Government College of Engineering, Jalgaon

Department of Computer Engineering

Expt.-No 5

Name of Student:- Ishwari Rajiv Narkhede

PRN :- 1841034

Batch :-B4

Date of Performance: - **Date of completion:-**

EXPERIMENT NO:-

TITLE: - Measure Cyclomatic Complexity of given program

AIM: - Measure Cyclomatic Complexity of given program

REQUIRED S/W: -

THEORY:-

Cyclomatic Complexity:

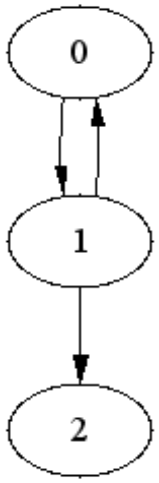
Cyclomatic complexity (or conditional complexity) is software metric (measurement). It was developed by Thomas J. McCabe, Sr. in 1976 and is used to indicate the complexity of a program. The name Cyclomatic Complexity is quite misleading for a programmer, as this metric does not only count cycles (loops) in the program. It is motivated by the number of different cycles in the program control flow graph, after having added an imagined branch back from the exit node to the entry node, A better name for popular usage would be Conditional Complexity, as "it has been found to be more convenient to count conditions instead of predicates when calculating complexity".

It directly measures the number of linearly independent paths through a program's source code. Complexity is measured by counting ‘decisions’ - the more decisions a program goes through in a single run, the more logical branches it has, and the more complex it becomes.
The Cyclomatic Complexity Metric "measures the amount of decision logic in a single software module". In my implementation I have taken the software module to be a function. The Cyclomatic complexity "is based entirely on the structure of the software's control flow graph". The formula for calculating the Cyclomatic complexity for each function using its control flow graph is:

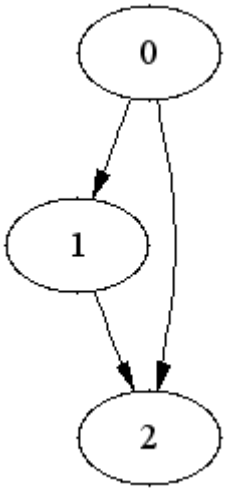
V (G) = E-N + 2

Where E and N are the number of edges and nodes in the control flow graph, respectively. Displays the control flow graphs for the control structures found in the C++ language except for. Understanding of the Cyclomatic Complexity Metric is that it measures how complex a function is and this determines how easy it will be to test the specific function.

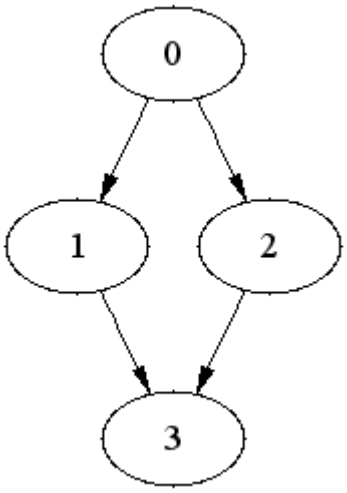
The name Cyclomatic Complexity is quite misleading for a programmer, as this metric does not only count cycles (loops) in the program. It is motivated by the number of different cycles in the program control flow graph, after having added an imagined branch back from the exit node to the entry node, A better name for popular usage would be Conditional Complexity, as "it has been found to be more convenient to count conditions instead of predicates when calculating complexity".



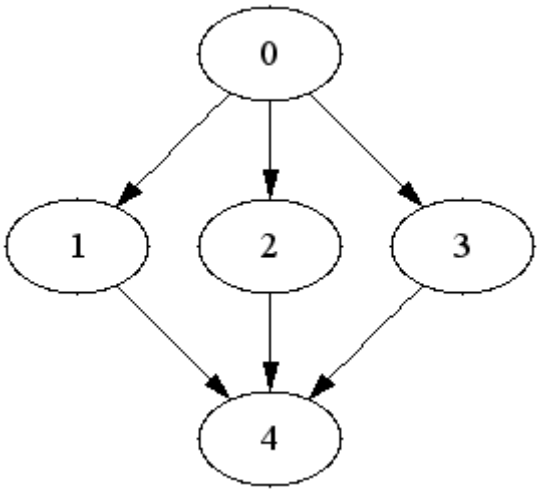
if



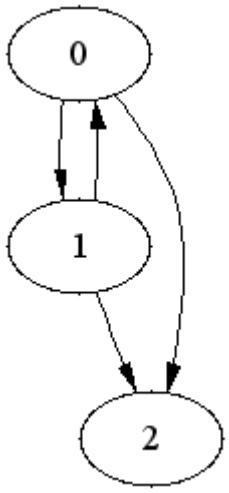
if/else



while



do



switch

Cyclomatic complexity has a foundation in graph theory and provides us with extremely useful software metric. A more refined measure is the Cyclomatic complexity measure proposed by McCabe, which is a graph-theory-based concept. Complexity is computed in one of the following ways:

1. The number of regions of the flow graph corresponds to the Cyclomatic complexity.
2. Cyclomatic complexity, V (G), for a flow graph G, defined as

$$\mathbf{V(G) = E\text{-}N + 2}$$

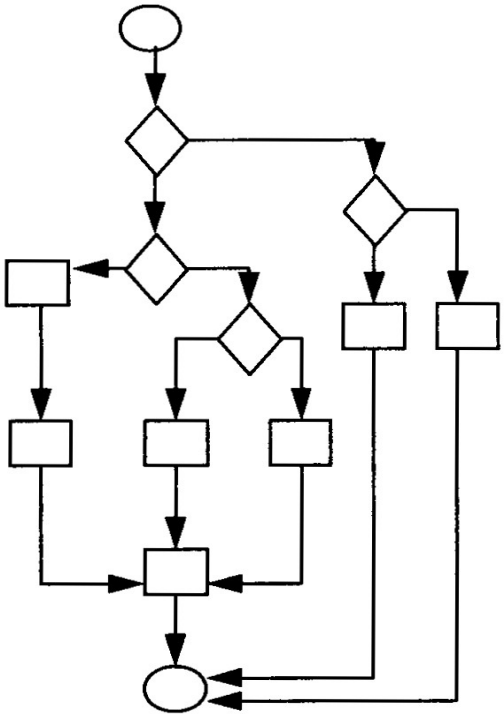
Where,
E= the number of flow graph edges
N= the number of flow graph nodes

3. Cyclomatic complexity, V (G), for a flow graph, G, is also defined as

$$\mathbf{V(G) = P + 1}$$

Where,
P= the number of predicate nodes contained in the flow graph G.

To use these formulas to define the Cyclomatic complexity of a module, the control flow graph G of the module is First drawn (figure 1.1). To construct a control flow graph of a program module, break the module into blocks delimited by Statements that affect the control flow, like *if, while, repeat,* and *goto.* These blocks from the nodes of the graph. If the Control from a block *i* can branch to a block *j*, then draw an arc from node *i* to node *j* in the graph. The control flow of a program can be constructed mechanically using either a procedural or object oriented languages.



If *G* is the control flowgraph of program *P* and *G* has *e* edges (arcs) and *n* nodes

$$v(G) = e - n + 2$$

v*(*G*) is the number of linearly independent paths in *G

here *e* = 16 *n* = 13 *v*(*G*) = 5

More simply, if *d* is the number of decision nodes in *G* then

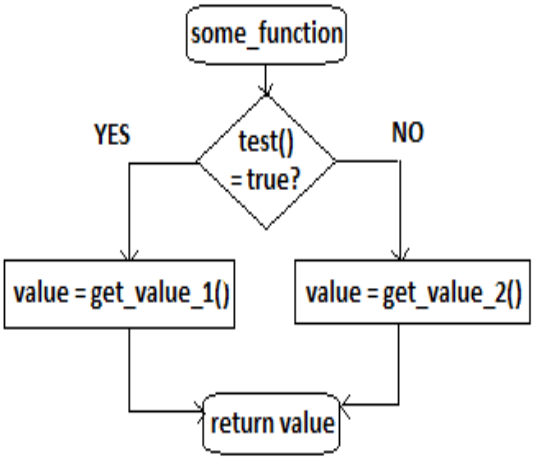
$$v(G) = d + 1$$

The number it is calculated in a more complex way, but as a general rule of thumb, it is the number of total decisions that can be made (logical if, try catches, for loops) plus one.

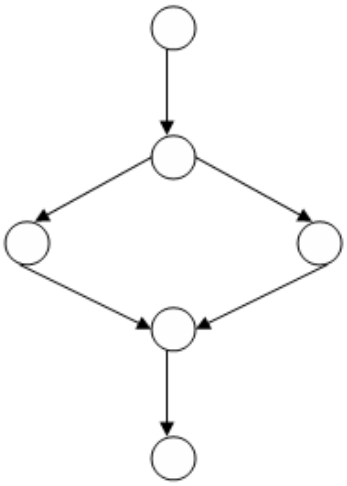
Example 1. Consider the following code:

```
def some_function ():
    if (test() ):

        value = get_value_1 ()

    else:

        value = get_value_2 ()

    return value
```

To measure cyclomatic complexity, we count the number of logical groups in the code. In this case, the code branches at the <if> statement. If we are to represent this as a flow chart,



Which can be translated into a graph that looks like this:



Conclusion:

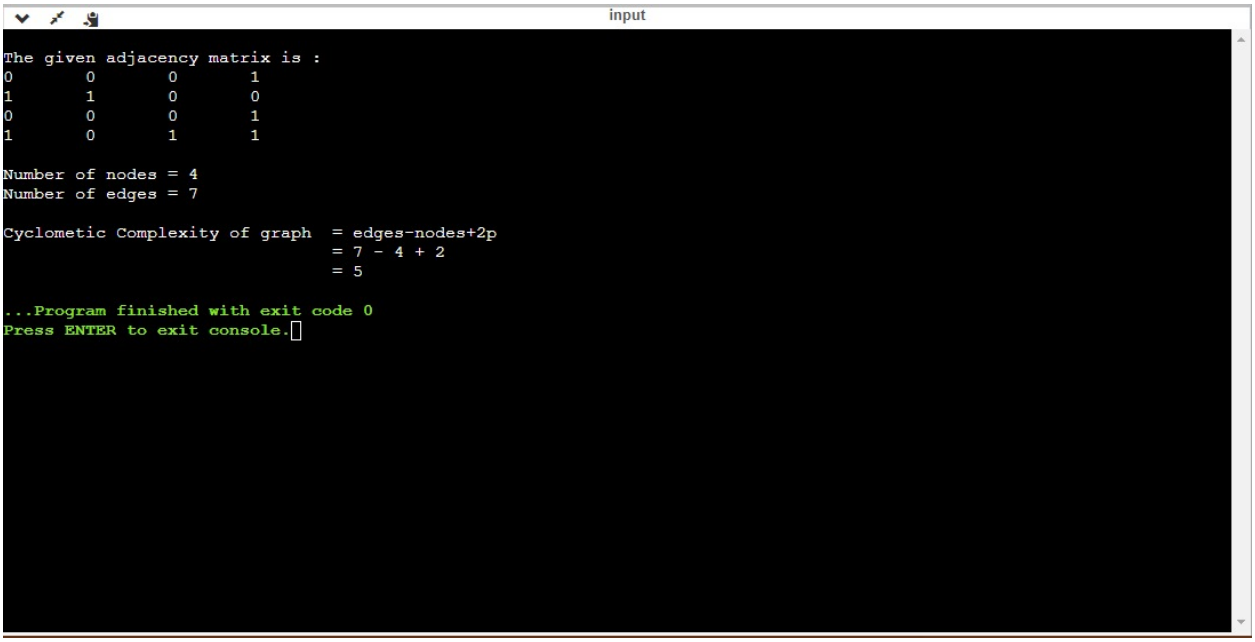
Hence,we have learned how to Measure Cyclomatic Complexity of given program.

PROGRAM :-

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10

void main()
{
  FILE *fp;
  int row,col,i,j,edges,nodes;
  char ch;
  int mat[MAX][MAX];

  clrscr();

  fp=fopen("matrix.txt","r");

  row=col=0;

  while(fscanf(fp,"%c",&ch)!=EOF)
  {
    if(ch==' ')
      col++;
    else if(ch=='\n')
    {
      row++;
      col=0;
    }
    else
      mat[row][col]=ch-48;
   }

fclose(fp);

printf("\nThe given adjacency matrix is : \n");

for(i=0;i<=row;i++)
{
  for(j=0;j<=col;j++)
    printf("%d\t",mat[i][j]);
```

```
  printf("\n");
}

nodes=row+1;
printf("\nNumber of nodes = %d",nodes);

edges=0;

for(i=0;i<=row;i++)
{
  for(j=0;j<=col;j++)
  {
    if(mat[i][j]==1)
      edges++;
  }
}
printf("\nNumber of edges = %d",edges);
printf("\n\nCyclometic Complexity of graph  = edges-nodes+2p\n");
printf("\t\t\t= %d - %d + 2",edges,nodes);
printf("\n\t\t\t\t= %d",edges-nodes+2);


getch();
}
```

OUTPUT :-


input
The given adjacency matrix is :
0 0 0 1
1 1 0 0
0 0 0 1
1 0 1 1
Number of nodes = 4
Number of edges = 7
Cyclometic Complexity of graph = edges-nodes+2p
= 7 - 4 + 2
= 5
...Program finished with exit code 0
Press ENTER to exit console.