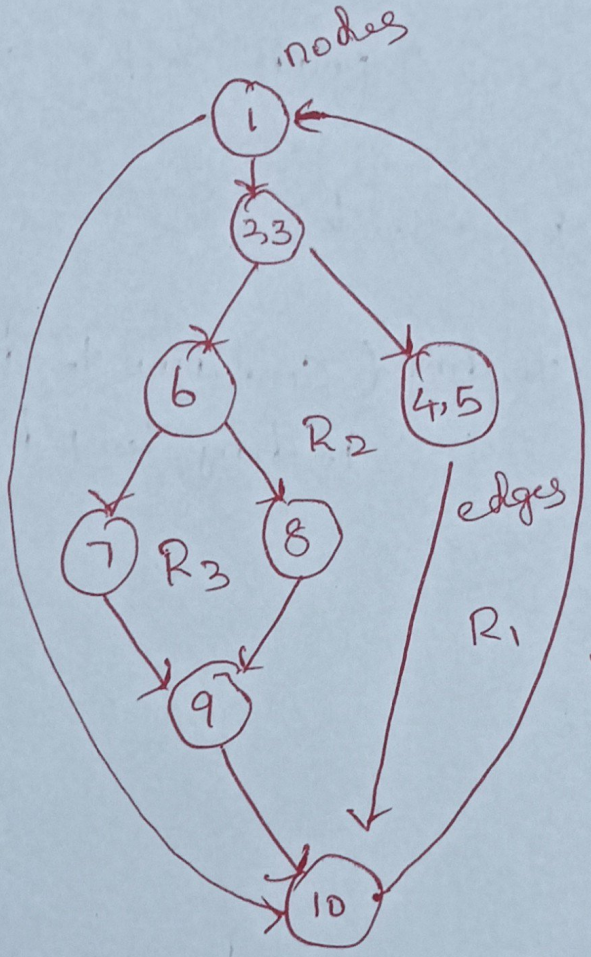
White Box testing (hints)

(1) Basis path testing is a white box testing technique to ensure that all possible path in a program are tested

↳ Control flow graph [CFG]
Graphical representation of a program flow



(2) independent program path

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

Predicate node is a node that contain a Condition or Decision Point (has two or more edges)

Cyclomatic complexity

(1) 4 Regions $(R_1, R_2, R_3, R_4) = 4$

(2) $V(G) = E - N + 2 = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4$

(3) $V(G) = P + 1 = 3 \text{ predicate nodes } 3 + 1 = 4$

here (1, (2, 3), 6) are predicate nodes

(3) Deriving test Case (4 points Refer book)

(4) graph matrices (Refer book)

(5) Control Structure testing (Condition testing, Data flow testing, loop testing)

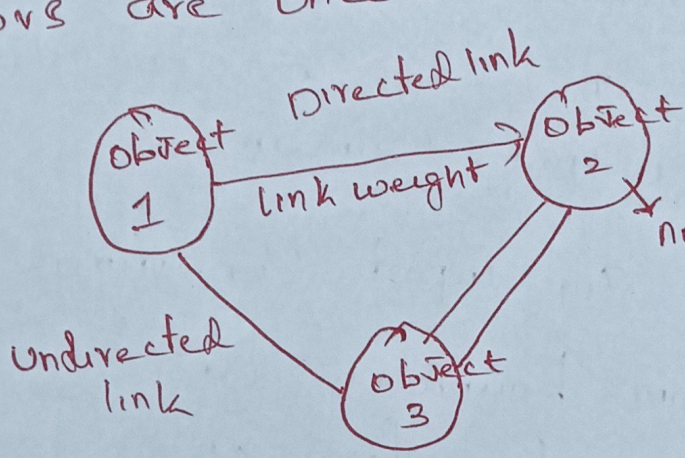
[So 5 topics]

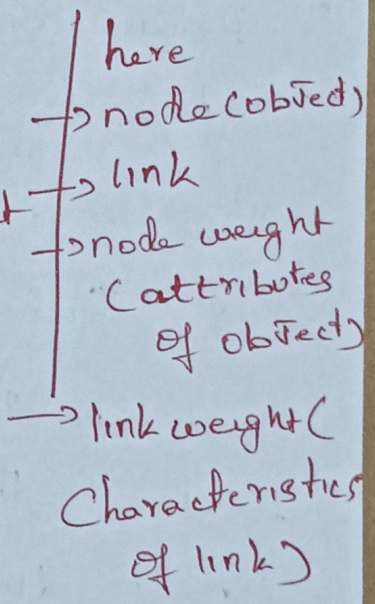
Black Box testing hints

also called behavioral testing focuses on the functional Requirements of the S/w

(1) graph based testing method

by creating a graph of important obj -ects and their relationship and then devising a series of test that will cover the graph so that each object and relationship is exercised & errors are uncovered





(2) Equivalence partitioning

Key principles

↳ each equivalence class should represent a group of input that the program processes

Similarly

↳ testing one representative from a class should be sufficient to assume Correctne -ss for all other values in that class

↳ Both valid & invalid equivalence classes should be Considered

(e.g) (1 to 100)

One valid equivalence class: Value within Range (1 to 100)

Two invalid equivalence class:

Value below the range (<1)

& above the range (>100)

(3) Boundary Value Analysis (BVA)

focuses on testing the edges or boundary values of input & output Domain

(e.g) Range [a, b] For input Conditions

Test at the minimum boundary (a)

test at the maximum boundary (b)

test Just below & Just above these boundaries

(a-1, b+1)

(e.g) Age validation for a System (valid range: 18 to 60)

Valid values : 18, 60

invalid values : 17 (below), 61 (above)

(4) orthogonal array testing

is a ~~combinatorial~~ black box testing techniques that helps reduce the number of test cases while maintaining effective test Coverage

(4) Th ques hints

Metric for the Requirement model ↓ [low value, medium, high]

- L_{ei} number of external input [3, 4, 6]
- L_{eo} number of external output [4, 5, 7]
- L_{eq} number of external inquiries [3, 4, 6]
- L_{ILF} number of internal logical files [7, 10, 15]
- L_{EIF} number of external interface files [5, 7, 10]

above 5 information Domain Value ranges from low, medium, high

Information Domain	Count		Simple	avg	com	
[Ei]	2 ×	(3)	4	6	=	6
[eo]	2 ×	(4)	5	7	=	8
[eq]	2 ×	(3)	4	6	=	6
[ILF]	2 ×	(7)	10	15	=	14
[EIF]	2 ×	(5)	7	10	=	10

Count total = 44

General Formula

$$Fp = \text{Count total} \times [0.65 \times 0.01 \times \Sigma(Fi)]$$

for $\Sigma(Fi)$ = you want to answer 14 question
answer value is from [0 to 5]

if 1 is answer for all 14 question

= 1+1+1+1+1+1+1+1+1+1+1+1+1+1

$\Sigma(Fi) = 14$ → Sub this value in Fp Formula

Refer Book for 14 questions

(c) Factor — level

P_1 — (1, 2, 3)

P_2 — (1, 2, 3)

P_3 — (1, 2, 3)

So there are $3^3 = 27$ possible test case

↳ But orthogonal array has a balancing property

Only 9 test case

↳ each factor is tested only once in every possible combination

↳ without missing important Combination, all pair wise interaction (between factor) are tested

(3) rd question

test strategies for Conventional Software

↳ unit testing

↳ integration testing

↳ Regression testing

↳ Smoke testing

5th question hints

metric for the Design model

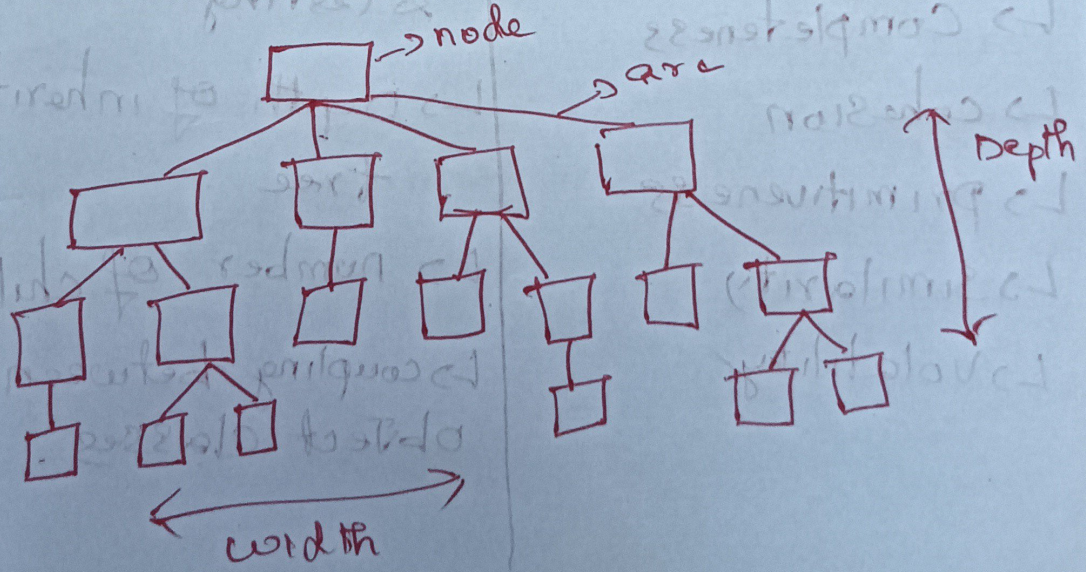
① Architectural Design metric

Structural Complexity: $S(i) = F_{out}^2(i)$

Data complexity: $D(i) = \frac{V(i)}{F_{out}(i)+1}$

V(i) = no of i/p & o/p variable Passed to & From module i,

System complexity = S(i) + D(i)



Size = n + a n = node/a=arc

~~width~~ 18 + 17

= 35

depth = 4

width = 6

(2) metrics for OOD

↳ Size

↳ complexity

↳ Coupling

↳ Sufficiency

↳ Completeness

↳ cohesion

↳ primitiveness

↳ similarity

↳ Volatility

(3) Class oriented metrics

↳ weighted methods per class

measures complexity & effort required for implementation & testing

↳ Depth of inheritance tree

↳ number of children

↳ coupling between object classes

- Response for a class
- Lack of Cohesion in methods

4. Lorenz & Kidd's classification of OO metric
   - Size metric
   - inheritance metric
   - internal metric
   - External metric

5. Component level Design metric
   - Cohesion metric
   - coupling metric

6th question

metric for Source Code

n_1 = number of Distinct operator that appear in a program

n_2 = number of Distinct operand that appear in a Program

N_1 = total number of operator occurences

N_2 = total nomber of operand occurences

length $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$

Program volume $V = N \log_2 (n_1 + n_2)$

metric for testing

Program level PL,

$$PL = \frac{1}{(n_1/2) \times (N_2/n_2)}$$

$$\text{effort } (e) = \frac{V}{PL}$$

testing effort for k module $(k) = \frac{e(k)}{\Sigma e(i)}$

$\Sigma e(i)$ = effort across all modules of the system

metric for object oriented testing
- Lack of Cohesion in methods
- Percent Public and protected
- public access to data member
- number of root Classes
- Fan in

Metric for maintanence:

M_T = number of modules in the Current release

F_c = number of modules in the current release that have been changed

F_a = number of modules in the current release that have been added

F_d = number of modules from the Pro-Coding release that were deleted in the Current release

Software maturity index

$$SMI = \frac{M_T - (F_a + F_c + F_d)}{M_T}$$