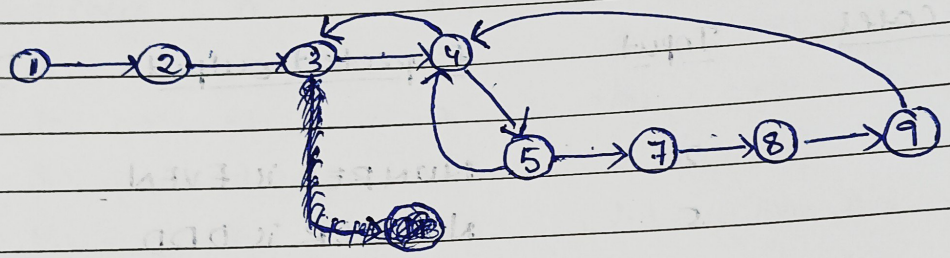
Assignment - White Box Testing

1) Consider the following program segment:

```
/* Sort takes an integer array and sorts it in ascending order */
1.  void sort (int a[], int n) {
2.      int i, j;
3.  for(int i = 0; i<n-1; i++)
4.      for(j = i+1; j<n; j++)
5.          if (a[i] > a[j])
6.          {
7.              temp = a[i];
8.              a[i] = a[j];
9.              a[j] = temp;
10.         }
11. }
```

a) Draw the program graph for this program segment.

Ans



b) Determine the cyclomatic Complexicity for this program (Show the intermediate steps of your Computation).

Ans For the given graph

the cyclomatic Complexicity V(G) = E-N+2

N = number of Nodes

E = Number of Edges.

Number of nodes (N) = 8
Number of edges (E) = 10

$V(G) = 10 - 8 + 2$

$\boxed{V(G) = 4}$

c) How is the cyclomatic Complexicity metric useful?

Cyclomatic Complexicity metric is useful in
a) estimation of structural complexicity of code
b) estimation of testing effort
c) estimation of program reliability.

Cyclomatic Complexicity metric is a measure of structural complexicity of a program. The reason for this is it is computed based on the code structure.
It can also be told as measure of the maximum number of basis paths. Thus it indicates the minimum number of test cases require to acheive path coverage.

2) Consider the following program segment:

```
1. int find-maximum (int i, int j, int k)
2. {
3.     int max
4.     if (i>j) then
5.         if (i<k) then  max = i;
6.         else  max = k;
7.     else if (j>k)  max = j;
8.         else  max = k
9.     return (max);
10. }
```

a) Draw the control flow graph for this program segment.

b) Determine the cyclomatic complexicity for this program (show the intermediate steps of your computation).

Cyclomatic Complexicity $V(G) = E - N + 2$

E - Number of Edges = 10
N - Number of Nodes = 8

$V(G) = 10 - 8 + 2$

$\boxed{V(G) = 4}$

c) How is the cyclomatic complexicity metric useful?

Cyclomatic Complexicity metric is useful in

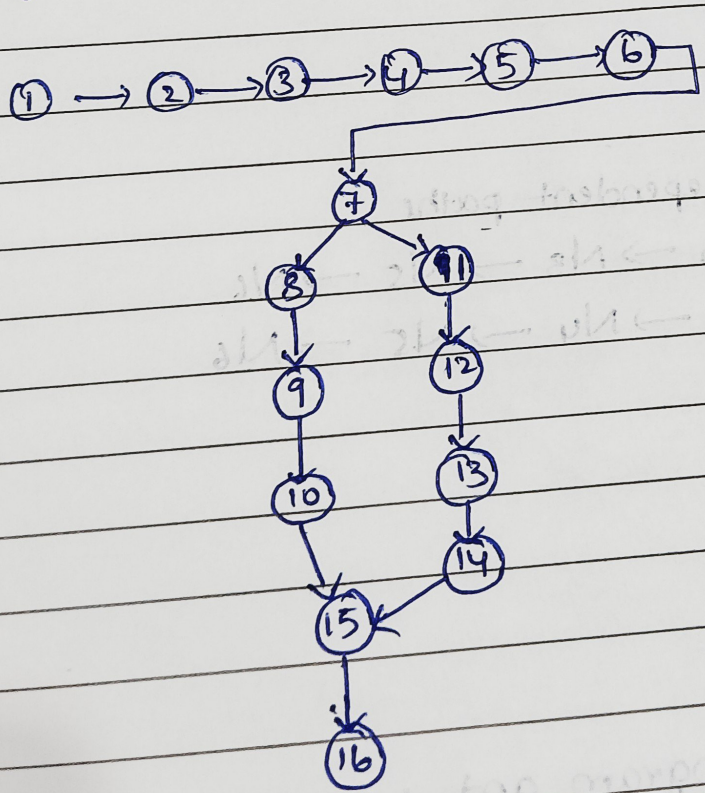
a) estimation of structural complexicity of code
b) estimation of effort
c) estimation of program realibility.

3) Write a program to determine wheather a number is even (or) odd.
Draw the program graph and DD path graph. Find the independent paths.

Ans

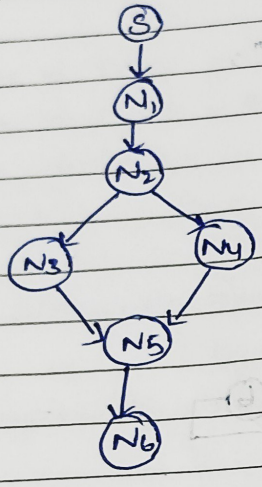
```
1. void main()
2. {
3.   int num;
4.   clrscr();
5.   cout << "Enter number";
6.   cin >> num;
7.   if (num % 2 == 0)
8.   {
9.   cout << "Number is even";
10.  }
11.  else
12.  {
13.   cout << "Number is odd";
14.  }
15.  getch();
16. }
```

Program graph



DD path graph

Program graph nodes	DD path Node	Comments
1	S	Source Node
2-6	N_1	Sequential Node
7	N_2	decision Node
8-10	N_3	Sequential Node
11-14	N_4	Sequential node
15	N_5	Junction node
16	D	Destination node



Here there are two independent paths

path-1 $S \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_5 \rightarrow N_6$

path-2 $S \rightarrow N_1 \rightarrow N_2 \rightarrow N_4 \rightarrow N_5 \rightarrow N_6$

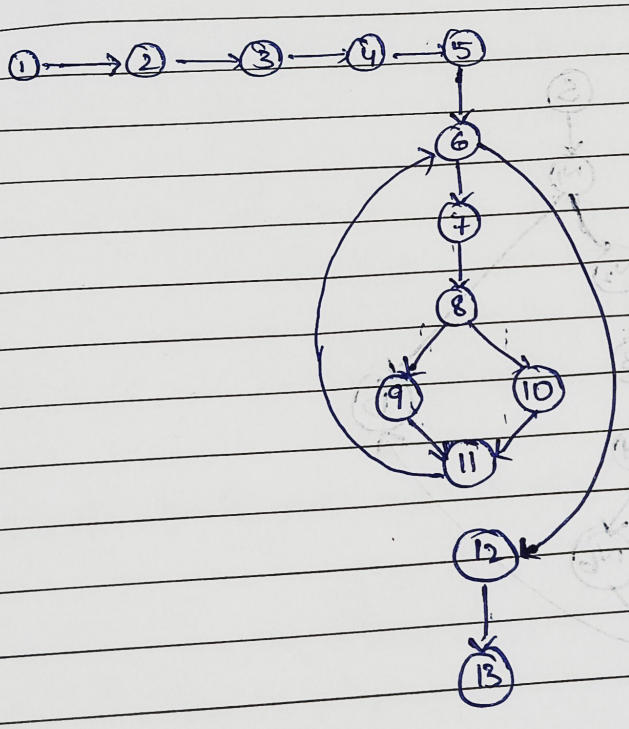
path-1 if num is even

path-2 if num is odd.

4) Consider the following program and draw the program path graph and DD path graph. Also find out cyclomatic complexicity and independent paths.

```
1. void main ()
2. {
3.   int x, y;
4.   scanf ("%d \n", &x)
5.   scanf ("%d \n", &y);
6.   while (x != y)
7.     {
8.       if (x > y)
9.           x = x - y;
10.      else  y = y - x;
11.    }
12.    printf ("x = \"%d", x);
13. }
```

Ans Program - path graph



Cyclomatic Complexicity V(G) = E - N + 2

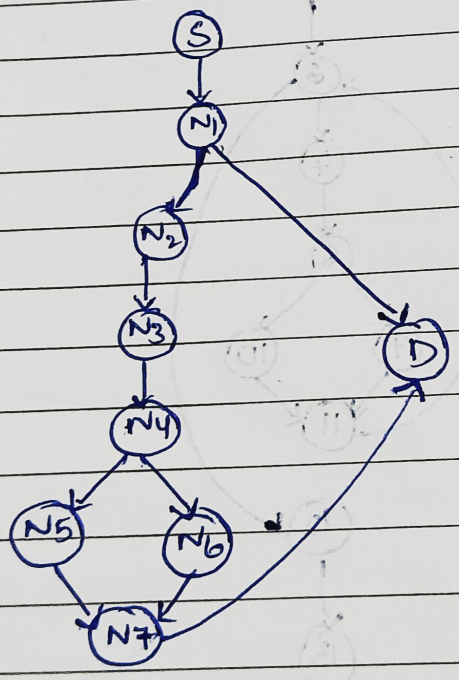
E = 14, N = 13

V(G) = 14 - 13 + 2

$V(G) = 3$

DD path graph

Program graph nodes	DD path Nodes	Comments
1	S	Source node
2-5	N_1	Sequential nodes
6	N_2	decision node
7	N_3	Sequential node
8	N_4	decision node
9	N_5	sequential node
10	N_6	Sequential node
11	N_7	Junction node
~~12~~	~~D~~	~~Sequential node~~
12- 13	D	Decision node.



Independent paths

There were total of 3 independent paths in the DD path graph.

path-1 : S → N1 → N2 → N3 → N5 → N7 → D
path-2 : S → N1 → N2 → N3 → N4 → N6 → N7 → D
path-3 : S → N1 → D

5) what is the value of the McCabe's cyclomatic Complexicity metric for the following code segment?

```
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. }
```

Identifying Decision points.

1. for (int j = l; j <= h-1; j++)
2. if (arr[j] <= x)

These two are decision points.

McCabe's cyclomatic Complexicity = D+1
= 2+1 = 3

CC = 3

6) what would be the cyclomatic Complexicity 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)
}
```

Ans Identifying Decision points

There are total 3 decision points in the program.

1. if (i>j)
2. if (i>k)
3. else if (j>k)

So D = 3

CC = D+1

cc = 3+1

cc = 4

7) What is the cyclomatic complexicity of the following C program Segment

```
int month;
Switch (month)
{
Case 1 : name = "January"; break;
Case 2 : name = "February"; break;
Case 3 : name = "March "; break;
Case 4 : name = "April" ; break;
Case 5 : name = "May" ; break;
Case 6 : name = "June" ; break;
```

```
case 7 : name = "July"; break;
case 8 : name = "August"; break;
case 9 : name = "September"; break;
case 10 : name = "October"; break;
case 11 : name = "November"; break;
case 12 : name = "December"; break;
default : name = "Invalid month"; break;
}
```

Identifying decision points

The whole switch statement is considered as single decision point

So D = 1, it has 13 branches.

CC = D + 1

CC = 1 + 1

CC = 2

8) Consider the following C function named bin-search:

```
/* num is the number the function searches in a presorted
integer array arr */
int bin_search (int num) {
    int min, max;
    min = 0;
    max = 100;
    while (min != max) {
        if (arr[(min+max)/2] > num)
            max = (min + max)/2;
        else if (arr[(min+max)/2] < num)
            min = (min+max)/2;
        else return ((min+max)/2);
    }
    return (-1);
}
```

Design a test suite for the function bin-search that satisfies the following white-box testing strategies (Show the intermediate steps in deriving the test cases):

Statement Coverage

Branch Coverage

Condition Coverage

Path Coverage.

Statement Coverage

It ensures that every excecutable statement in the program is excecuted atleast once.

Test-cases for Statement Coverage:

Test case-1 :- The element is found in the array (this ensures the return statement inside loop is excecuted)

Test case-2 :- The element is not found in the array (this ensures the return (-1) statement is excecuted)

Ex:-

Test case-1 :- num = 49

Test case-2 :- num = 102

Branch Coverage

It ensures that every Branch of Conditional statements is excecuted atleast once.

Test-cases

case-1 : The condition if (arr[(min+max)/2] > num) is true.

Case-2 : The condition if (arr[(min+max)/2] < num) is true

Case-3 The else condition is true

Ex:- Arr = {1, - - - 100}

Case-1 : num = 20 arr[50] > 20 (presorted array)

Case-2 : num = 80 arr[50] < 80

Case-3 : num = 50 arr[50] = 50

Conditional Coverage

It ensures that each condition in Conditional statement takes on all possible outcomes

Test-cases

case-1 : arr[(min+max)/2] > num is true

case-2 : arr[(min+max)/2] > num is false

case-3 : arr[(min+max)/2] < num is true

case-4 : arr[(min+max)/2] < num is false

Ex

case-1 : num=20

case-2 : num=80

case-3 : num=50

Path Coverage :

It ensures all possible paths in the program are excecuted.

Test-cases

Case-1 : First if condition is true

case-2 : The next else if condition is true

Case-3 : The number is found (else statement is excecuted)

case-4 : The array only has one element and (min == max)

Ex :

case-1 : num=25

Case-2 : num=75

Case-3 : num=50

Case-4 : Array has only one element (50) num=50

9) Consider the following C function named Sort

/* Sort takes an integer array and sorts it in ascending order */

```
void sort (int a[], int n) {
  int i, j
  for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
      if (a[i] > a[j])
      {
        temp = a[i];
        a[j] = a[j];
        a[j] = temp;
      }
}
```

a) Determine the cyclomatic complexicity of the sort function
b) Design a test-suite for the function sort that satisfies the following white box testing strategies.
i) Statement Coverage
ii) Branch Coverage
iii) Condition Coverage
iv) Path Coverage.

~~Ans~~ a) C.C = D+1

Identifying decision points.

1. for (i=0; i<n-1; i++)
2. for (j=i+1; j<n; j++)
3. if (a[i] > a[j])

So D = 3

CC = D+1

CC = 4

b) Statement Coverage:

It ensures that every executable statement in the program is executed atleast once.

Test-Cases

Case-1: Array with more than one element in unsorted order so if statement executes and swap occurs.

Ex: [5,2,1]

Case-2: Array already in sorted order so that no swap occurs.

Ex: [1,2,3,4]

Branch Coverage

It ensures that both true and false of each decision point are executed at least once.

Test-Cases

Case-1: An unsorted array [5,2,1] ensures the if condition is true and loop continues (i<n-1 and j<n both true)

Case-2: Already sorted array [1,2,3,4] ensures the if condition is false and no swap occurs and test false branches.

Case-3: A single element array [1] so (i<n-1) being false immediately.

Condition Coverage

It ensures each condition in a decision is tested for both true and false outcomes independently.

Test-Cases

Case-1: An unsorted array [5,2,1] tests the true condition a[i] > a[j]

Case-2: A sorted array [1,2,3,4] test the false condition a[i] > a[j]

Case-3: An array with two equal elements [1,1] tests the false condition a[i] > a[j]

Case-4: A single element array like [1] ensures that the for loop doesn't iterate.

Path Coverage

It ensures that all possible paths of program are excecuted at least once.

Test-Cases

case-1: An unsorted array [5,2,1] ensures the if Condition is true and swap occurs (path-1 followed)

case-2: A sorted array [1,2,3,4] ensures the if condition is false (path-2)

case-3: A single element array [1] ensures the for loop is not entered (path-3)

case-4: An empty array (path-3).