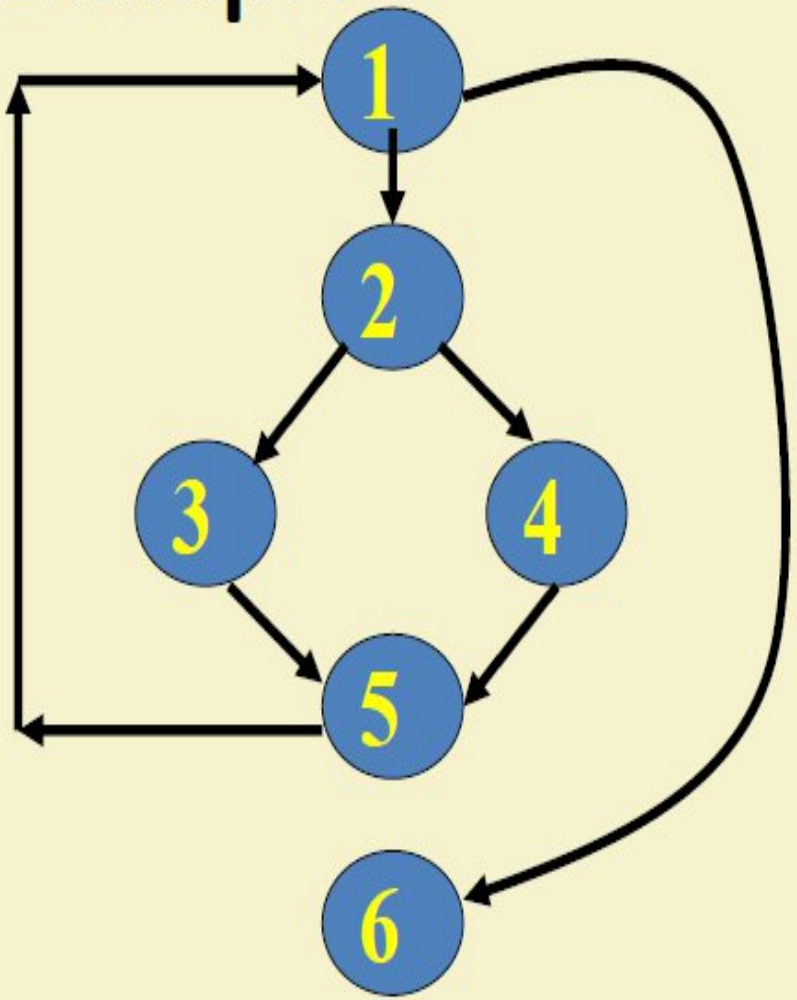
Cyclomatic Complexity

How to Draw Control Flow Graph?

- **Number all statements of a program.**
- Numbered statements:
  - Represent nodes of control flow graph.
- Draw an edge from one node to another node:
  - **If execution of the statement representing the first node can result in transfer of control to the other node.**

```
int f1(int x,int y){
1 while (x != y){
2    if (x>y) then
3        x=x-y;
4    else y=y-x;
5  }
6 return x;       }
```

Example

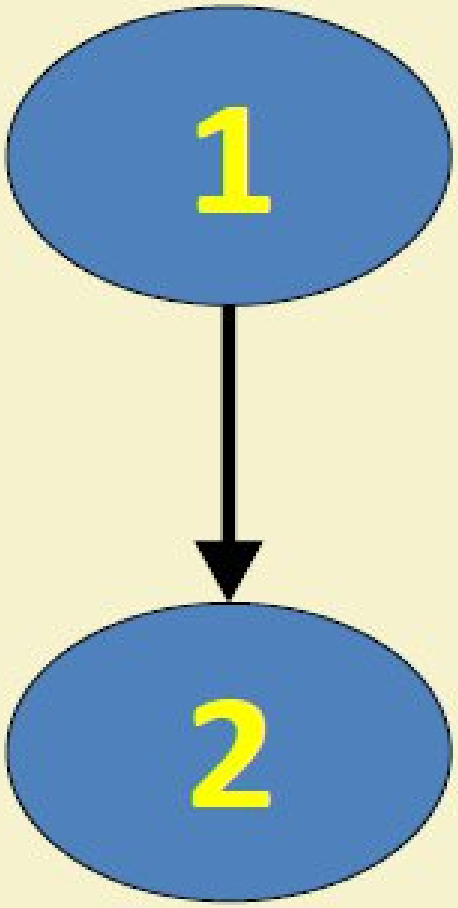


How to Draw Control Flow Graph?

- Every program is composed of:
  - **Sequence**
  - **Selection**
  - **Iteration**
- If we know how to draw CFG corresponding these basic statements:
  - We can draw CFG for any program.

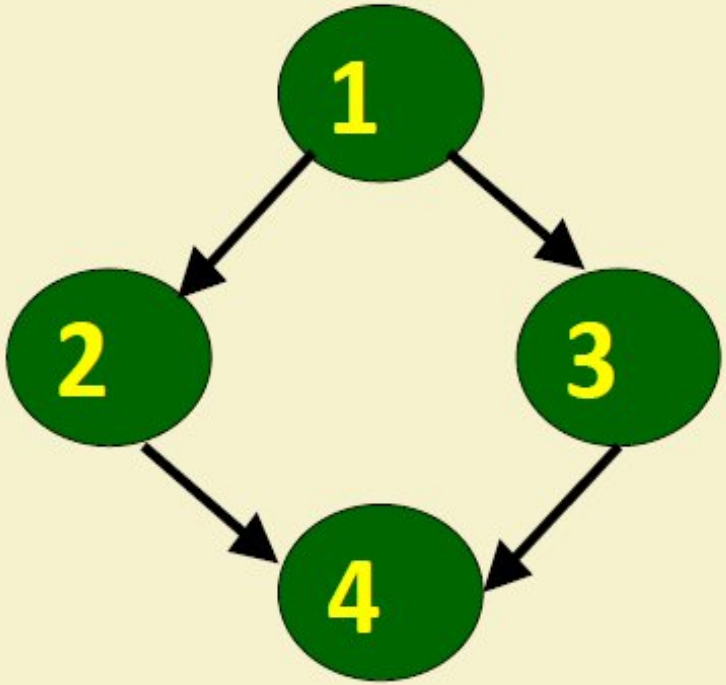
How to Draw Control Flow Graph?

- **Sequence:**
  - 1 a=5;
  - 2 b=a*b-1;



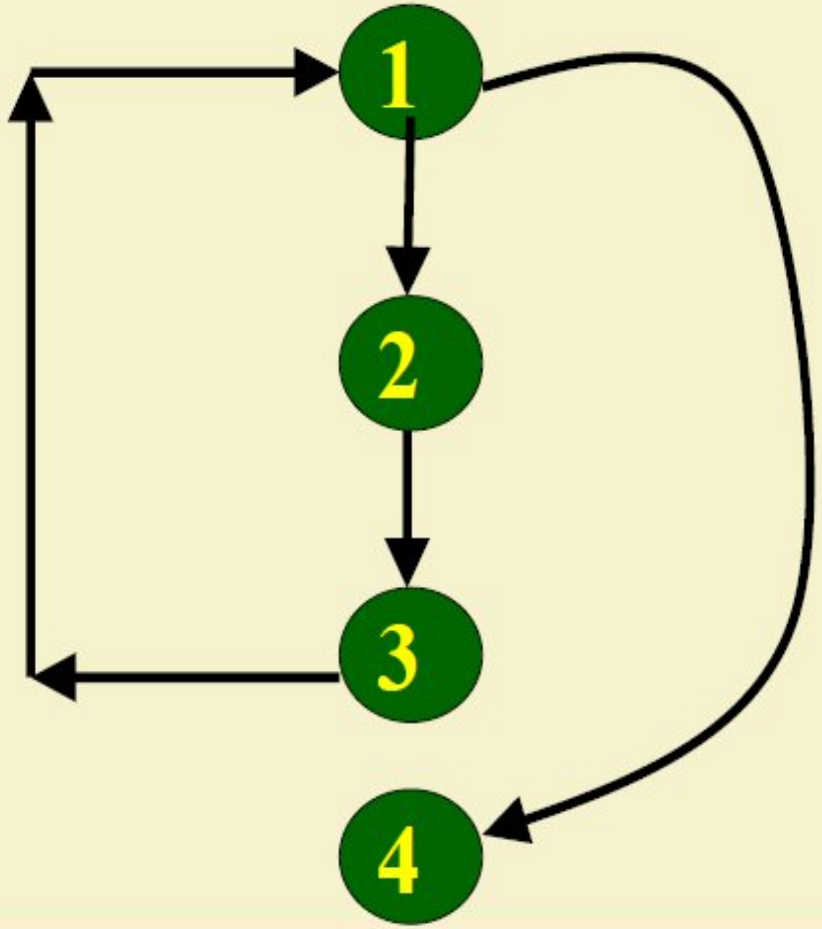
How to Draw Control Flow Graph?

- **Selection:**
  - 1 if(a>b) then
  - 2 c=3;
  - 3 else c=5;
  - 4 c=c*c;



How to Draw Control Flow Graph?

- Iteration:
  - 1 while(a>b){
  - 2 b=b*a;
  - 3 b=b-1;}
  - 4 c=b+d;



Path

- A path through a program:

–A node and edge sequence from the starting node to a terminal node of the control flow graph.

–There may be several terminal nodes for program.

All Path Criterion

- In the presence of loops, the number paths can become extremely large:

–This makes all path testing impractical

Linearly Independent Path

- Any path through the program that:

–Introduces at least one new edge:

•Not included in any other independent paths.

- It is straight forward:

–To identify linearly independent paths of simple programs.

- For complicated programs:

–It is not easy to determine the number of independent paths.

McCabe's CyclomaticMetric

- An upper bound:

–For the number of linearly independent paths of a program

- Provides a practical way of determining:

–The maximum number of test cases required for basis path testing.

McCabe's CyclomaticMetric

- Given a control flow graph G, cyclomatic complexity V(G):
  - **V(G)= E-N+2**
- N is the number of nodes in G
- E is the number of edges in G

- Cyclomatic complexity *V*(*G*) for a flow graph *G* is also defined as ***V*(*G*)=*P*+1**
  - where *P* is the number of predicate nodes contained in the flow graph *G*.

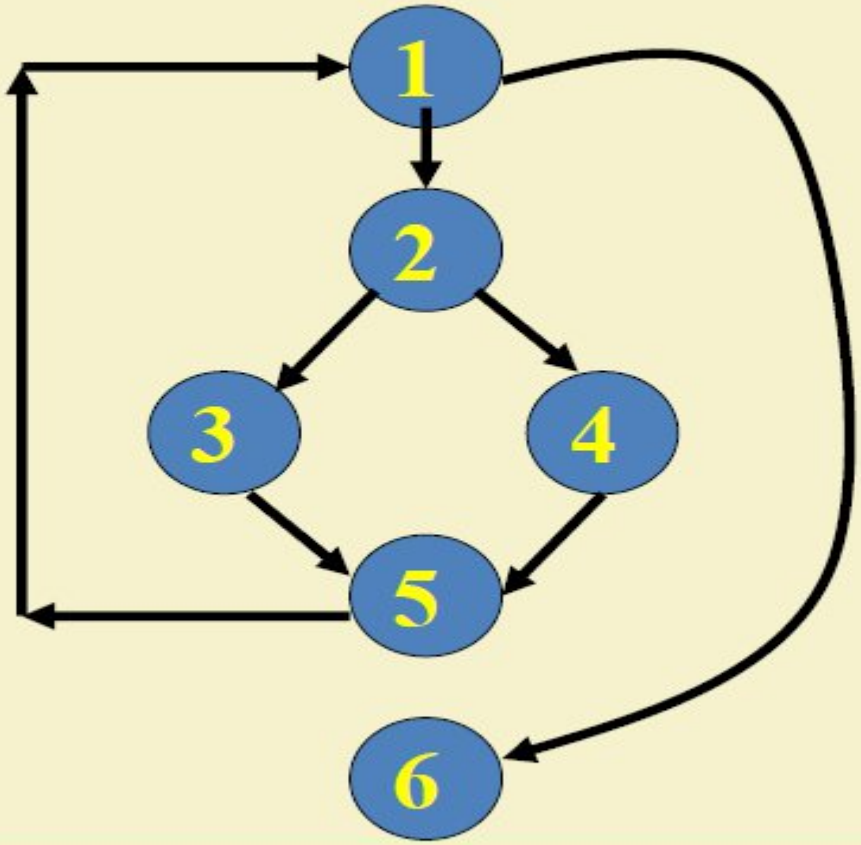
CyclomaticComplexity

- Another way of computing cyclomatic complexity:
  - inspect control flow graph
  - determine number of bounded areas in the graph
- V(G) = Total number of bounded areas + 1
  - Any region enclosed by a nodes and edge sequence.

OR

The **number of regions** of the flow graph corresponds to the cyclomatic complexity.

Example Control Flow Graph



Cyclomatic complexity = 7-6+2 = 3.

Cyclomatic Complexity

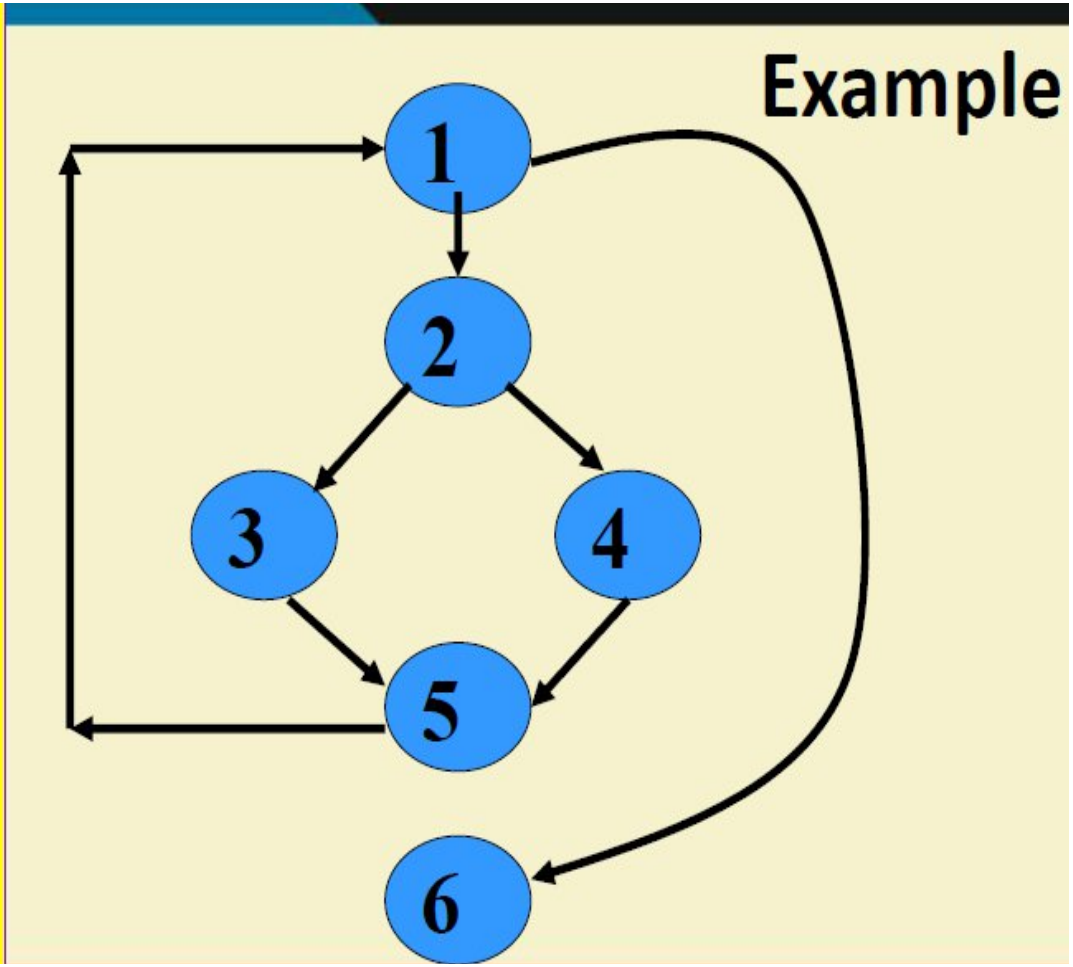
- McCabe's metric provides:
- **A quantitative measure of testing difficulty and the reliability**
- Intuitively,

–Number of bounded areas increases with the number of decision nodes and loops.

Derivation of Test Cases

- Draw control flow graph.
- Determine V(G).
- Determine the set of linearly independent paths.
- Prepare test cases:
  - Force execution along each path.
  - Not practical for larger programs.

```
int f1(int x,int y){
1 while (x != y){
2    if (x>y) then
3        x=x-y;
4    else y=y-x;
5  }
6 return x;       }
```



Derivation of Test Cases

- **Number of independent paths: 3**
  - **1,6 test case (x=1, y=1)**
  - **1,2,3,5,1,6 test case(x=2, y=1)**
  - **1,2,4,5,1,6 test case(x=1, y=2)**

An Interesting Application of Cyclomatic Complexity

- Relationship exists between:
  - McCabe's metric
  - The number of errors existing in the code,
  - Time required to correct the errors.
  - Time required to understand the program

Cyclomatic Complexity

- Cyclomatic complexity of a program:

–**Indicates the psychological complexity of a program.**

–Difficulty level of understanding the program.

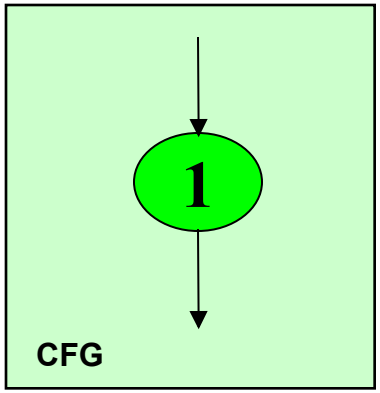
Cyclomatic Complexity

- From maintenance perspective,
  - Limit cyclomatic complexity of modules
    - To some reasonable value.
  - Good software development organizations:
    - Restrict cyclomatic complexity of functions to a maximum of ten or so.

Simple Examples

```
Statement1;
Statement2;
Statement3;
Statement4;
```

Can be represented as **one** node as there is no branch.



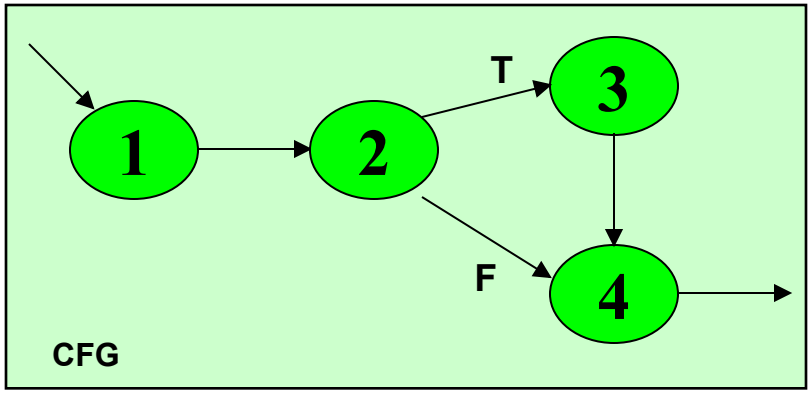
```
Statement1;
Statement2;

if X < 10 then
   Statement3;

Statement4;
```

1

2

3

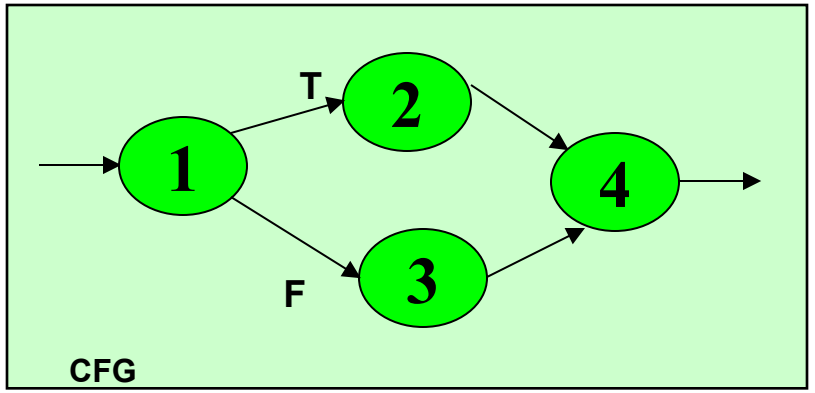
4



More Examples

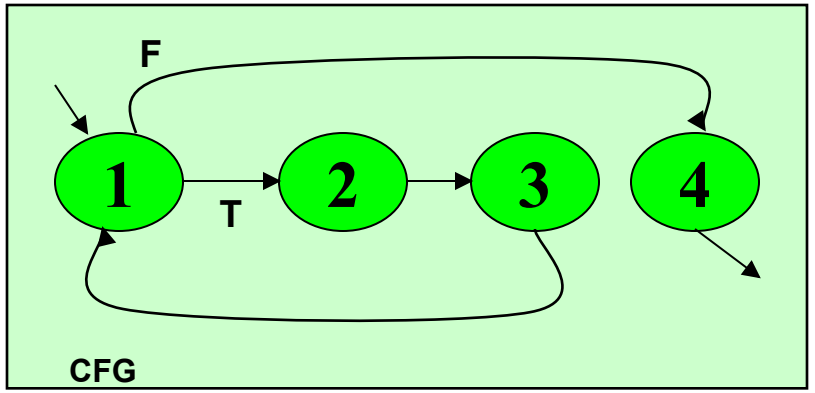
```
if X > 0 then
   Statement1;
else
   Statement2;
```

1
2
3



```
while X < 10 {
   Statement1;
   X++; }
```

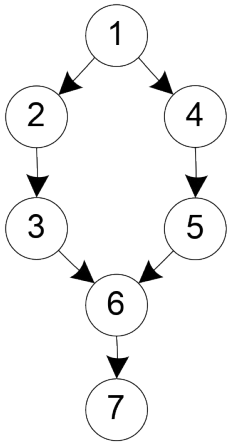
1
2
3



Program Graphs of Structured Programming Constructs

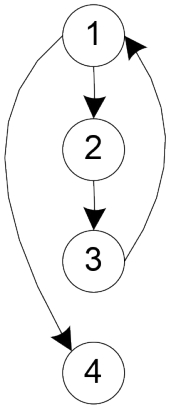
If-Then-Else

```
1   If <condition>
2      Then
3         <then statements>
4      Else
5         <else statements>
6   End If
7   <next statement>
```



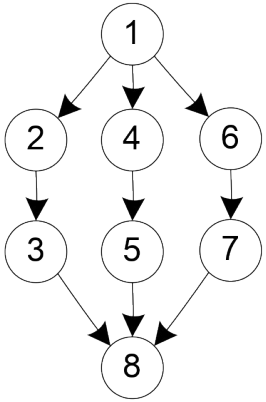
Pre-test Loop

```
1   While <condition>
2      <repeated body>
3   End While
4   <next statement>
```



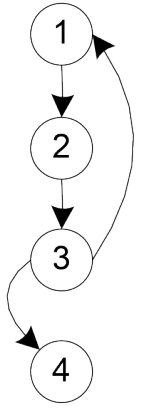
Case/Switch

```
1   Case  n Of 3
2      n=1:
3         <case 1  statements>
4      n=2:
5         <case 2  statements>
6      n=3:
7         <case 3  statements>
8   End Case
```



Post-test Loop

```
1   Do
2      <repeated body>
3   Until <condition>
4   <next statement>
```

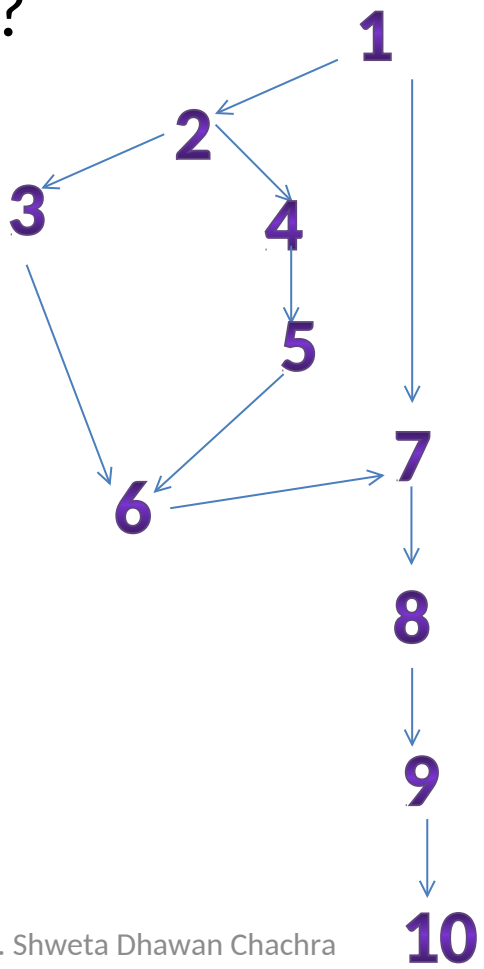


What is the McCabe's Cyclomatic Complexity for the following code segment?

```
IF A = 10 THEN
 IF B > C THEN
   A = B
 ELSE
   A = C
 ENDIF
ENDIF
Print A
Print B
Print C
```

What is the McCabe's Cyclomatic Complexity for the following code segment?

```
1  IF A = 10 THEN
2    IF B > C THEN
3      A = B
     ELSE
4      A = C
5    ENDIF
6  ENDIF
7  Print A
8  Print B
9  Print C
10
```



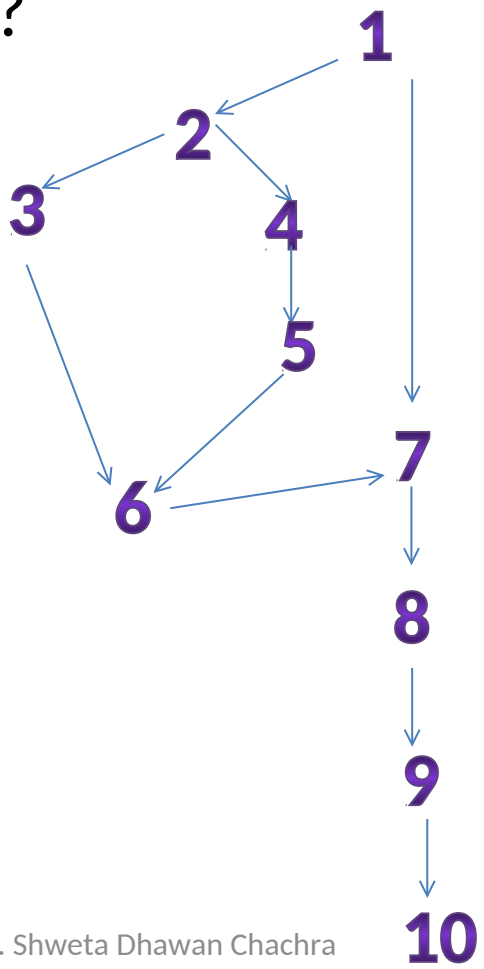
V(G)=NO OF BOUNDED AREAS + 1
=2+1
=3

V(G)=E-N+2
=11-10+2
=1+2
=3

V(G)=P+1
=2+1
=3

What is the McCabe's Cyclomatic Complexity for the following code segment?

```
1  IF A = 10 THEN
2    IF B > C THEN
       A = B
3    ELSE
4      A = C
5    ENDIF
6  ENDIF
7  Print A
8  Print C
9
10
```



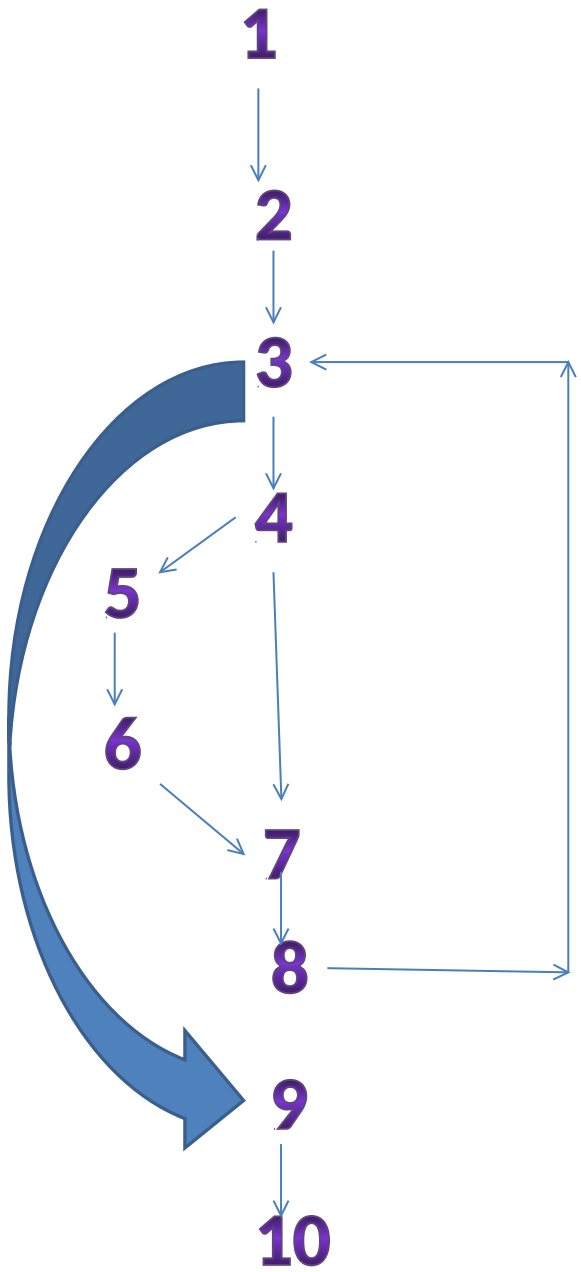
3 INDEPENDENT PATHS

PATH1=1,2,3,6,7,8,9,10
PATH2=1,2,4,5,6,7,8,9,10
PATH3=1,7,8,9,10

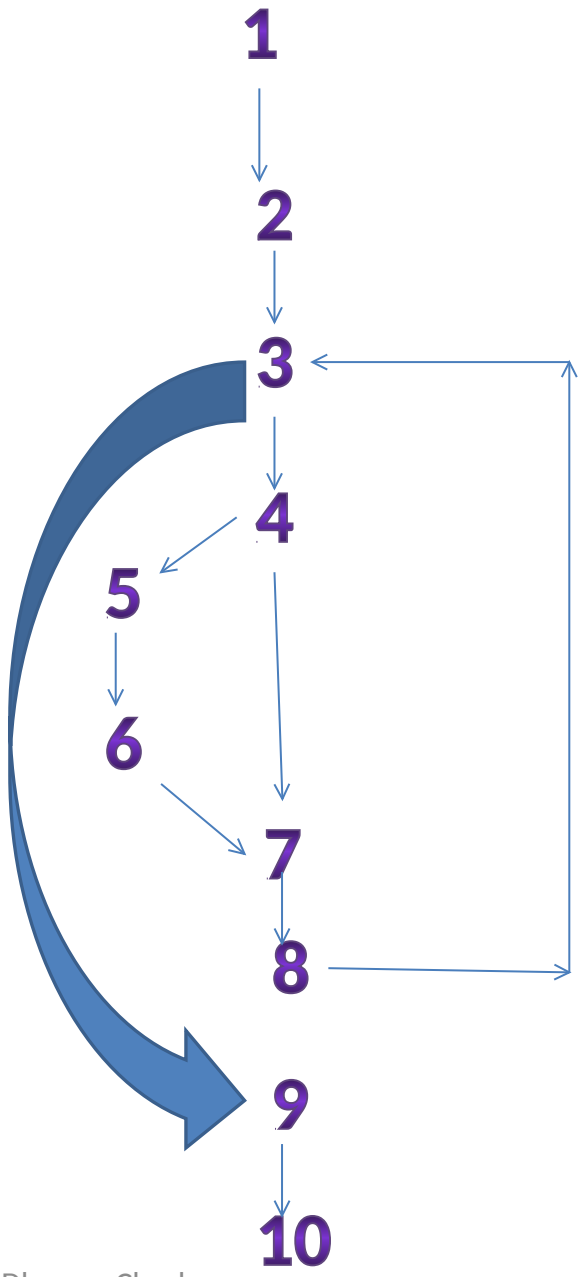
What is the McCabe's Cyclomatic complexity for the following code segment?

```
int partition (int arr[], int l, int h){
    int x = arr[h], i = (l - 1),t;

    for (int j = l; j <= h- 1; j++) {
        if (arr[j] <= x){
            i++;
           t=&arr[i]; &arr[i]= &arr[j]); &arr[j]=t;
        }
    }
    t=&arr[i + 1];  &arr[i+1])= &arr[h]; &arr[h]=t;
}
```

```
1  int partition (int arr[], int l, int h){
2   int x = arr[h], i = (l - 1),t;

3   for (int j = l; j <= h- 1; j++) {
4     if (arr[j] <= x){
5       i++;
6       t=&arr[i]; &arr[i]= &arr[j]); &arr[j]=t;
7     }
8   }
9   t=&arr[i + 1];  &arr[i+1])= &arr[h]; &arr[h]=t;
10 }
```



```
1  int partition (int arr[], int l, int h){
2   int x = arr[h], i = (l - 1),t;

3   for (int j = l; j <= h- 1; j++) {
4     if (arr[j] <= x){
5       i++;
6       t=&arr[i]; &arr[i]= &arr[j]); &arr[j]=t;
7     }
8   }
9   t=&arr[i + 1];  &arr[i+1])= &arr[h]; &arr[h]=t;
10 }
```

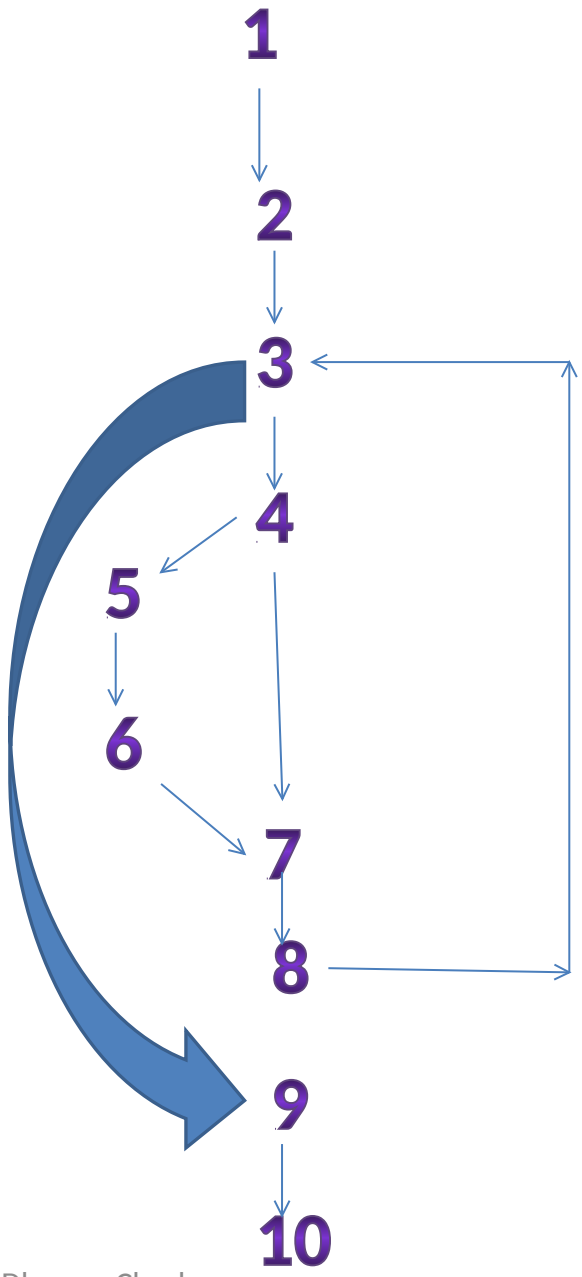


V(G)=NO OF BOUNDED AREAS + 1
=2+1
=3

V(G)=E-N+2
=11-10+2
=1+2
=3

V(G)=P+1
=2+1
=3

```
1  int partition (int arr[], int l, int h){
2    int x = arr[h], i = (l - 1),t;

3   for (int j = l; j <= h- 1; j++) {
4     if (arr[j] <= x){
5       i++;
6       t=&arr[i]; &arr[i]= &arr[j]); &arr[j]=t;
7     }
8   }
9   t=&arr[i + 1];  &arr[i+1])= &arr[h]; &arr[h]=t;
10 }
```



3 INDEPENDENT PATHS

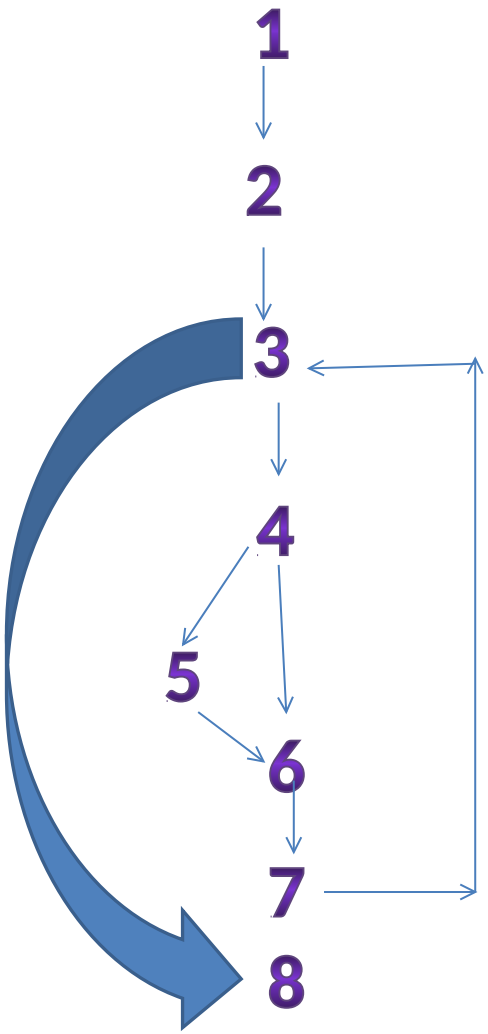
PATH1=1,2,3,4,5, 6,7,8,3,9,10

PATH2=1,2,3,4,7, 8,3,9,10

PATH3=1,2,3,9,1 0

```
min = A[0];
I = 1;

while (I < N) {
      if (A[I] < min)
         min = A[I];
      I = I + 1;
}
print min
```

```
1 min = A[0];
2 I = 1;

3 while (I < N) {
4     if (A[I] < min)
5         min = A[I];
6     I = I + 1;
7 }
8 print min
```

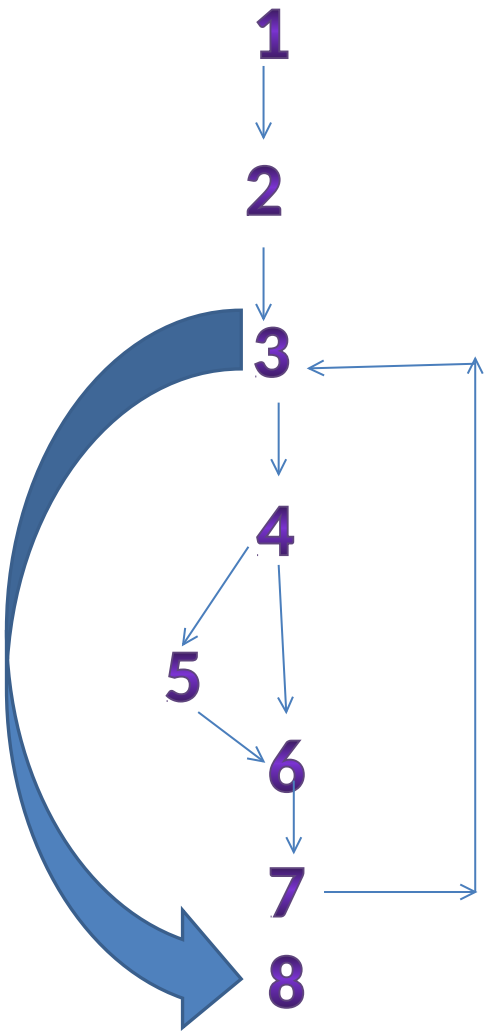


V(G)=NO OF BOUNDED AREAS + 1
=2+1
=3

V(G)=E-N+2
=9-8+2
=1+2
=3

V(G)=P+1
=2+1
=3

```
1 min = A[0];
2 I = 1;

3 while (I < N) {
4     if (A[I] < min)
5         min = A[I];
6     I = I + 1;
7 }
8 print min
```



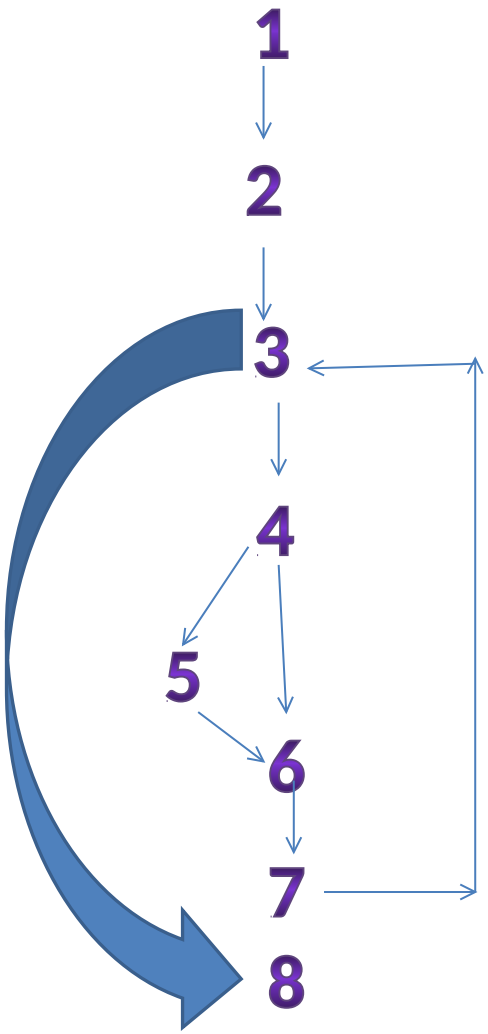
3 INDEPENDENT PATHS

PATH1=1,2,3,8

PATH2=1,2,3,4,5, 6,7,3,8

PATH3=1,2,3,4,6, 7,3,8

```
1 min = A[0];
2 I = 1;

3 while (I < N) {
4     if (A[I] < min)
5         min = A[I];
6     I = I + 1;
7 }
8 print min
```

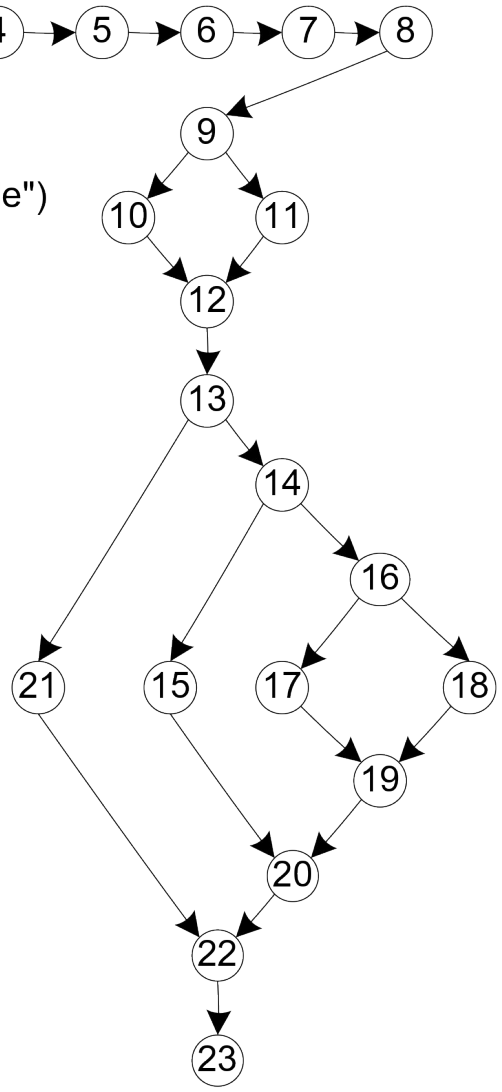


Sample Program Graph

```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATriangle As Boolean
4  Output("Enter 3 integers which are sides of a triangle")
5  Input(a,b,c)
6  Output("Side A is ",a)
7  Output("Side B is ",b)
8  Output("Side C is ",c)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10    Then IsATriangle = True
11    Else IsATriangle = False
12 EndIf
13 If IsATriangle
14    Then  If (a = b) AND (b = c)
15             Then Output ("Equilateral")
16             Else    If (a≠b) AND (a≠c) AND (b≠c)
17                        Then  Output ("Scalene")
18                        Else  Output ("Isosceles")
19                     EndIf
20             EndIf
21    Else    Output("Not a Triangle")
22 EndIf
23 End triangle2
```

Sample Program Graph

```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATriangle As Boolean
4  Output("Enter 3 integers which are sides of a triangle")
5  Input(a,b,c)
6  Output("Side A is ",a)
7  Output("Side B is ",b)
8  Output("Side C is ",c)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10    Then IsATriangle = True
11    Else IsATriangle = False
12  EndIf
13  If IsATriangle
14    Then  If (a = b) AND (b = c)
15            Then Output ("Equilateral")
16            Else    If (a≠b) AND (a≠c) AND (b≠c)
17                        Then  Output ("Scalene")
18                        Else  Output ("Isosceles")
19                    EndIf
20            EndIf
21    Else    Output("Not a Triangle")
22  EndIf
23  End triangle2
```

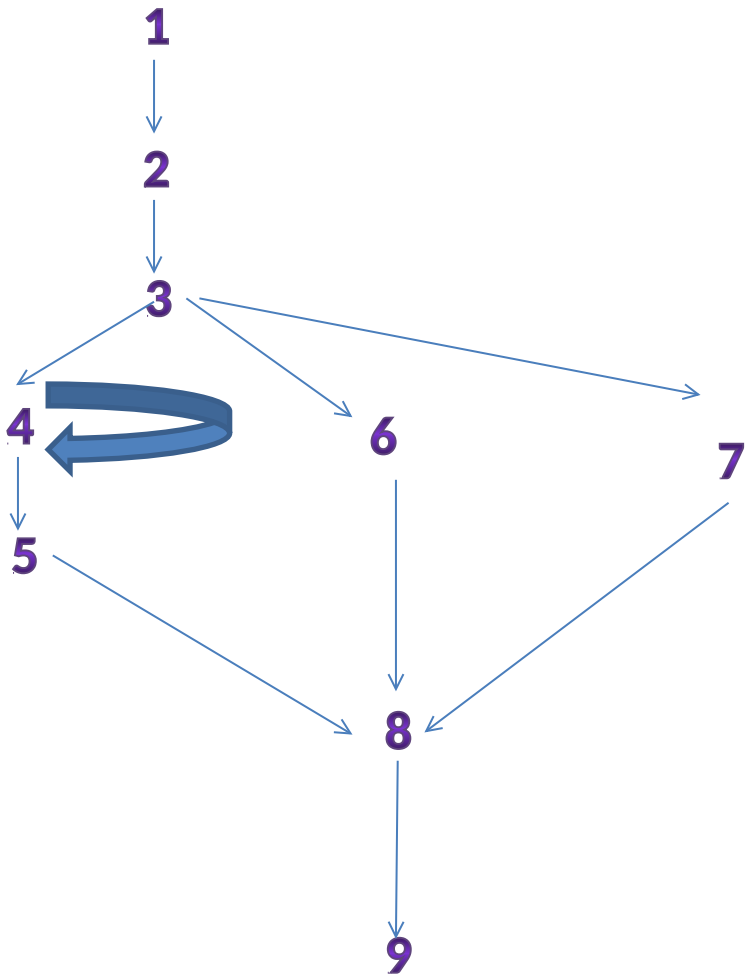


What would be the Cyclomatic complexity of the following program?

```
int     find-maximum(int i,int j,    int     k){
        int     max;
        if(i>j)     then
        if(i>k)     then     max=i;
        else        max=k;
        else        if(j>k) max=j
        else        max=k;
        return(max);
}
```

Correct Solution

```
1  int      find-maximum(int i,int j,   int      k){
2      int      max;
3      if(i>j)    then
4      if(i>k)    then     max=i;
5      else       max=k;
6      else       if(j>k) max=j
7      else       max=k;
8      return(max);
9  }
```



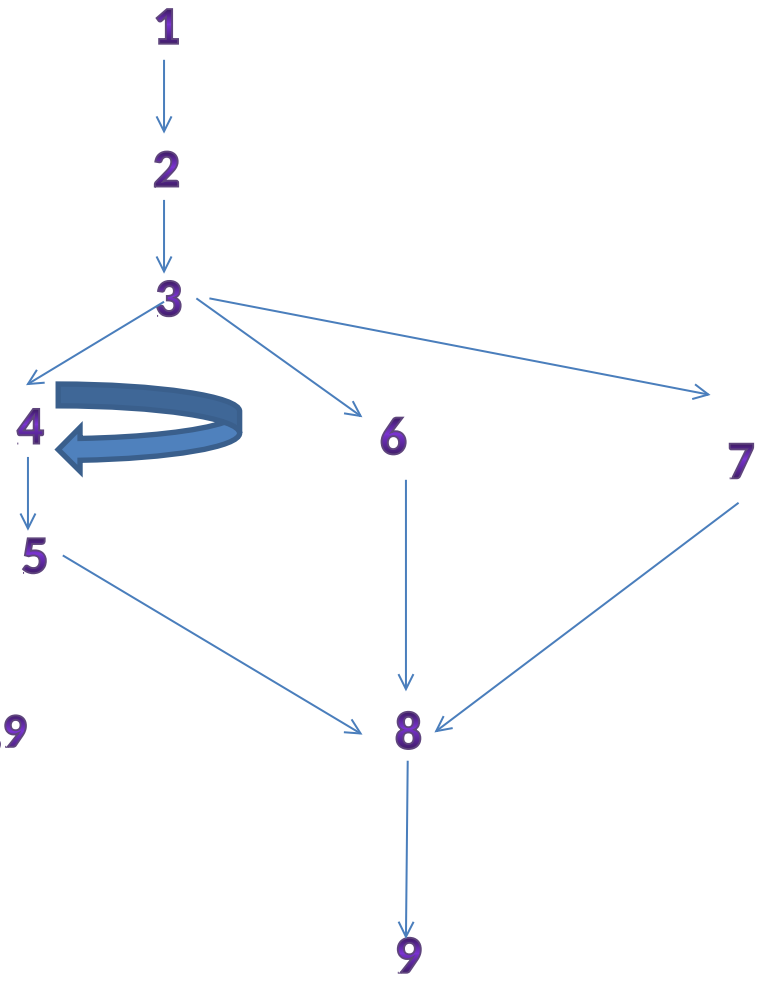
V(G)=NO OF BOUNDED AREAS + 1
=3+1
=4

V(G)=E-N+2
=11-9+2
=2+2
=4

V(G)=P+1
=3+1
=4

Solution

```
1  int      find-maximum(int i,int j,   int      k){
2     int      max;
3     if(i>j)  then
4     if(i>k)  then     max=i;
5     else     max=k;
6     else     if(j>k) max=j
7     else     max=k;
8     return(max);
9  }
```



4 INDEPENDENT PATHS

PATH1=1,2,3,4,5,8,9

PATH2=1,2,3,6,8,9

PATH3=1,2,3,7,8,9

PATH4=1,2,3,4,4,5,8,9