```
    {
        if(y > z)
            printf("y is greatest");
        else
            printf("z is greatest");
    }
    getch();
}
```

(a) Draw the DD graph for the program.

(b) Calculate the cyclomatic complexity of the program using all four methods.

(c) List all the independent paths.

(d) Design all the test cases from independent paths.

(e) Derive all the du-paths and dc-paths using data flow testing.

5.11 Consider the following program, which multiplies two matrices:

```
#include <math.h>
#include <stdio.h>
#define SIZE 5
main()
{
int a[SIZE][SIZE], b[SIZE][SIZE],
c[SIZE][SIZE], i, j, k, row1, colm1,
row2, colm2;

printf("Enter the order of first matrix <= %d %d \n", SIZE, SIZE);
scanf("%d%d",&row1, &colm1);
printf("Enter the order of second
matrix <= %d %d \n", SIZE, SIZE);
scanf("%d%d",&row2, &colm2);
if(colm1==row2)
    {
    printf("Enter first matrix");
    for(i=0; i<row1; i++)
        {
        for(j=0; j<colm1; j++)
        scanf("%d", &a[i][j]);
        }
    printf("Enter second matrix");
    for(i=0; i<row2; i++)
        {
        for(j=0; j<colm2; j++)
        scanf("%d", &b[i][j]);
        }
    printf("Multiplication of two
    matrices is");
        for(i=0; i<row1; i++)
        {
        for(j=0; j<colm1; j++)
            {
            c[i] [j] = 0;
            for(k=0; k<row2; k++)
            c[i][j]+ = a[i][k] +
            b[k][j];
            printf("%6d", c[i][j]);
            }
        }
    }
else
{
    printf("Matrix multiplication
    is not possible");
}
}
```

(a) Draw the DD graph for the program.

(b) Calculate the cyclomatic complexity of the program using all four methods.

(c) List all the independent paths.

(d) Design all the test cases from the independent paths.

(e) Derive all the du-paths and dc-paths using data flow testing.

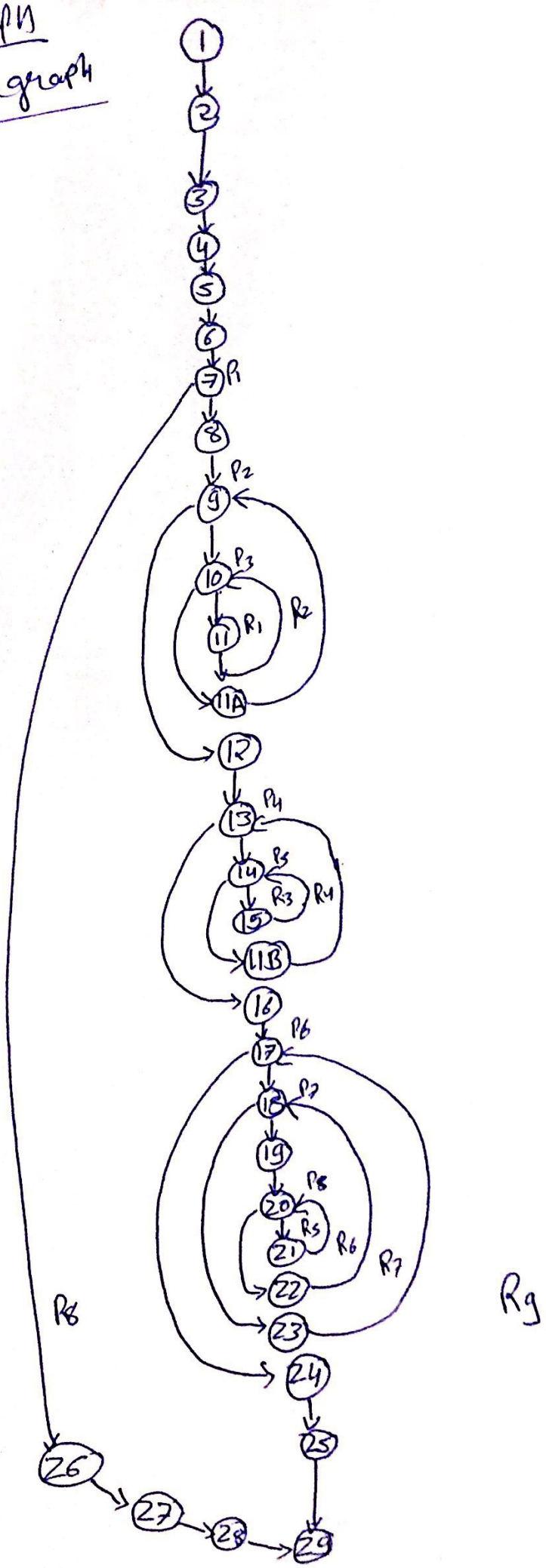
5.12 Consider the following program for finding the prime numbers, their sum, and count:

```
main()
{
    int num, flag, sum, count;
    int CheckPrime(int n);
    sum = count = 0;
    printf("Prime number between 1
    and 100 are");
    for(num=1; num<=50; num++)
{
    flag = CheckPrime(num);
    if(flag)
    {
        printf("%d", num);
        sum+ = num;
        count++;
        }
    }
    printf("Sum of primes %d", count);
}
```

CFG 5.11 & DD graph

Page 157

Decision to Decision graph



Cyclomatic Complexity

E → Edges
N → Nodes

① $V(G) = E - N + 2$
$V(G) = 38 - 31 + 2$
$V(G) = 9$

② $V(G) =$ No. of Predicate nodes $+ 1$
$V(G) = 8 + 1$
$V(G) = 9$

③ $V(G) =$ Number of regions
$V(G) = 9.$

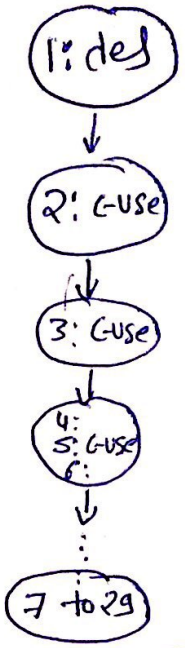
List of Independent Paths :

① 1-2-3-4-5-6-7-26-27-28-29
② 1-2-3-4-5-6-7-8-9-12-13-16-17-24-25-29
③ 1-2-3-4-5-6-7-8-9-10-11A-9-12-13-16-17-24-25-29
④ 1-2-3-4-5-6-28-9-10-11-10-11A-9-12-13-16-17-24-25-29

Here, I have explored for first "if" statement & 2 "for" loops. Like above you keep on exploring a Edge to create new path.

Derive du-path for

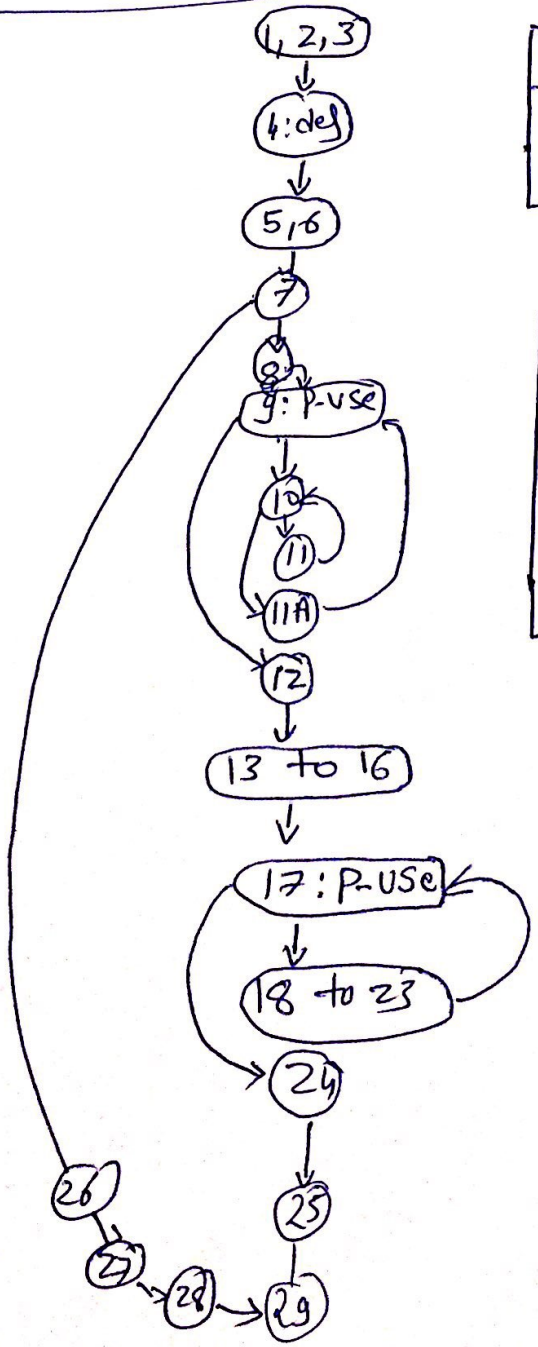
(A) SIZE:



Variable	Def at	Used at
SIZE	1	2, 3, 5

Variable	du Path (beg-end)	dc Path?
SIZE	1-2	YES
	1-3	YES
	1-5	YES

(B) row1:



Variable	Def at	Used at
row1	4	9, 17

Variable	du Path (beg-end)	dc Path?
row1	4-9	YES
	4-17	YES

Note:

Def^n: dc-Path

A path from definition node to use node with no intermediate definition node for same variable.

*: I have solved for SIZE & row1, like this you may draw for other vars.