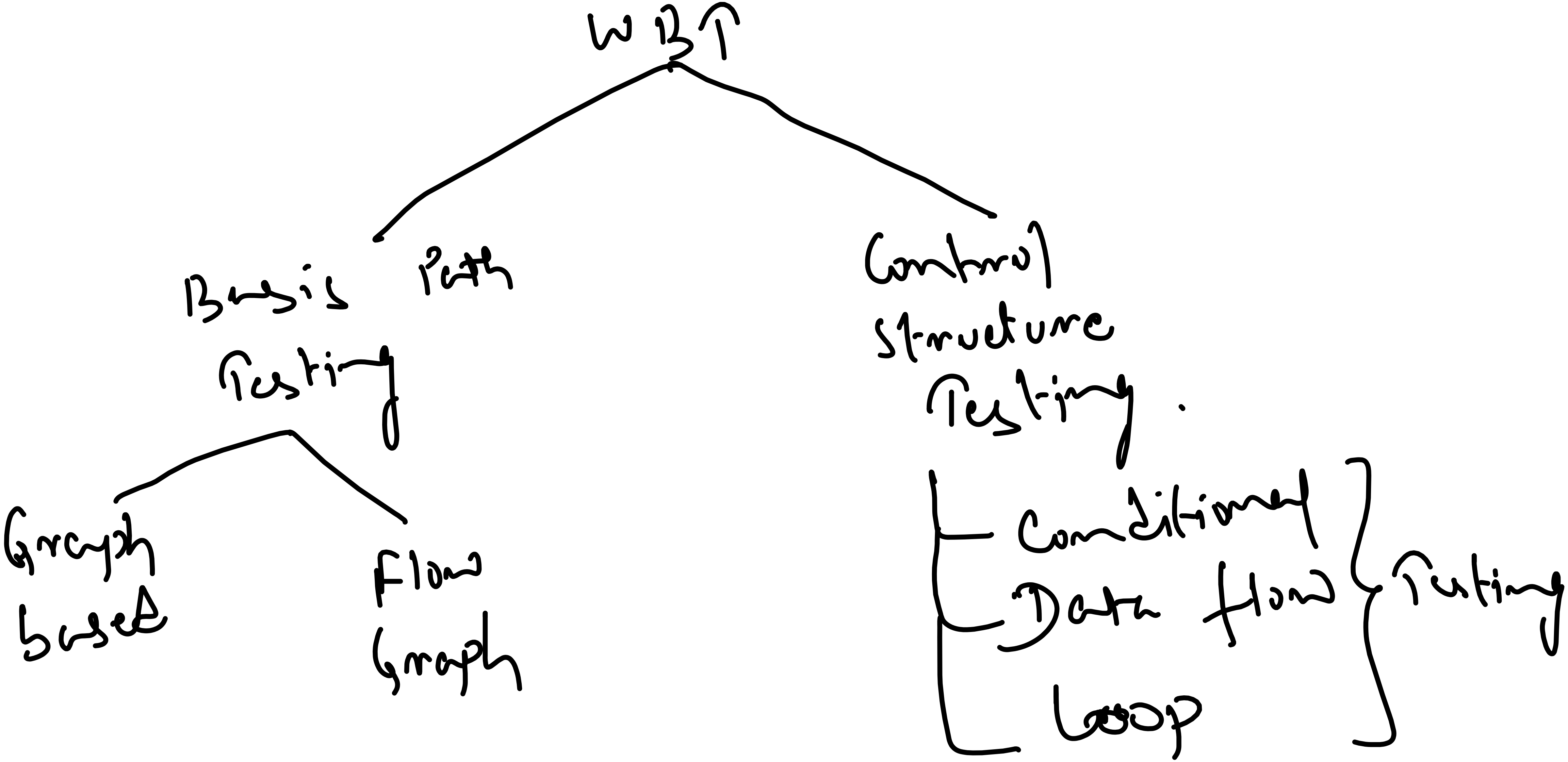
SOFTWARE TESTING

① White Box Testing – internal specification/
(WBT)
logic driven approach.

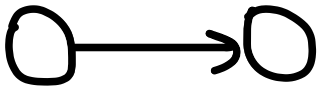
② Black Box Testing – external specification/
(BBT)
I/O driven approach


WBT
Basis Path Testing
Control Structure Testing.
Graph based
Flow Graph
Conditional
Data flow
Loop
Testing

BASIS PATH TESTING

① Proposed — Tom McCabe.

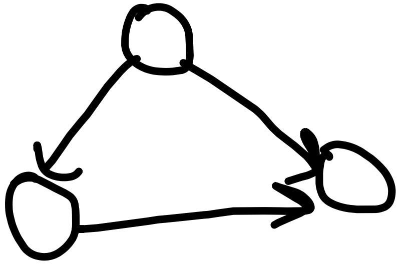
* Flow Graph Notation

① Sequential 

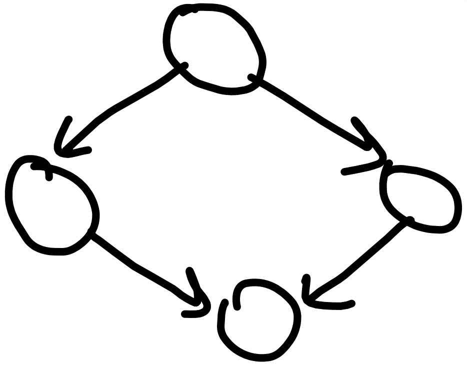
② Selective statements.

a) Bidirectional

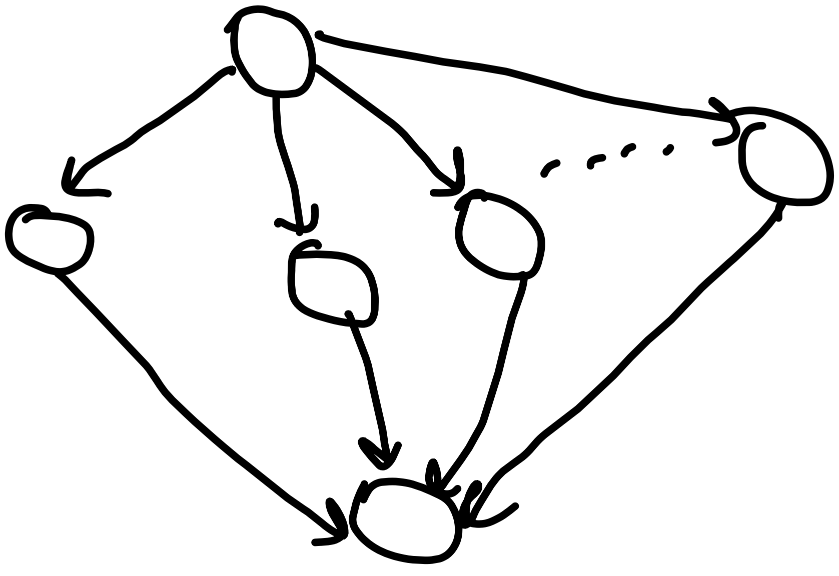
Simple if



if-else

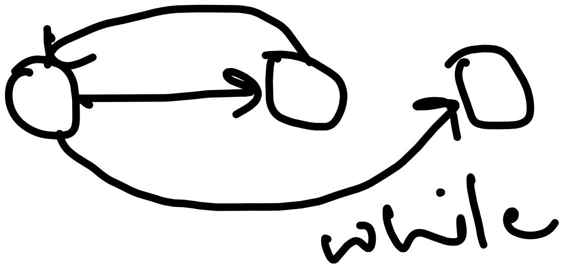


b) Multidirectional.

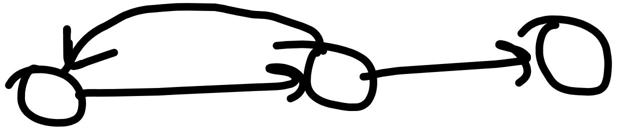


③ Repetitive statements.

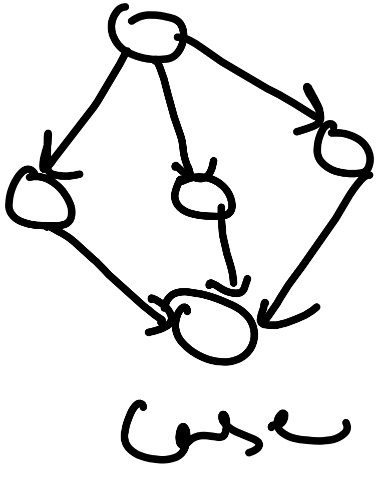
a) entry control



b) exit control



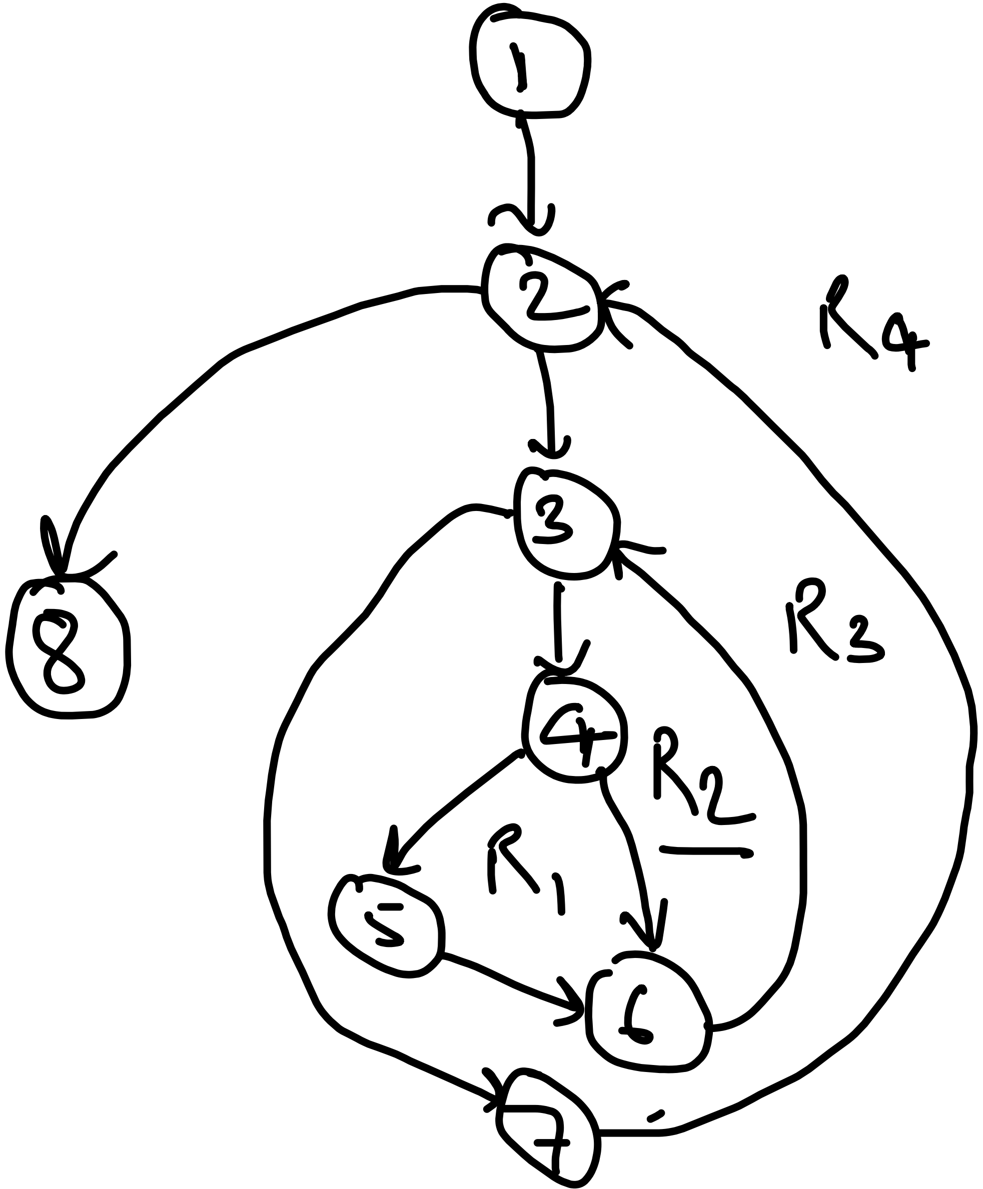
c)



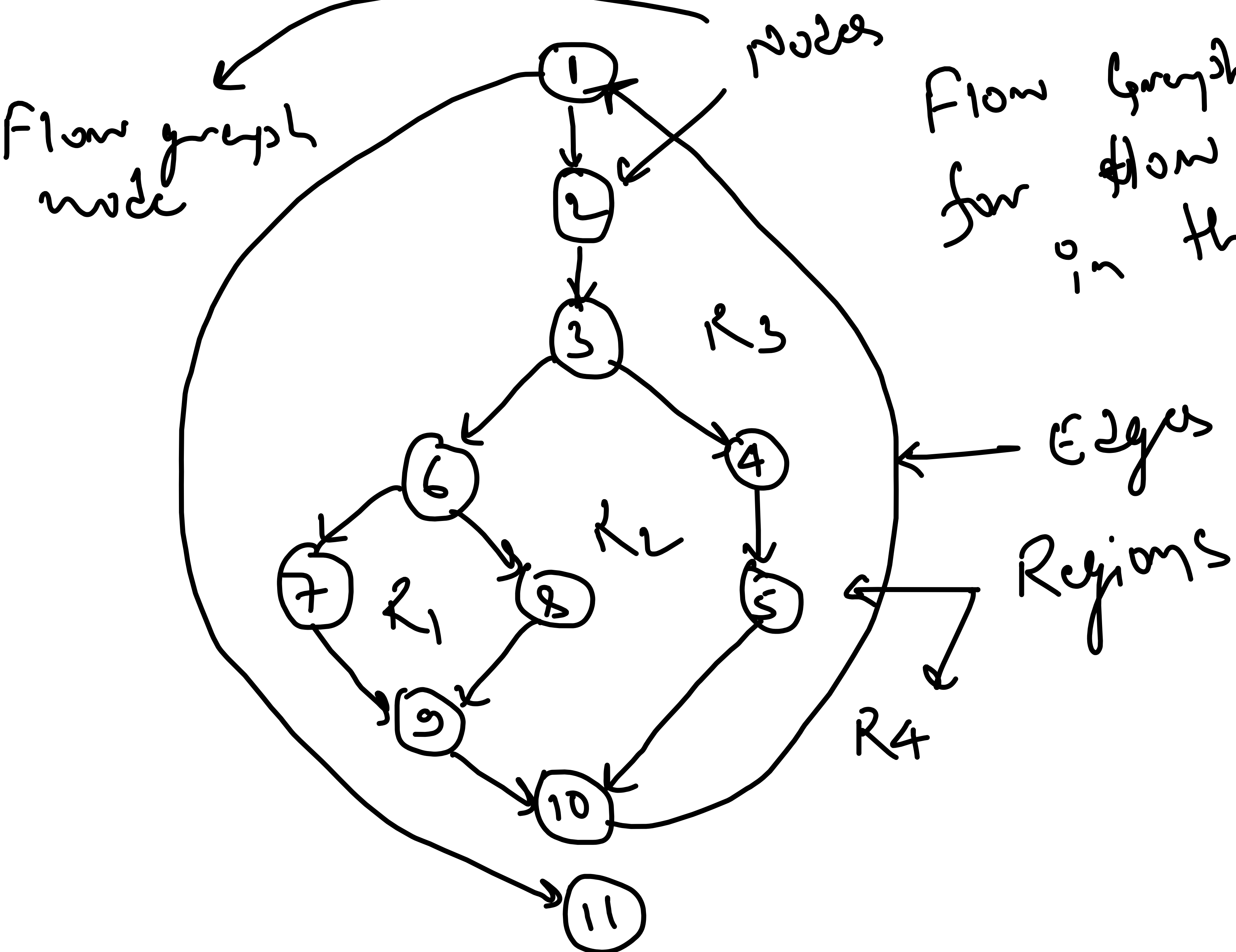
eg:- Swapping elements in an array.

```
int i, j, A[100], n;          (1)
while (i > n)                 (2)
do
    while (j > n)             (3)
    do
        if (A[i] > A[j]) then (4)
            swap(A[i], A[j]); (5)
        endif;                (6)
    end do;                   (7)
end do;                       (8)
```

Flow Graph



1. Flow graph using notations
2. Finding Regions



Flow graph for flow chart in the slide

* Flow Graph Approach.

① Construct a program for given problem

② Evaluate Cyclomatic Complexity.

③ Identify independent paths.

④ Prepare test cases based on the number of independent paths.

* CYCLOMATIC COMPLEXITY – s/w metric

① Provides quantitative measure of logical complexity of a program.

② It provides the no. of independent paths.

* Independent Paths.

Path 1 : 1-11

Path 2 : 1-2-3-4-5-10-1-11

Path 3 : 1-2-3-6-8-9-10-1-11

Path 4 : 1-2-3-6-7-9-10-1-11

① new edge is introduced

② constitute basis set of flow graph

Path : 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

not independent

Cyclomatic complexity.

① Number of regions in the flow graph

② $V = E - N + 2$ (E - Edge, N - Node)

③ $V = P + 1$ (p - No. of predicates in flow graph)

where,

V — Cyclomatic complexity.

Flow graph CC

① $V = 4$

② $N = 11$, $E = 13$

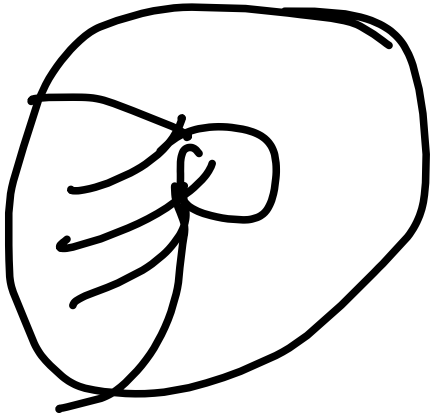
$V = 13 - 11 + 2 = 4$

③ $P = 3$

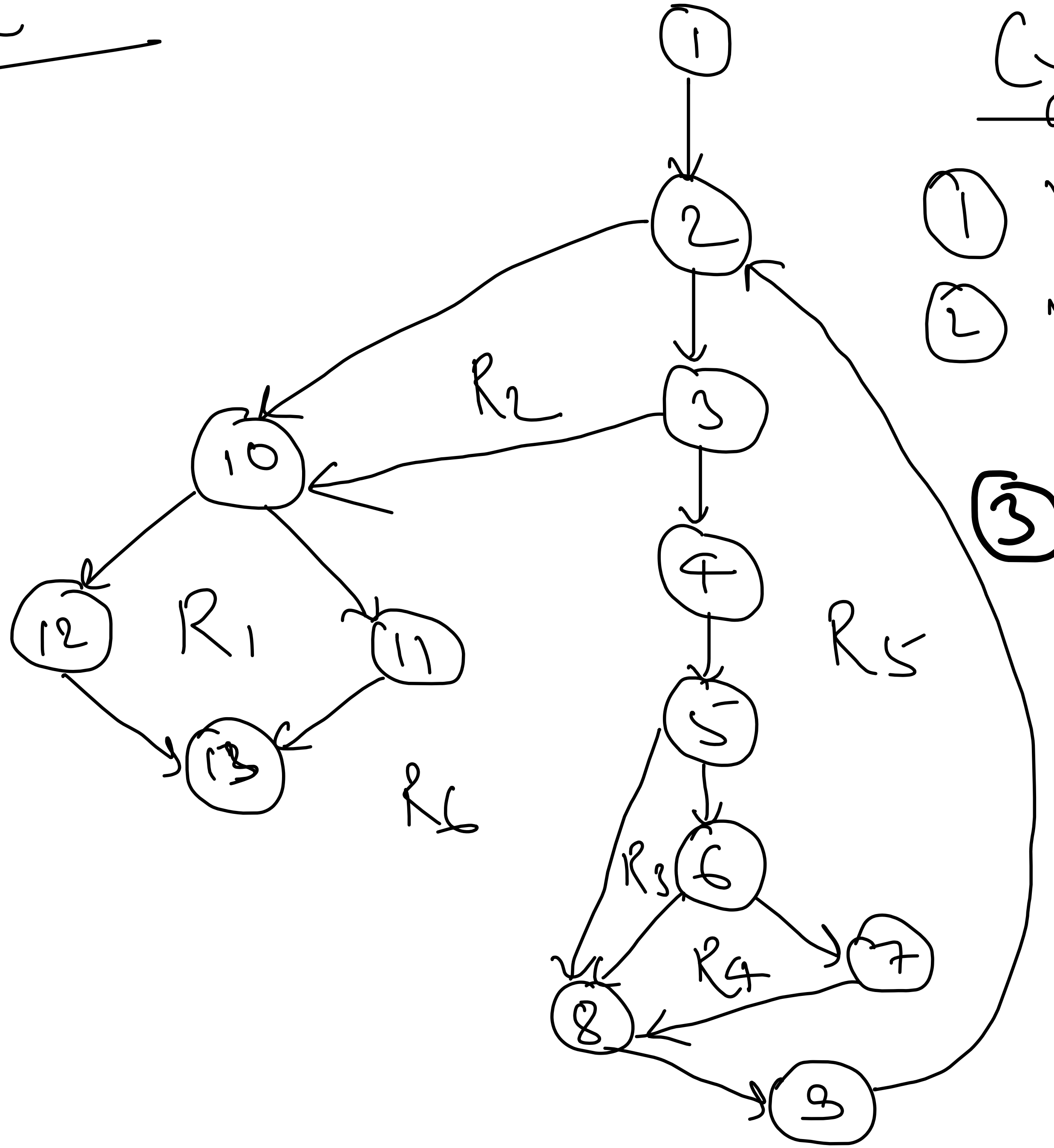
$V = 3 + 1 = 4$

* Program to find whether Δ^{le} is Scalene, isosceles, equilateral or not a triangle.

```
void check(int a, int b, int c)
{
        a=b, b=c, c==a.
            cond^n 1 —
        else a!=b, b!c, c!=a —
            cond^n 2
        else a=b || b=c || c=a
            cond^n 3. —
        else a+b <c || b+c<a || c+a < b
            cond^n 4.
}.
```



Example



Cyclomatic Complexity.

① $V = 6$ = Total no. of regions

② $V = E - N + 2$

$= 17 - 13 + 2 = 6$

③ $V = P + 1$

$= 5 + 1 = 6$

* Independent Paths.

Path 1 : 1-2-10-11-13

Path 2 : 1-2-10-12-13.

Path 3 : 1-2-3-10-11-13

Path 4 : 1-2-3-4-5-8-9-2-....

Path 5 : 1-2-3-4-5-6-8-9-2 ....

Path 6 : 1-2-3-4-5-6-7-8-9-2 ....

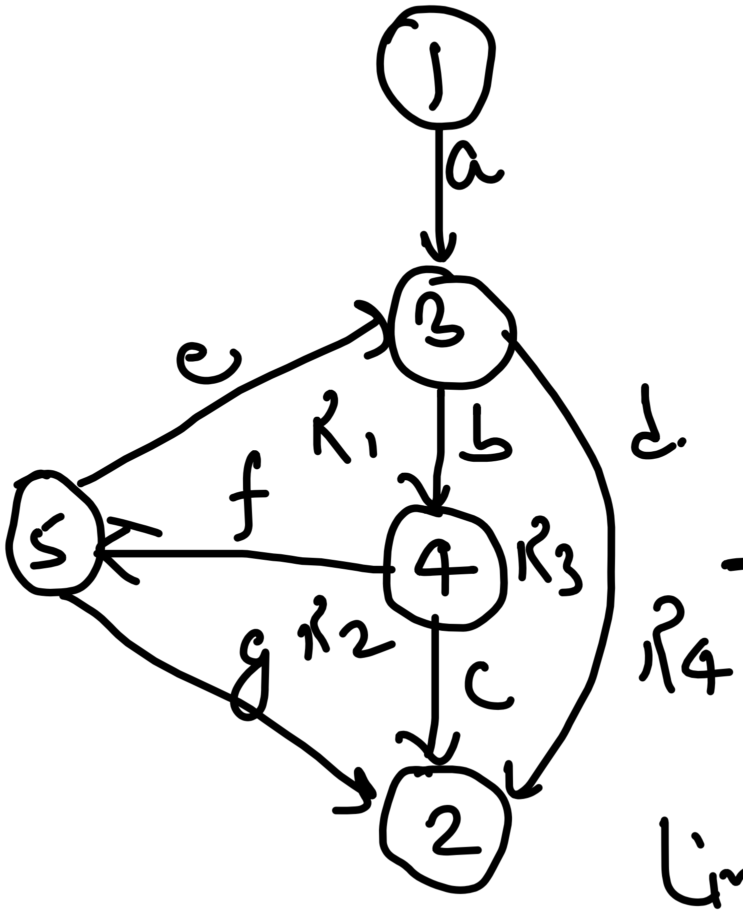
(...) — those nodes can be included in the path

GRAPH MATRICES.

Beizer. — data structure for flow graph

1. A square matrix.
2. Size of matrix = No. of nodes in the flow graph.
3. Rows & columns → nodes.
4. Entries in matrix → connections (edges).

Flow graph.



Graph Matrix.

Node \ Connected to Node	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

$$\text{link weight} = \begin{cases} 0, & \underline{\text{no connection}} \\ 1, & \underline{\text{connection}} \end{cases}$$

Possible options for link weights.

1. Probability that a link is executed.
2. Processing time for traversal of link
3. Memory required for traversal.
4. Resources are required for traversal.
5. Connection / No connections.

Connection Matrix.

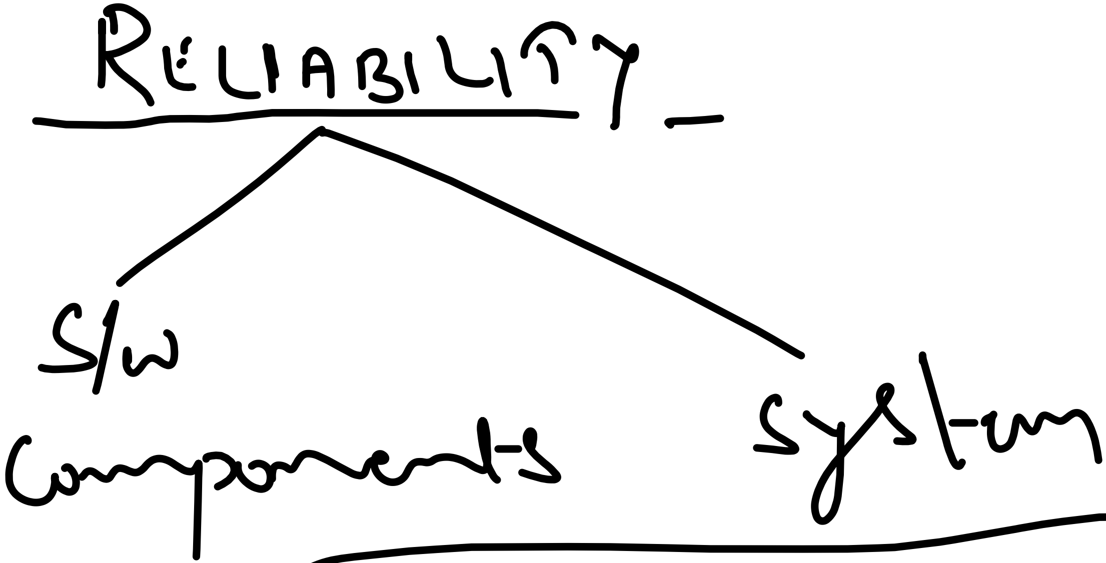
Predicate Nodes

	1	2	3	4	5	
1			1			1 − 1 = 0
2						0.
3		1		1		2 − 1 = 1
4		1			1	2 − 1 = 1
5		1	1			2 − 1 = 1

3 + 1 = 4 → Cyclomatic Complexity

① If two or more entries → Predicate node

② If no entries → Sink node



RELIABILITY

S/w Components

System

$$\text{Reliability of system} = \left(\text{Reliability of a module}\right)^{\text{No. of modules}}$$

Eg:-

① Reliability of a module = 0.999.

No. of modules = 100

Modules are put in series.

Reliability of whole s/w $= (0.999)^{100}$

$= 0.9048$

② Reliability of a module = 0.999

No. of modules = 50

Reliability of whole system = 0.999

Build, Release, Version.

Build	Release	Version
It is an executable file.	Release means which is ready to use.	Version is extension of build.
It is handed over to the tester to test the developed part of the project.	It is handed over to client/customer after completion of development & testing phases.	It is the number of release made according to the addition of requirement of the client.

Build	Release	Version
It refers to the S/w part which still in testing or not tested yet.	It is S/w which is no longer in testing.	It is variation of an earlier or original type of S/w.
) It can be rejected by test team if defect is found.	One Release can have several builds associated with it.	It is based on build but not vice-versa.
) Part of App^{n}	App^{n} itself	App^{n} itself.
Eg:- Components	Eg:- Apple has new release iphone 4.	Eg:- Download latest version of IE.

Semantic Versioning.

API - Appl
Program
Interf

X . Y . Z

- X — Major version (Major changes, breaks the API)
- Y — Minor Version (minor changes, does not break the API)
  - new features are added in a backward-compatible way.
- Z — Patches (Bug fixes)

Eg:- IDM 6.18.7

Pre-Releases — append a hyphen to the end of semantic versioning (SemVer) sequence

1.0.0 — 1.0.0-alpha.1

.2

⋮

NOTE:-

1. 1st version starts at 0.1.0 & not at 0.0.1
2. 1.0.0 $\xrightarrow{\text{before}}$ development phase is carried out.
3. 1st stable version — 1.0.0 (v1.0.0)