DD path graph

As we know, flow graph generation is the first step of path testing. The second step is to dr a DD path graph from the flow graph. The DD path graph is known as decision to decision pa graph. Here, we concentrate only on decision nodes. The nodes of flow graph, which are ir sequence are combined into a single node.

Hence, DD Path graph is a directed graph in which nodes are sequences of statemer and edges represent control flow between nodes.

In order to understand the concept correctly, we consider the previous date generati program, as given in Fig. 8.15 with its flow graph as given in Fig. 8.16. The nodes number from 1 to 9 are combined together to form a new node, say n_1 due to sequential flow. If we ent into node 1, we will only be allowed to come out of node 9. Hence, flow graph of Fig. 8.16 can converted into DD Path graph using Table 8.7.

Table 8.7: Mapping of flow graph nodes and DD path graph nodes

Flow graph nodes	*DD path graph corresponding node*	*Remarks*
1 to 9	n_1	There is a sequential flow from node 1 to 9.
10	n_2	Decision node, if true goto 13 else goto 44.
11	n_3	Decision node, if true goto 12 else goto 19.
12	n_4	Decision node, if true goto 13 else goto 15.
13, 14	n_5	Sequential nodes and are combined to form new node n_5.
15, 16, 17	n_6	Sequential nodes.
18	n_7	Edges from node 14 and 17 are terminated here.
19	n_8	Decision node, if true goto 20 else goto 37.
20	n_9	Intermediate node with one input edge and one output edge.
21	n_{10}	Decision node, if true goto node 22 else, goto node 27
22	n_{11}	Intermediate node
23	n_{12}	Decision node, if true goto node 24, else node 26.
24, 25	n_{13}	Sequential nodes
26	n_{14}	Two edges from node 25 and 23 are terminated here.
27	n_{15}	Two edges from node 26 and 21 are terminated here. Also a decision node.
28, 29	n_{16}	Sequential nodes.
30	n_{17}	Decision node, if true goto 31, else goto 33.
31, 32	n_{18}	Sequential nodes
33, 34, 35	n_{19}	Sequential nodes

(Contd.)...

Flow graph nodes	*DD path graph corresponding node*	*Remarks*
36	n_{20}	Three edges from nodes, 29, 32 and 35 are terminated here.
37	n_{21}	Decision node, if true goto 38 else goto 40.
38, 39	n_{22}	Sequential nodes
40, 41, 42	n_{23}	Sequential nodes
43	n_{24}	Three edges from nodes 36, 39, and 42 are terminated here.
44	n_{25}	Decision node if true goto 45 else 82. Three edges from 18, 43 and 10 are also terminated here.
45	n_{26}	Decision node, if true goto 46 else goto 77.
46	n_{27}	Decision node, if true goto 47 else goto 51.
47, 48, 49, 50	n_{28}	Sequential nodes
51	n_{29}	Decision node, if true goto 52 else goto 68.
52	n_{30}	Intermediate node with one input edge and one output edge.
53	n_{31}	Decision node, if true goto 54 else goto 59.
54	n_{32}	Intermediate node
55	n_{33}	Decision node if true goto 56 else goto 58.
56, 57	n_{34}	Sequential nodes
58	n_{35}	Two edges from nodes 57 and 55 are terminated here.
59	n_{36}	Decision node, if true goto 60 else goto 63. Two edges from nodes 58 and 53 are terminated.
60, 61, 62	n_{37}	Sequential nodes
63, 64, 65, 66	n_{38}	Sequential nodes
67	n_{39}	Two edges from node 62 and 66 are terminated here.
68	n_{40}	Decision node, if true goto 69 else goto 72.
69, 70, 71	n_{41}	Sequential nodes
72, 73, 74, 75	n_{42}	Sequential nodes
76	n_{43}	Four edges from nodes 50, 67, 71 and 75 are terminated here.
77, 78, 79	n_{44}	Sequential nodes
80	n_{45}	Two edges from nodes 76 and 79 are terminated.
81	n_{46}	Intermediate node
82, 83, 84	n_{47}	Sequential nodes
85	n_{48}	Two edges from nodes 81 and 84 are terminated here.
86, 87	n_{49}	Sequential nodes with exit node.

The DD path graph is given in Fig. 8.17.

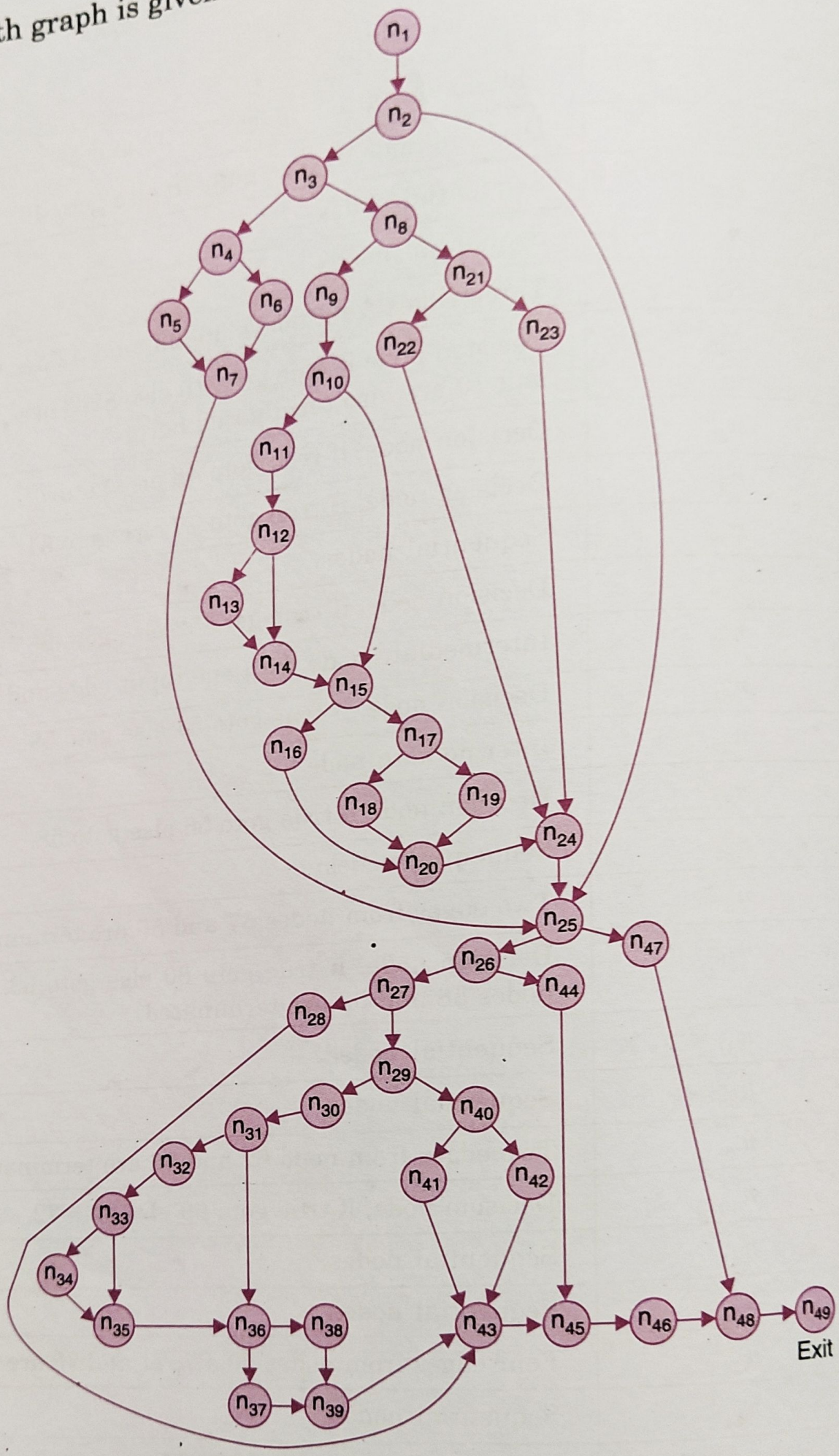


Fig. 8.17: DD path graph of previous date problem

Independent paths

The DD path graph is used to find independent paths. We are interested to execute all independent paths at least once during path testing.

An independent path is any path through the DD path graph that introduces at least one new set of processing statements or new conditions. Therefore, an independent path must move along at least one edge that has not been traversed before the path is defined.

We consider the previous date problem and its DD path graph which is given in Fig. 8.17. The independent paths are found and are given in Fig. 8.18. There are 18 independent paths.

	Independent paths of previous date problem
1	$n_1, n_2, n_{25}, n_{47}, n_{48}, n_{49}$
2	$n_1, n_2, n_3, n_4, n_5, n_7, n_{25}, n_{47}, n_{48}, n_{49}$
3	$n_1, n_2, n_3, n_4, n_6, n_7, n_{25}, n_{47}, n_{48}, n_{49}$
4	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
5	$n_1, n_2, n_3, n_8, n_{21}, n_{23}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
6	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{17}, n_{19}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
7	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{17}, n_{18}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
8	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{13}, n_{14}, n_{15}, n_{17}, n_{18}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
9	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{14}, n_{15}, n_{17}, n_{18}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
10	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
11	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{44}, n_{45}, n_{46}, n_{48}, n_{49}$
12	$n_1, n_2, n_3, n_8, n_9, n_{11}, n_{12}, n_{14}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{27}, n_{28}, n_{43}, n_{45}, n_{46}, n_{48}, n_{49}$
13	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{14}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{40}, n_{41}, n_{43}, n_{45}, n_{46}, n_{48}, n_{49}$
14	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{14}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{40}, n_{42}, n_{43}, n_{45}, n_{46}, n_{48}, n_{49}$
15	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{36}, n_{38}, n_{39}, n_{43}, n_{45}, n_{46}, n_{48}, n_{49}$
16	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{36}, n_{37}, n_{39}, n_{43}, n_{45}, n_{46}, n_{48}, n_{49}$
17	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{32}, n_{33}, n_{34}, n_{35}, n_{36}, n_{37}, n_{39}, n_{43}, n_{45}, n_{46}, n_{48}, n_{49}$
18	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{32}, n_{33}, n_{35}, n_{36}, n_{37}, n_{39}, n_{43}, n_{45}, n_{46}, n_{48}, n_{49}$

Fig. 8.18: Independent paths of previous date problem

It is quite interesting to use independent paths in order to ensure that

(*i*) Every statement in the program has been executed at least once.

(*ii*) Every branch has been exercised for true and false conditions.

There are high quality commercial tools that generate the DD path graph of a given program. The vendors make sure that the products work for wide variety of programming languages. In practice, it is reasonable to make DD Path graphs for programs upto about 100 source lines. Beyond that, we should go for a standard tool.

Example 8.13

Consider the program for the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (say a, b, c) and values may be from interval [0, 100]. The program is given in Fig. 8.19. The output may have one of the following words: [Not a quadratic equation; real roots; Imaginary roots; Equal roots]

Draw the flow graph and DD path graph. Also find independent paths from the DD path graph.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
1     int main()
2   {
3     int a,b,c,validInput=0,d;
4     double D;
5     printf("Enter the 'a' value: ");
6     scanf("%d",&a);
7     printf("Enter the 'b' value: ");
8     scanf("%d",&b);
9     printf("Enter the 'c' value: ");
10    scanf("%d",&c);
11    if ((a >= 0) && (a <= 100) && (b >= 0) && (b <= 100) && (c >= 0)
        && (c <= 100)) {
12      validInput = 1;
13      if (a == 0) {
14        validInput = -1;
15      }
16    }
17    if (validInput==1) {
18      d = b*b - 4*a*c;
19      if (d == 0) {
20        printf("The roots are equal and are r1 = r2 = %f\n",
                -b/(2*(float) a));
21      }
22      else if ( d > 0 ) {
23        D=sqrt(d);
24        printf("The roots are real and are r1 = %f and r2 = %f\n",
                (-b-D)/(2* a), (-b+D)/(2* a));
25      }
26      else {
27        D=sqrt(-d)/(2*a);
28        printf("The roots are imaginary and are r1 = (%f,%f) and
                r2 = (%f,%f)\n", -b/(2.0*a),D,-b/(2.0*a),-D);
29      }
30    }
31    else if (validInput == -1) {
```

(Contd...)

```
        printf("The values do not constitute a Quadratic equation.");
    }
    else {
        printf("The inputs belong to invalid range.");
    }
    getche();
    return 1;
}
```

Fig. 8.19: Code of quadratic equation problem

Solution

The flow graph is given below:

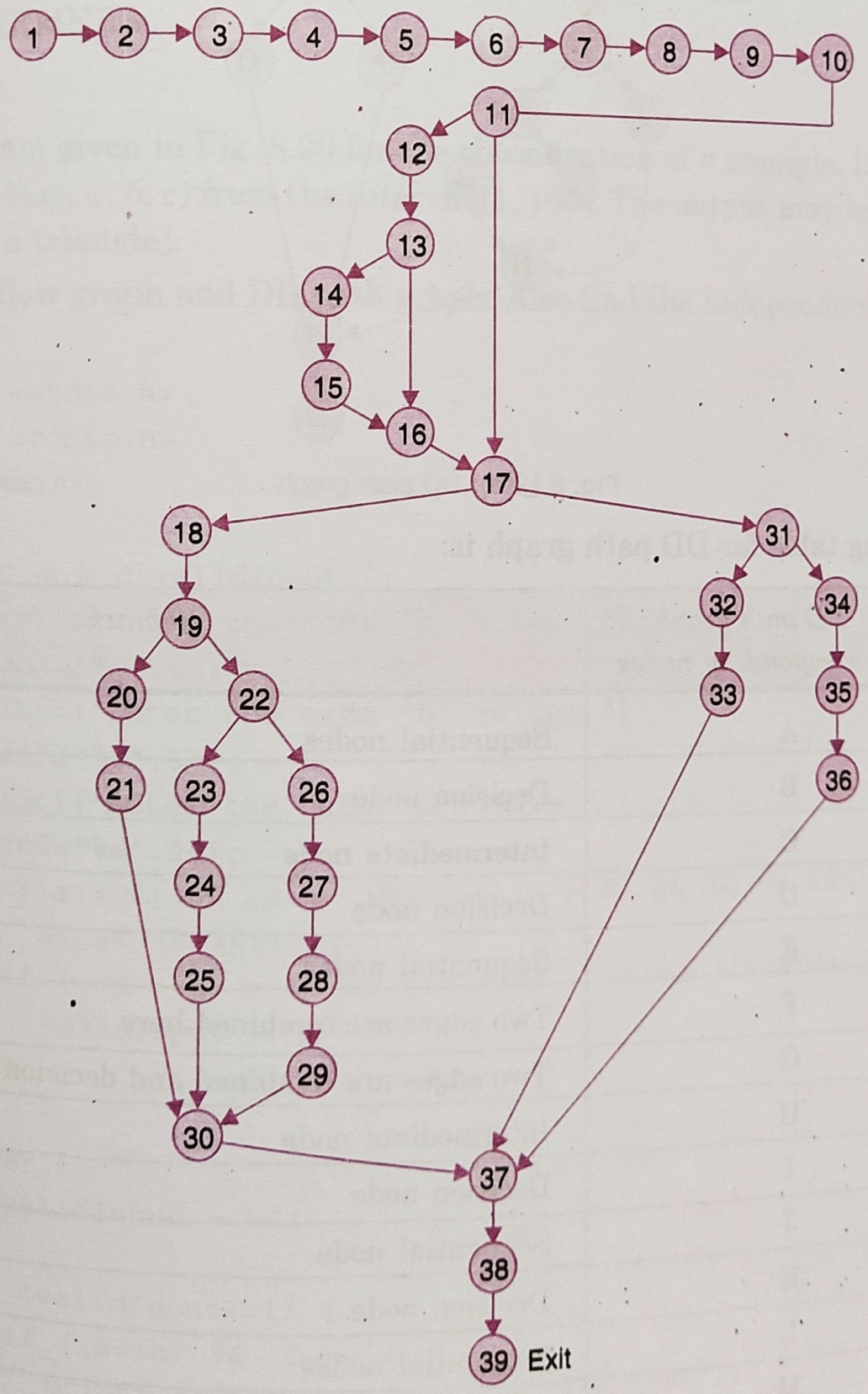


Fig. 8.19(a): Program flow graph

DD Path graph is given below :

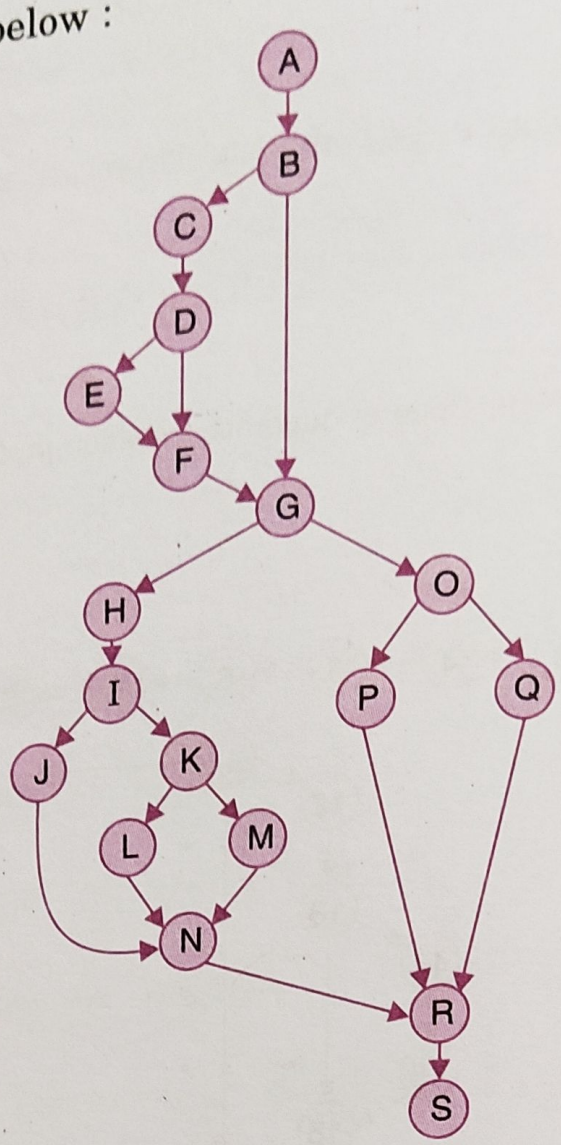


Fig. 8.19(b): DD path graph

The mapping table for DD path graph is:

Flow graph nodes	*DD path graph corresponding nodes*	*Remarks*
1 to 10	A	Sequential nodes
11	B	Decision node
12	C	Intermediate node
13	D	Decision node
14, 15	E	Sequential nodes
16	F	Two edges are combined here
17	G	Two edges are combined and decision node
18	H	Intermediate node
19	I	Decision node
20, 21	J	Sequential node
22	K	Decision node
23, 24, 25	L	Sequential nodes
26, 27, 28, 29	M	Sequential nodes

(Contd.)...

30	N	Three edges are combined
31	O	Decision node
32, 33	P	Sequential nodes
34, 35, 36	Q	Sequential nodes
37	R	Three edges are combined
38, 39	S	Sequential nodes with exit node.

Independent paths are:

(*i*) ABGOQRS
(*ii*) ABGOPRS
(*iii*) ABCDFGOQRS
(*iv*) ABCDEFGOPRS
(*v*) ABGHIJNRS
(*vi*) ABGHIKLNRS
(*vii*) ABGHIKMNRS

Example 8.14

Consider a program given in Fig. 8.20 for the classification of a triangle. Its input is a triple of positive integers (say, a, b, c) from the interval [1, 100]. The output may be [Scalene, Isosceles, Equilateral, Not a triangle].

Draw the flow graph and DD path graph. Also find the independent paths from the DD path graph.

```
#include <stdio.h>
#include <conio.h>
1    int main()
2    {
3      int a,b,c,validInput=0;
4      printf("Enter the side 'a' value: ");
5      scanf("%d",&a);
6      printf("Enter the side 'b' value: ")
7      scanf("%d",&b);
8      printf("Enter the side 'c' value:");
9      scanf("%d",&c);
10     if ((a > 0) && (a <= 100) && (b > 0) && (b <= 100) && (c > 0)
           && (c <= 100)) {
11       if ( (a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
12         validInput = 1;
13       }
14     }
15     else {
16       validInput = -1;
17     }
18     If (validInput==1) {
19       If ((a==b) && (b==c)) {
20         printf("The trinagle is equilateral");
21       }
22       else if ( (a == b) || (b == c) || (c == a) ) {
```

(Contd.)...

```
23          printf("The triangle is isosceles");
24        }
25        else {
26          printf("The trinagle is scalene");
27        }
28      }
29      else if (validInput == 0) {
30        printf("The values do not constitute a Triangle");
31      }
32      else {
33        printf("The inputs belong to invalid range");
34      }
35      getche();
36      return 1;
37  }
```

Fig. 8.20: Code of triangle classification problem

Flow graph of trinagle problem is:

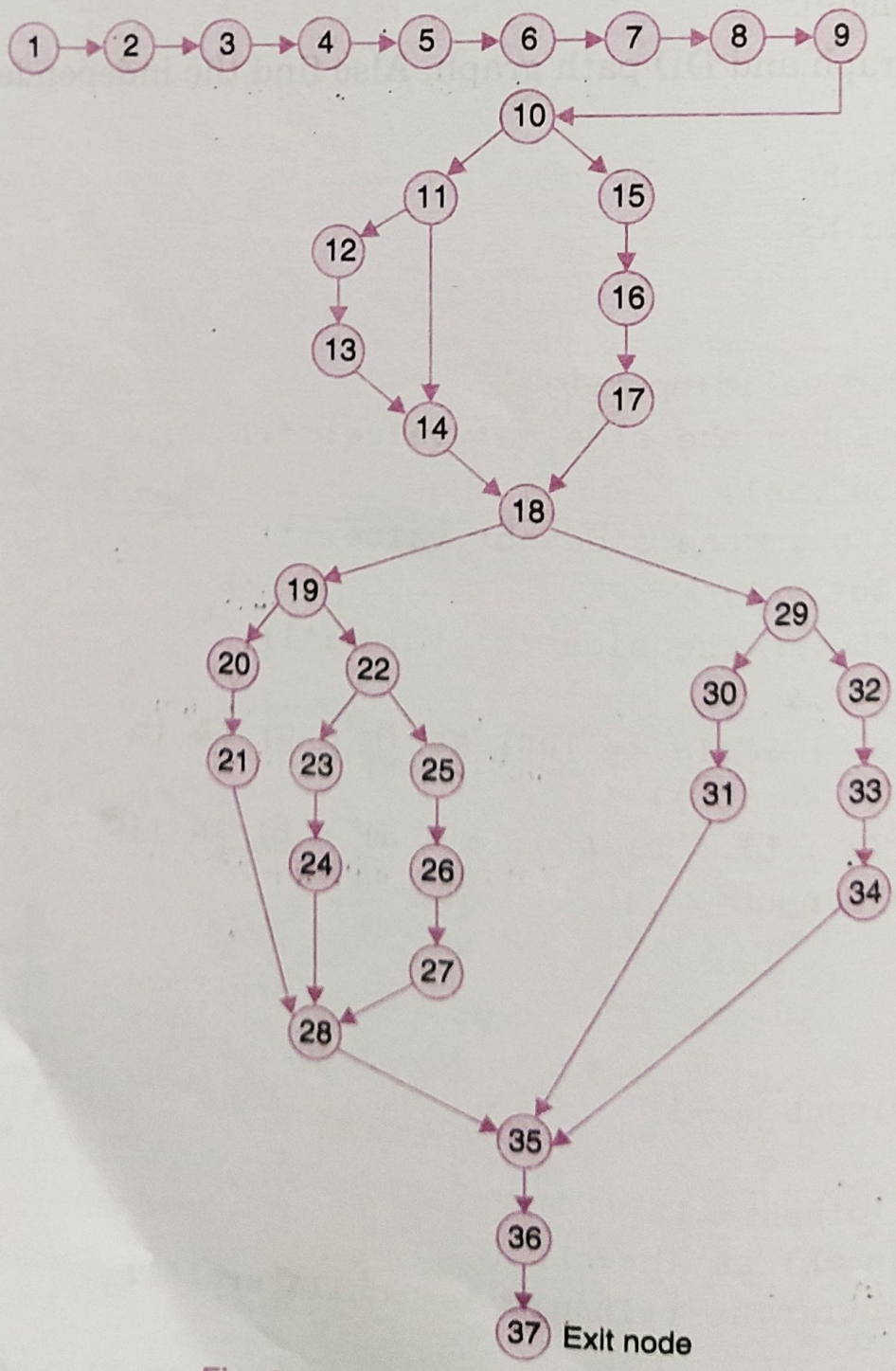


Fig. 8.20(a): Program flow graph

The mapping table for DD path graph is:

Flow graph nodes	*DD path graph corresponding nodes*	*Remarks*
1 to 9	A	Sequential nodes
10	B	Decision node
11	C	Decision node
12, 13	D	Sequential nodes
14	E	Two edges are joined here
15, 16, 17	F	Sequential nodes
18	G	Decision nodes plus joining of two edges
19	H	Decision node
20, 21	I	Sequential nodes
22	J	Decision node
23, 24	K	Sequential nodes
25, 26, 27	L	Sequential nodes
28	M	Three edges are combined here
29	N	Decision node
30, 31	O	Sequential nodes
32, 33, 34	P	Sequential nodes
35	Q	Three edges are combined here
36, 37	R	Sequential nodes with exit node

DD path graph is given in Fig. 8.20 (*b*).

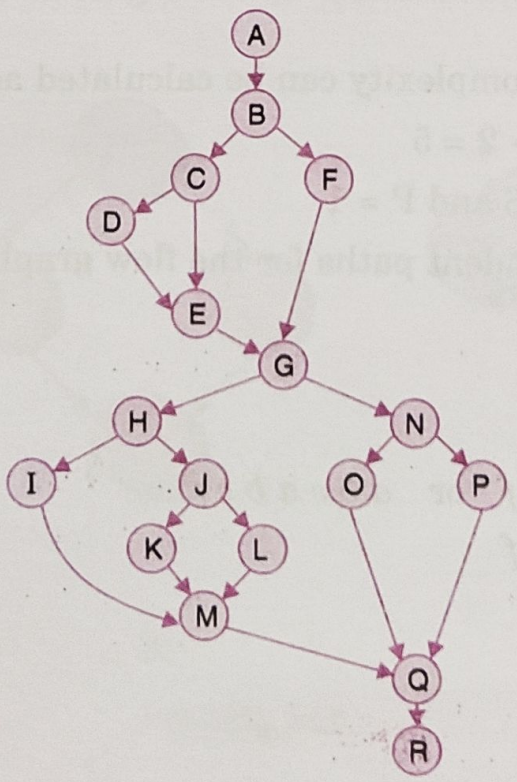


Fig. 8.20(b): DD path graph

Independent paths are:

(*i*) ABFGNPQR
(*ii*) ABFGNOQR
(*iii*) ABCEGNPQR
(*iv*) ABCDEGNOQR
(*v*) ABFGHIMQR
(*vi*) ABFGHJKMQR
(*vii*) ABFGHJLMQR

8.4.2 Cyclomatic Complexity

The cyclomatic complexity is also known as structural complexity because it gives interna view of the code. This approach is used to find the number of independent paths through program. This provides us the upper bound for the number of tests that must be conducted ensure that all statements have been executed at least once and every condition has bee executed on its true and false side. If a program has backward branch then it may have infinit number of paths. Although it is possible to define a set of algebraic expressions that gives th total number of possible paths through a program, however, using total number of paths ha been found to be impractical. Because of this, the complexity measure is defined in terms independent paths—that when taken in combination will generate every possible pat [MCCA76]. An independent path is any path through the program that introduces at least on new set of processing statements or a new condition.

McCabe's cyclomatic metric [MCCA 76] V(G) of a graph G with n vertices, e edges, an P connected components is $V(G) = e - n + 2P$.

Given a program we will associate with it a directed graph that has unique entry an exit nodes. Each node in the graph corresponds to a block of code in the program where th flow is sequential and the arcs correspond to branches taken in the program. This graph i classically known as flow graph and it is assumed that each node can be reached by the entr node and each node can reach the exit node. For example, a flow graph shown in Fig. 8.21 wit entry node '*a*' and exit node '*f*'.

The value of cyclomatic complexity can be calculated as

$$V(G) = 9 - 6 + 2 = 5$$

Here $e = 9$, $n = 6$ and $P = 1$

There will be five independent paths for the flow graph illustrated in Fig. 8.21.

path 1 : *a c f*
path 2 : *a b e f*
path 3 : *a d c f*
path 4 : *a b e a c f* or *a b e a b e f*
path 5 : *a b e b e f*

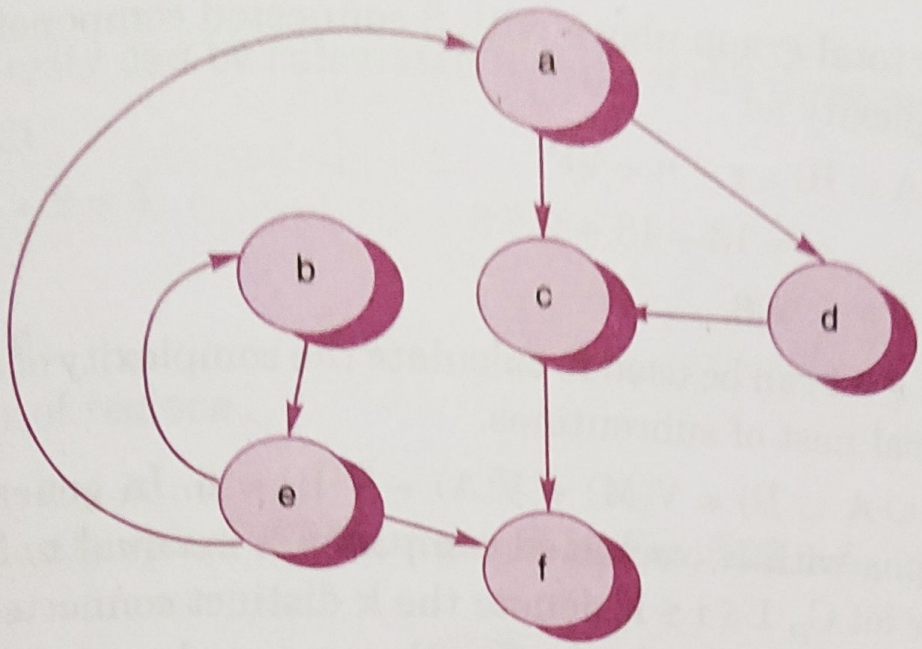


Fig. 8.21

Notice that the sequence of an arbitrary number of nodes always has unit complexity and that cyclomatic complexity conforms to our intuitive notion of minimum number of paths. Several properties of cyclomatic complexity are stated below:

1. $V(G) \geq 1$
2. V(G) is the maximum number of independent paths in graph G.
3. Inserting and deleting functional statements to G does not affect V(G).
4. G has only one path if and only if $V(G) = 1$.
5. Inserting a new row in G increases V(G) by unity.
6. V(G) depends only on the decision structure of G.

The role of P in the complexity calculation $V(G) = e - n + 2P$ is required to be understood correctly. We define a flow graph with unique entry and exit nodes, all nodes reachable from the entry, and exit reachable from all nodes. This definition would result in all flow graphs having only one connected component. One could, however, imagine a main program M and two called subroutines A and B having a flow graph shown in Fig. 8.22.

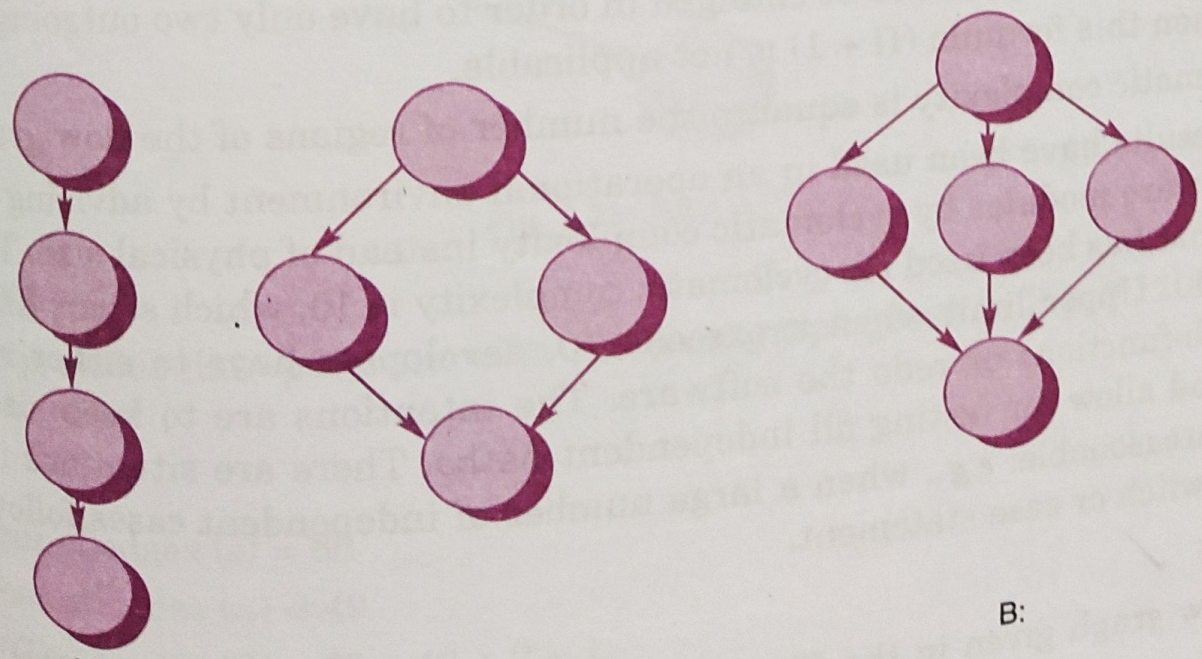


Fig. 8.22

Let us denote the total graph above with 3 connected components as M ∪ A ∪ B. Since P = 3, we calculate complexity as

$$V(M \cup A \cup B) = e - n + 2P$$
$$= 13 - 13 + 2 * 3$$
$$= 6$$

This method with P ≠ 1 can be used to calculate the complexity of a collection of programs, particularly a hierarchical nest of subroutines.

Notice that V(M ∪ A ∪ B) = V(M) + V(A) + V(B) = 6. In general, the complexity of a collection C of flow graphs with K connected components is equal to the summation of their complexities. To see this let C_i, $1 \le i \le K$ denote the k distinct connected component, and let e_i and n_i be the number of edges and nodes in the ith-connected component. Then

$$V(C) = e - n + 2p = \sum_{i=1}^{k} e_i - \sum_{i=1}^{k} n_i + 2K$$

$$= \sum_{i=1}^{k} (e_i - n_i + 2) = \sum_{i=1}^{k} V(C_i)$$

Since the calculation V = e − n + 2P can be quite tedious for a developer, an effort has been made to simplify the complexity calculations. Two alternate methods are available for the complexity calculations.

1. Cyclomatic complexity V(G) of a flow graph G is equal to the number of predicate (decision) nodes plus one [MILL72].

$$V(G) = \Pi + 1$$

Where Π is the number of predicate nodes contained in the flow graph G.

The only restriction is that every predicate node should have two outgoing edges i.e., one for "true" condition and another for "false" condition. If there are more than two outgoing edges, the structure is required to be changed in order to have only two outgoing edges. If it is not possible, then this formula (Π + 1) is not applicable.

2. Cyclomatic complexity is equal to the number of regions of the flow graph.

These results have been used in an operational environment by advising developers to limit their software modules by cyclomatic complexity instead of physical size. The particular upper bound that has been used for cyclomatic complexity is 10, which seems like reasonable, but not magical. Upper limit when it exceeds 10, developers have to either recognise and modularise sub-functions or redo the software. The intentions are to keep size of modules manageable and allow for testing all independent paths. There are situations in which this limit seems unreasonable: *e.g.*, when a large number of independent cases follow a selection function like switch or case statement.

Example 8.15

Consider a flow graph given in the Fig. 8.23 and calculate the cyclomatic complexity by all three methods.

Solution

Cyclomatic complexity can be calculated by any of the three methods.

1. $V(G) = e - n + 2P$
   $= 13 - 10 + 2 = 5$
2. $V(G) = \Pi + 1$
   $= 4 + 1 = 5$
3. $V(G)$ = number of regions
   $= 5$

Therefore, complexity value of a flow graph in Fig. 8.23 is 5.

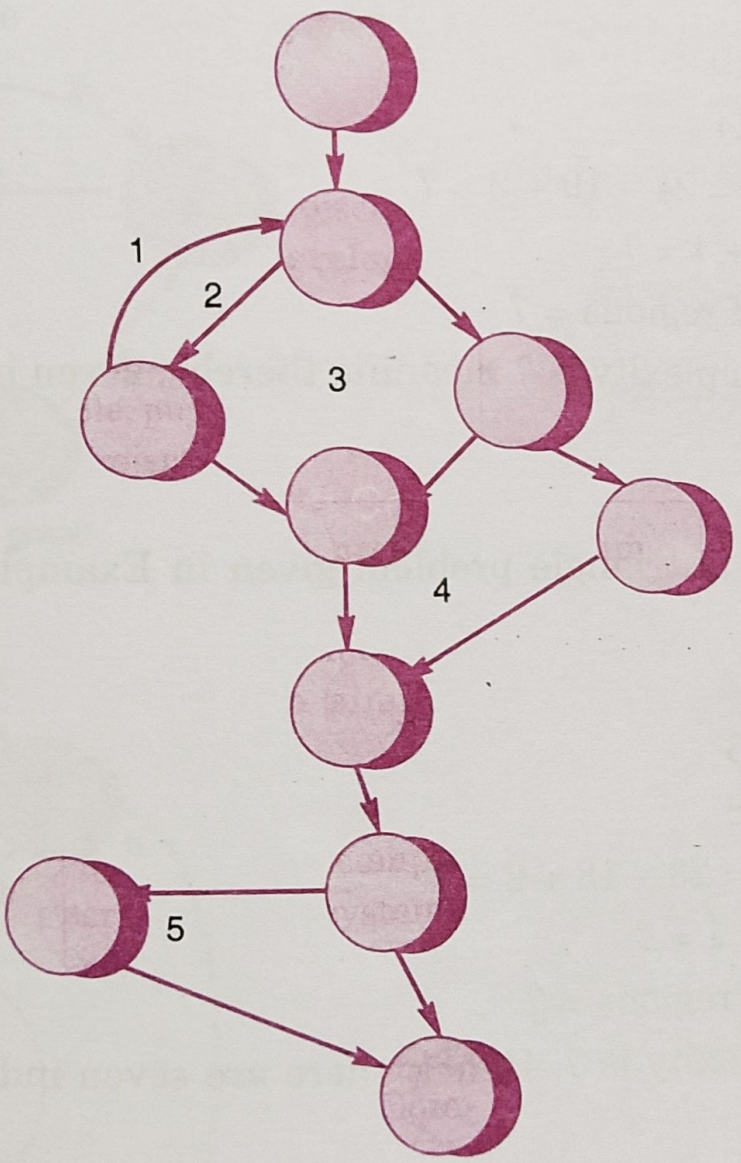


Fig. 8.23: [MCCA76]

Example 8.16

Consider the previous date program with DD path graph given in Fig. 8.17. Find cyclomatic complexity.

Solution

Number of edges (e) = 65

Number of nodes (n) = 49

(*i*) $V(G) = e - n + 2P = 65 - 49 + 2 = 18$

(*ii*) $V(G) = \Pi + 1 = 17 + 1 = 18$

(*iii*) $V(G)$ = Number of regions = 18.

Therefore cyclomatic complexity is 18. This value is quite high and indicates that there is a need to redesign the program. If we review the code, it is clear that we may have two separate modules, one for "checking the validity of date" and another for "previous date calculation". Actually, these two functions are independent and should be placed in different modules. If we do so, cyclomatic complexity will be reduced to 10 and 8 respectively (split occurs at n_{25}). Hence, this method gives us some idea about the structure of the code and modularity of the program.

Example 8.17

Consider the quadratic equation problem given in Example 8.13 with its DD path graph. Find the cyclomatic complexity:

Solution

Number of nodes = 19

Number of edges = 24

(*i*) V(G) = $e - n + 2P = 24 - 19 + 2 = 7$

(*ii*) V(G) = $\Pi + 1 = 6 + 1 = 7$

(*iii*) V(G) = Number of regions = 7

Hence cyclomatic complexity is 7 meaning thereby, seven independent paths in the DD path graph.

Example 8.18

Consider the classification of triangle problem given in Example 8.14. Find the cyclomatic complexity.

Solution

Number of edges = 23

Number of nodes = 18

(*i*) V(G) = $e - n + 2P = 23 - 18 + 2 = 7$

(*ii*) V(G) = $\Pi + 1 = 6 + 1 = 7$

(*iii*) V(G) = Number of regions = 7

The cyclomatic complexity is 7. Hence, there are seven independent paths as given in Example 8.14.

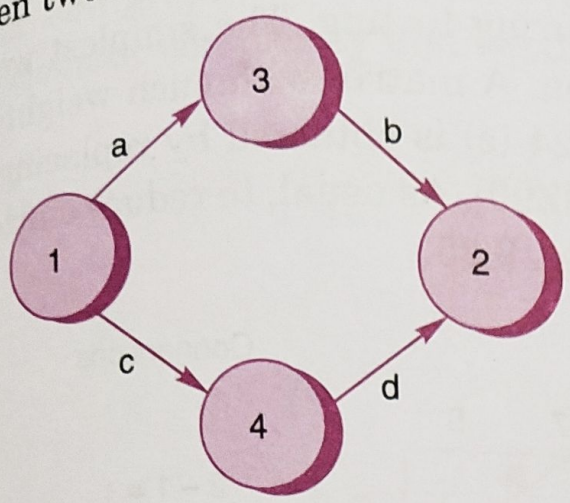
8.4.3 Graph Matrices

Whenever graphs are used for testing, we are interested to find independent paths. The objective is to trace all links of the graph at least once. Path tracing is not an easy task and is subject to errors. If the size of graph increases, it becomes difficult to do path tracing manually. In practice, it is always advisable to go for testing tool. To develop such a tool, a data structure, called graph matrix can be quite helpful.

A graph matrix is a square matrix with one row and one column for every node in the graph. The size of the matrix (*i.e.*, the number of rows and columns) is equal to the number of nodes in the flow graph. Some examples of graphs and associated matrices [BEIZ90] are shown in Fig. 8.24.

In the graph matrix, there is a place to put every possible direct connection between any node and any other node. A connection from node i to node j does not imply a connection from

node j to node i. In Fig. 8.24 (c) the (5, 6) entry is m but the (6, 5) entry is c. If there are several links between two nodes, then the entry is a sum ; the '+' sign denotes parallel links.

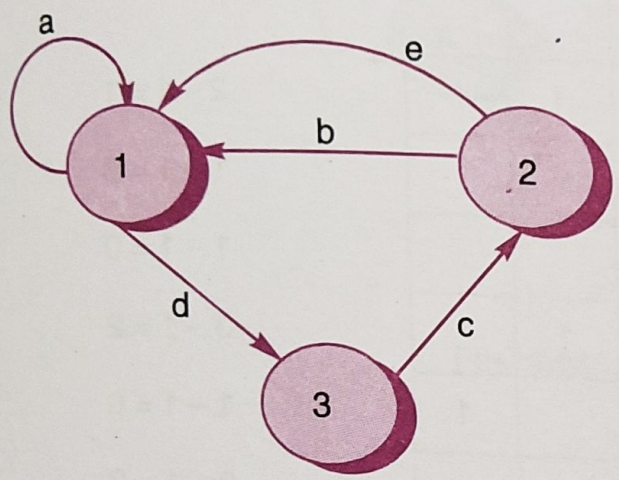


Flow graph

	1	2	3	4
1			a	c
2				
3		b		
4		d		

Graph matrix

(a)

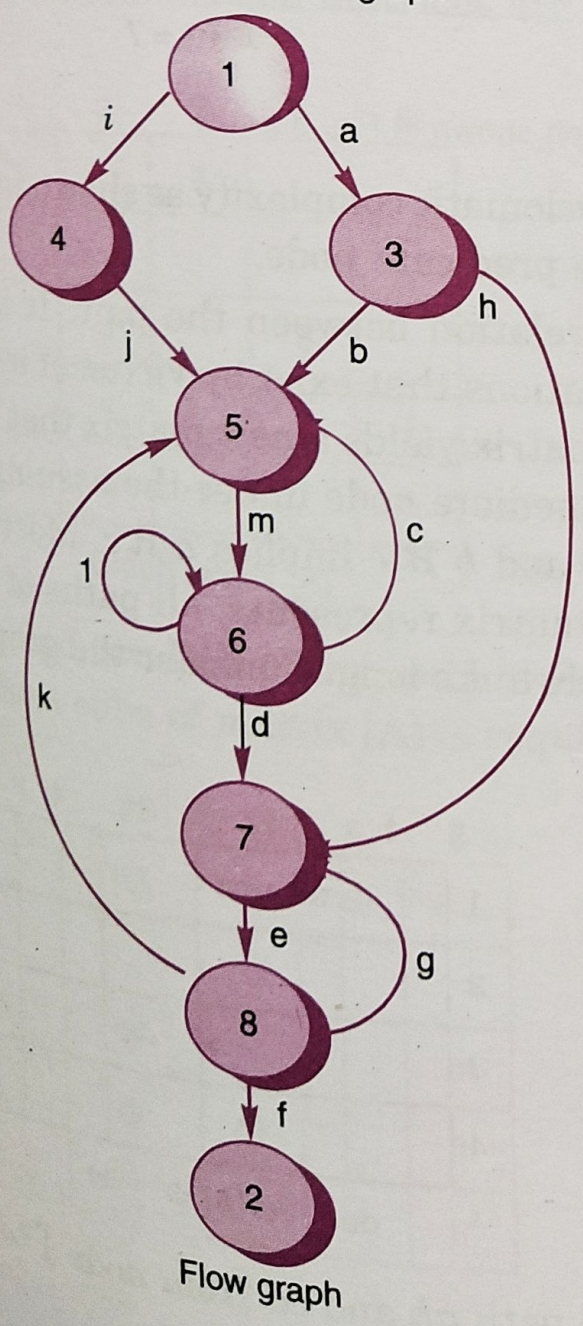


Flow graph

	1	2	3
1	a		d
2	b + e		
3		c	

Graph matrix

(b)



Flow graph

	1	2	3	4	5	6	7	8
1			a	i				
2								
3					b		h	
4					j			
5						m		
6					c	l	d	
7								e
8		f			k		g	

Graph matrix

(c)

Fig. 8.24: Flow graphs and graphs matrices