	balance	prime	years_prime	discount		
1	$3,000.01	T	6	0.15	$2,550.00	5-6-7-8-9
2	$3,000.00	T	6	0.15	$3,000.00	5-9
3	$3,000.01	F	6	0.15	$3,000.01	5-6-9
4	$3,000.01	T	5	0.15	$3,000.01	5-6-7-9

Boundary values/extreme ranges tested are:

balance ($)	0.0 3000.00	3000.01 10,000.00
prime	0 5	6 100

So we need to add in test for red BVs above

Multiple Variables with Boundary Conditions (cont.)

```
1    public double applyDiscount (double balance, boolean prime,
                                  int years_prime, double discount) {
2
3    double invoice;
4    invoice=balance;
5    if (balance>3000.00)
6        if (prime)
7            if (years_prime>5)
8                invoice=(1-discount)*balance;
9    return invoice;
10   }
```

Mentally convert to

```
return (balance>3000.00&&prime&&year_prime>5) ? (1-discount)*balance:balance;
```

This is of the form a&&b&&c

MCDC statements 5-8	balance	prime	years_prime	discount	Result
TTT	$3,000.01	T	6	0.15	$2,550.00
FTT	$3,000.00	T	6	0.15	$3,000.00
TFT	$3,000.01	F	6	0.15	$3,000.01
TTF	$3,000.01	T	5	0.15	$3,000.01

These tests are test cases 1-4 in the previous test case table - so we don't need to add any MCDC tests

Multiple Variables with Boundary Conditions (cont.)

```
1   public double applyDiscount (double balance, boolean prime,
                                 int years_prime, double discount) {
2
3   double invoice;
4   invoice=balance;
5   if (balance>3000.00)
6      if (prime)
7         if (years_prime>5)
8            invoice=(1-discount)*balance;
9   return invoice;
10  }
```

Test Case	Inputs				Exp Out Result	Basis Path	MCDC
	balance	prime	years_prime	discount			
1	$3,000.01	T	6	0.15	$2,550.00	5-6-7-8-9	stmt 5-8 TTT
2	$3,000.00	T	6	0.15	$3,000.00	5-9	stmt 5-8 FTT
3	$3,000.01	F	6	0.15	$3,000.01	5-6-9	stmt 5-8 TFT
4	$3,000.01	T	5	0.15	$3,000.01	5-6-7-9	stmt 5-8 TTF
5	$0.00	T	5	0.15	$0.00	-	-
6	$10,000.00	T	5	0.15	$10,000.00	-	-
7	$3,000.01	T	0	0.15	$3,000.01	-	-
8	$3,000.01	T	100	0.15	$2,550.00	-	-

- value is a don't care (any value can be used)