1. Decision Table Testing:

③ rule combine all the i/P condition→o/P

no.of IP	condition state	T- True F- False J- immaterial state/Don't care
no.of oP	Action state	'x' - o/P

1. A shopowner wants to sale is compny with different discount rate based upon the following condition.

i) If the order is placed by $\underset{IP}{\underline{ABC}}$ company irrespective of order value Give $\underset{o/p}{\underline{10\%}}$ of discount

ii) If order value is $>\underset{IP}{\underline{50,000}}$ and order is given by an $\underset{IP}{\underline{agent}}$ that give $\underset{o/P}{\underline{15\%}}$ discount. If order was given by $\underset{IP}{\underline{retailer}}$ give $\underset{o/P}{\underline{12\%}}$ discount

iii) If order value is >20,000 and <50000 order is given by agent then give 12% discount If order is given by retailer give 10% discount

iv) If order value is <20000 and order is given by agent the give 8% of discount If order is given by retailer give 5%.

v) If order is furniture irrespective of all give 10% discount.

Apply 3condition:

step1: i/p and o/p conditions

a, ABC – 10% dis

b, O>50000 Agent-15%

Retailer=12%

c, O>20000 A – 12%
O<50000 R – 10%

d, O<2000 A – 8%

R – 5%

e, O= furniture 10% dis

step2: Decision Table

		R1	R2	R3	R4	R5	R6	R7	R8
C-S	C1: ABC	T	F	F	F	F	F	F	F
	C2: Agent	F	T	F	T	F	T	F	F
	C3: Retailer	F	F	T	F	T	F	T	F
	C4: O>50000	I	T	T	F	F	F	F	F
	C5: O>20000 O<50000	I	F	F	T	T	F	F	F
	C6: O<20000	I	F	F	F	F	T	T	F
	C7: O: furniture	F	F	F	F	F	F	F	T
A-S	A1: 10% Dis	X	–	–	–	X	–	–	X
	A2: 15%	–	X	–	–	–	–	–	–
	A3: 12%	–	–	X	X	–	–	–	–
	A4: 8%	–	–	–	–	–	X	–	–
	A5: 5%	–	–	–	–	–	–	X	–

Step: 3

Test case Table → No. of TCS = No. of Rules = 8TCS

TC ID	TEST I/P			ER	AR	states
	Type of customer	order value	furniture			
1.	ABC	>write original 1000	NO	10% Dis	10% Dis	pass.
2.	Agent	55000	NO	15%	10%	Failure
3.	retailer	1000	NO	12%	12%	pass
4.	Agent	22000	NO	12%	10%	Failure
5.	Retailer	45000	NO	10%	10%	pass
6.	agent	15000	NO	8%	8%	pass
7.	Retailer	2000	NO	5%	5%	pass
8.	agent	1000	yes	10%	6%	failure

cause and Effect (C&E) graph based Testing:

steps of C&E graph:

1. Read and understand the scenario, List down the input (cause) and output (effect) condition in it.

2. Draw cause and effect graph

3. create decision table

4. create test case table based decision table data.

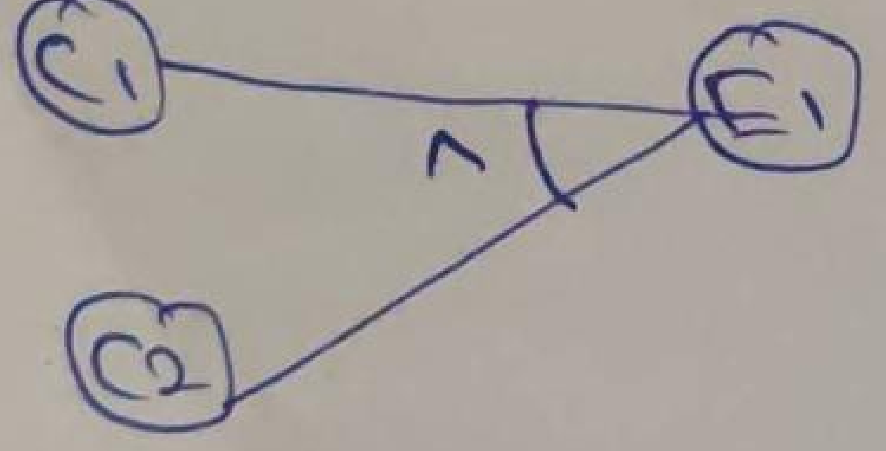
notations:

Ⓒ ⇒ cause C1, C2 . . . C-n

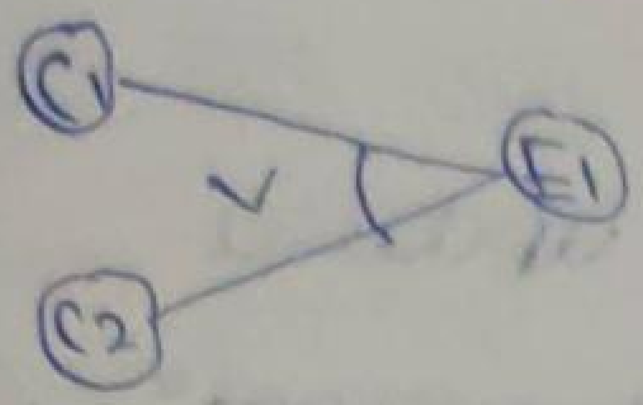
Ⓔ ⇒ Effect E1, E2 . . . E-n

notations: ∧

AND- (for Effect E) to be time, both the causes C1 and C2 should be true

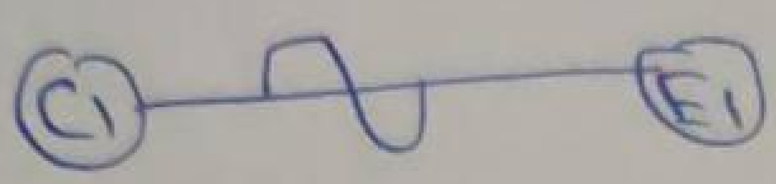


Or: V   For effect E1 to be true, either of causes C1 or C2 should be true



ⓧ Anyone

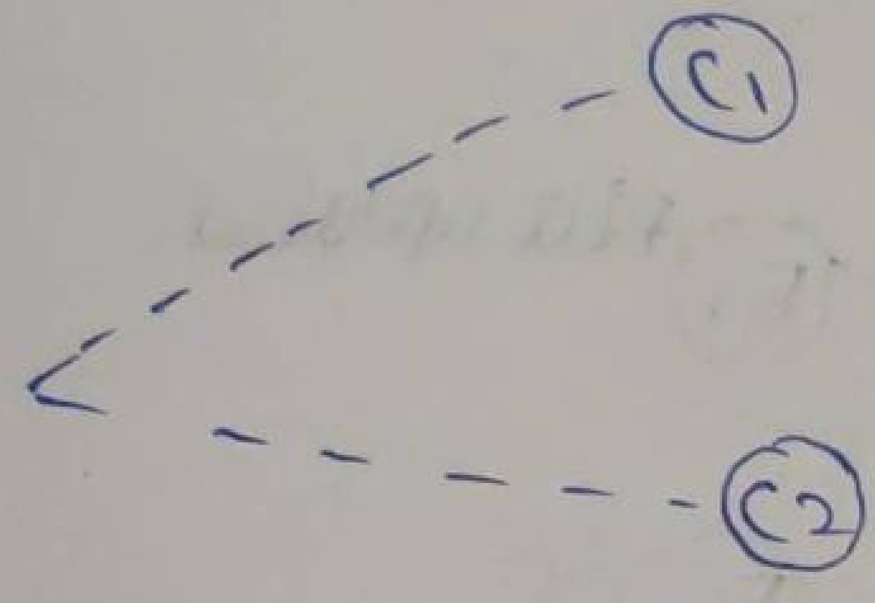
not:



ⓧ NO → yes

yes → NO

Mutually exclusive   when only one of the causes will hold true



Ex:

The "Print message" is software that read two characters and depending of their values, messages must be printed.

- the first character must be an "A" or a "B"
- The second character must be a digit.
- If the first character is an "A" or "B" and the second character is a digit, then the program will print the message file must be updated.
- If the first character is incorrect (not an "A" or "B"), the message X must be printed
- If the second character is incorrect (not a digit) the message Y must be printed.

step1:

I/P

C1- First character is A

C2 - " " " B

C3 - second " " Digit

O/P

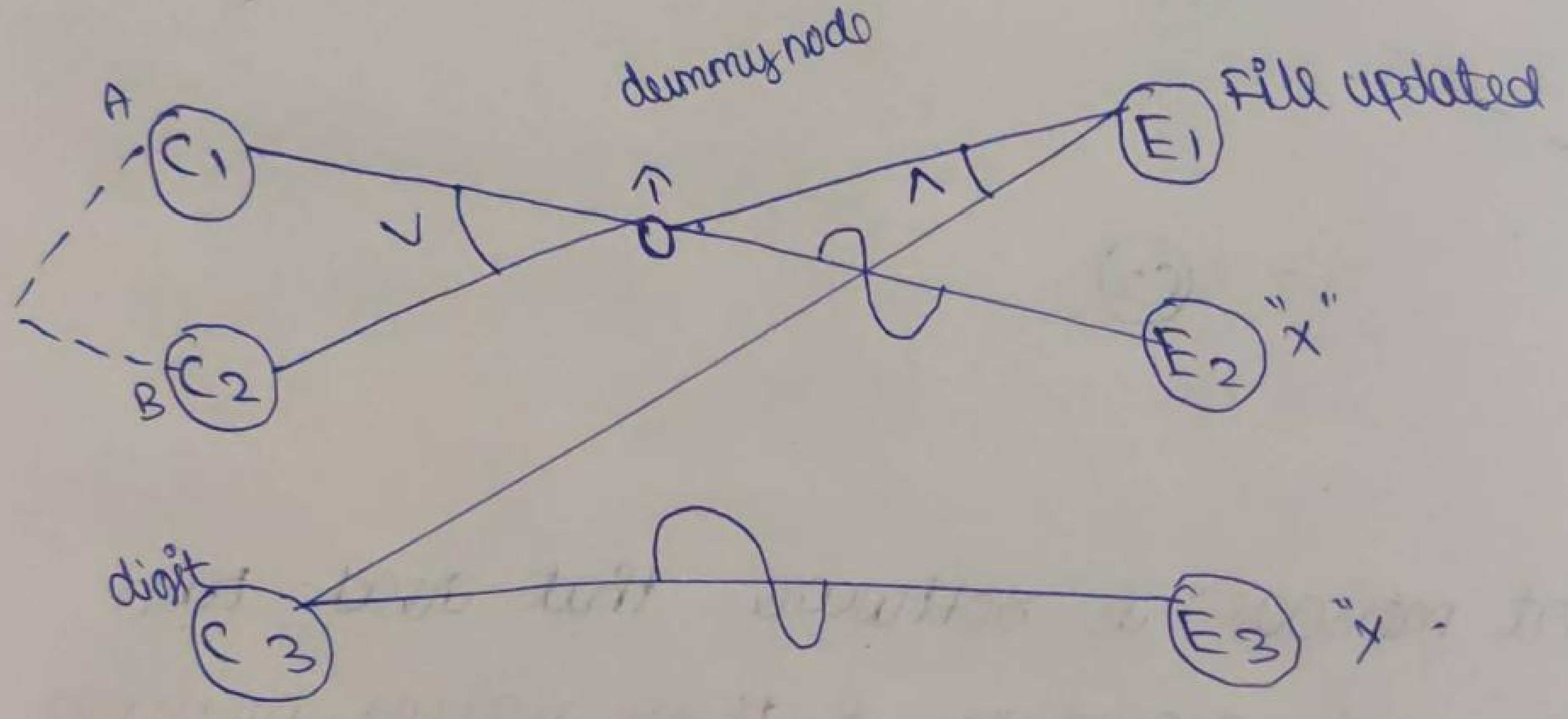
E1- file updated

E2- print message "x"

E3 - print message "y"

step2:

Draw cause and Effect Graph



step3: Create Desision Table

		Rule			
		R1	R2	R3	R4
condition stub	C1: A	T	F	F	I
	C2: B	F	T	F	I
	C3: digit	T	T	I	F
Action stub					
	A1: File updated	X	X	–	–
	A2: X	–	–	X	–
	A3: Y	–	–	–	X

step4:

Create Test case Table:

TCID	T I/P		ER	AR	Status
	1st char	2nd char			
1.	A	5	File updated	File update	Pass
2.	B	7	File updated	File update	Pass
3.	C	7	X	X	Pass
4.	A	$/b	Y	Y	Pass

Scanned by TapScanner

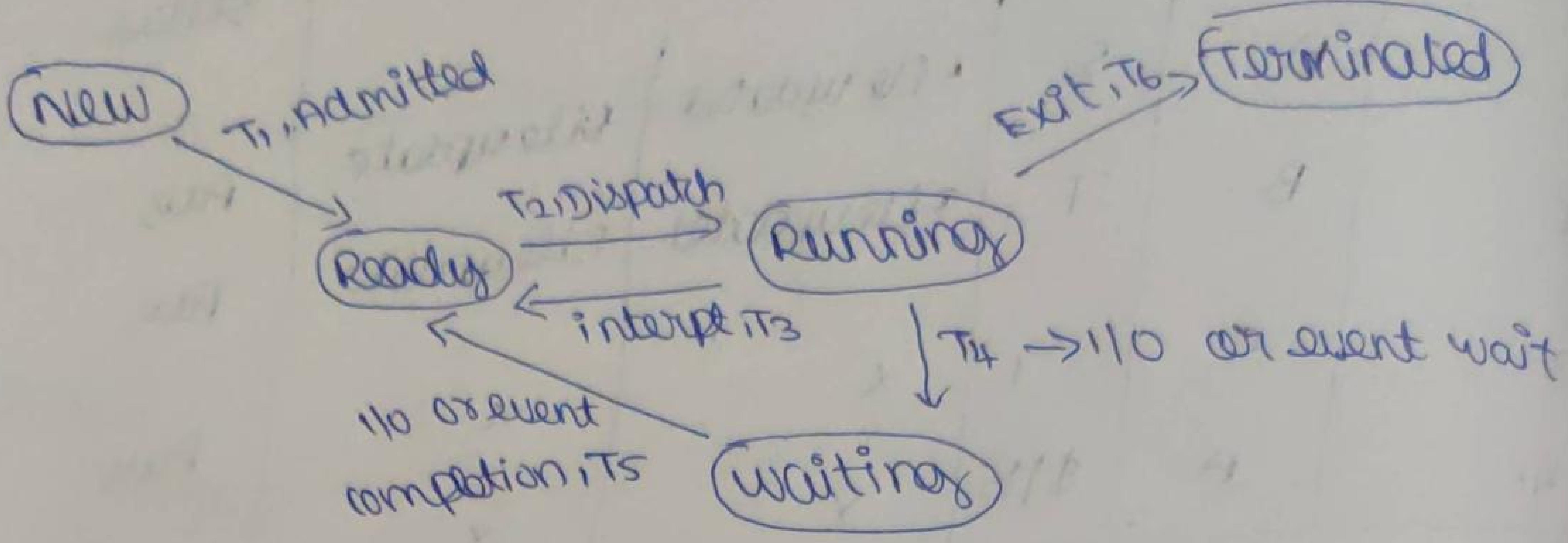
State Table Based Testing.

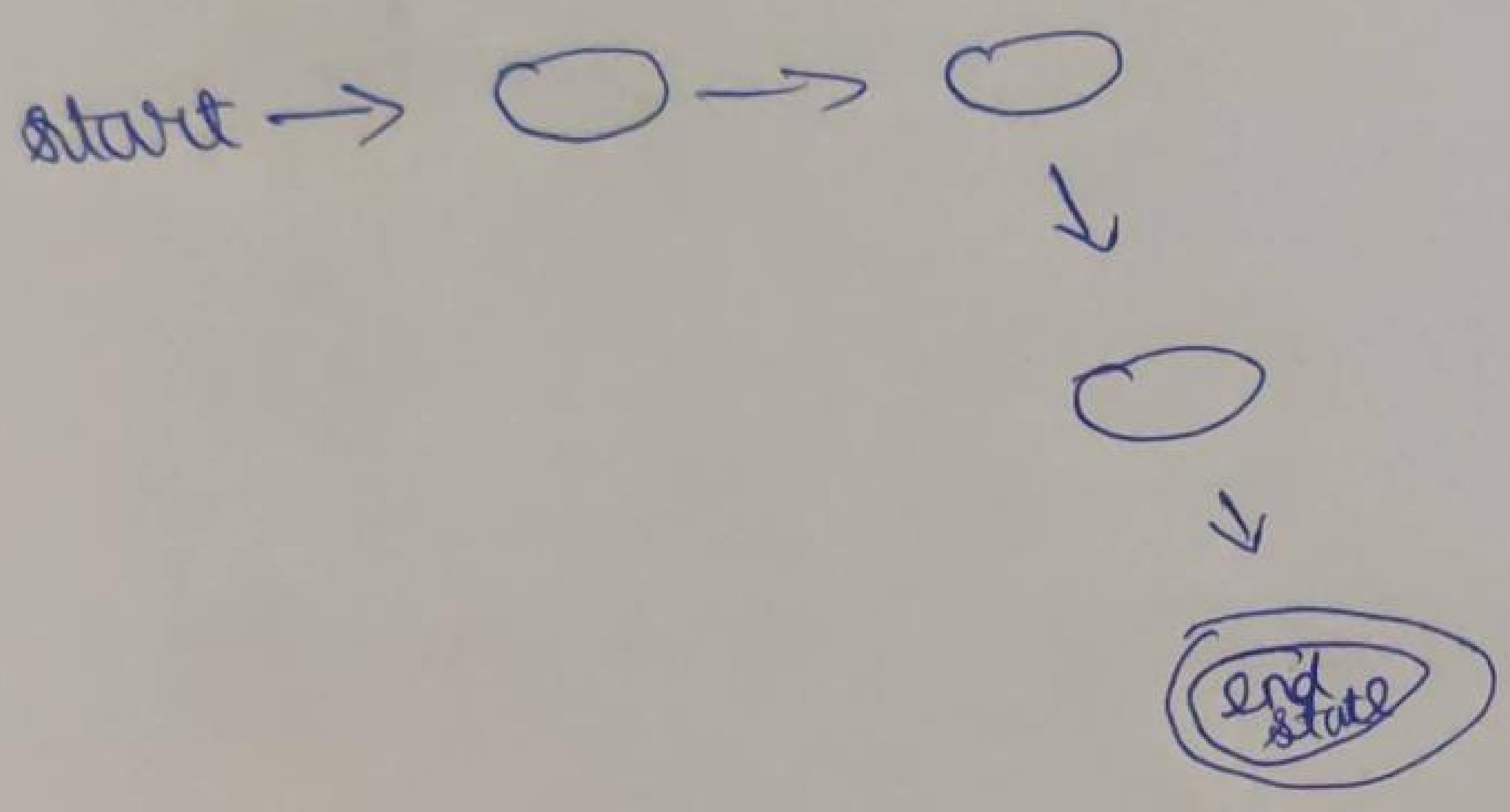
○ → Nodes

→ Transaction state

Finite State Machine (FSM)

State Transition Diagram or State Graph:





Conditions.

Step1: Read nd understand the secenario

Step2: Draw Finite State machine Diagram

Step3: Create State Table

Step4: Test case

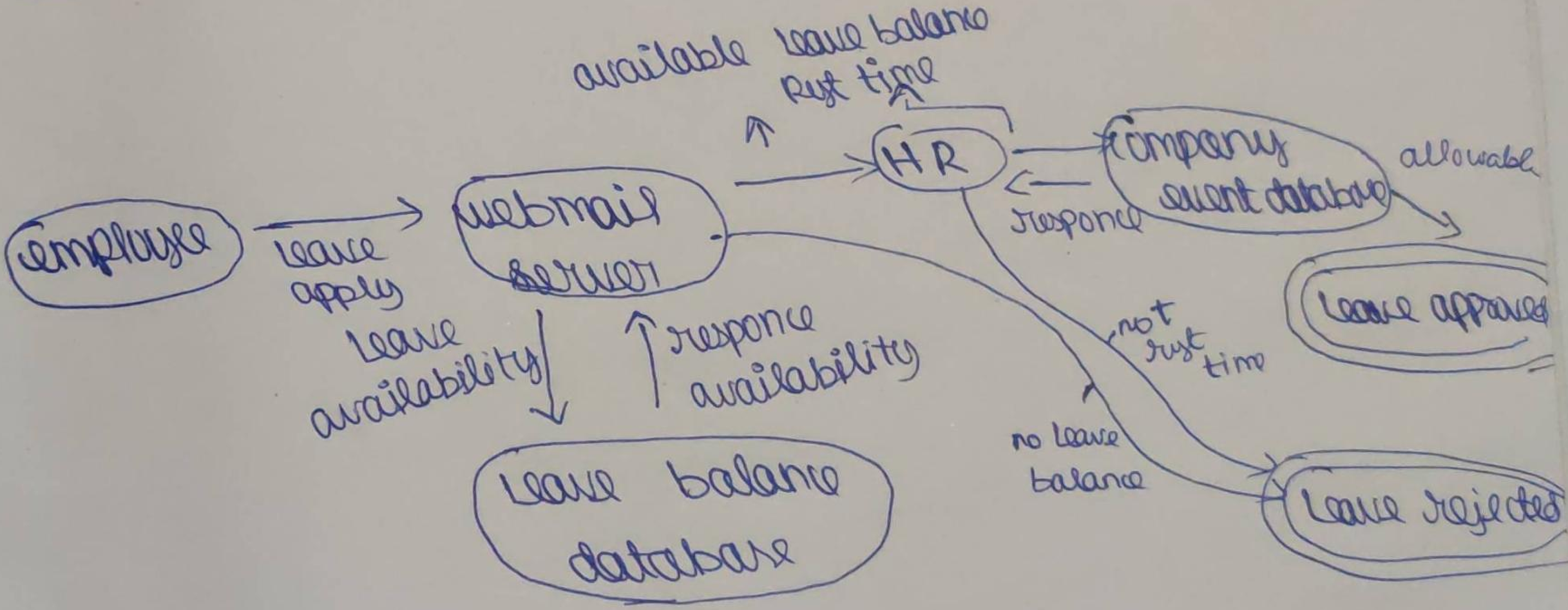
no. of state × no. of Transaction

○ →

4 6

so an employee in an organisation is going to apply leave through web mail server That leave request cross verified against employee leave balance database. once it confirm that the request will be pass to human resource of the company. HR will be verify whether this is Right time to approve the leave or not by means of cross verifying company event database if there is no issue then HR will Approves.

machine diagram:



T x 9

Module-3

White Box Testing:

develop doing -> white Box Testing

different Techniques:

1. state coverage Testing
2. Logical coverage Testing
3. conditional or multiconditional coverage Testing } simple
4. Loop based Testing
5. Basis path Based Testing
6. Dataflow Testing
7. Mutation Testing -> error guessing.

1. statement coverage Testing:

```
1. main()
2. {
3.  int x, y;
4.  cin>>x;
5.  cin>>y;
6.  while (x!=y)
7.  {
8.  if (x>y)
9.  cout <<" x is Big";
10. else
11. cout <<"y is Big";
12. }
13. }
```

ex: checking:

```
1.
2
3
4
5
6 while (5!=5)
→①
1.
2.
3.
4.
5.
6. while (5!=10)
7. {
8. if (5>10) →false → Jump to line no 10
10. else
11.
12.
13.
→②
1.
2.
3.
4.
5.
6. wile (10!=5)
7. {
8. if (10>5) → true → gogo line no 9 → then 12,13
9.
12.
13.
→③
```

I want to cover each nd every li no of the program is called statement coverage

1. x=n , y=n   n=5

ER:

1,2,3,4,5,6 ,13

2. x=n, y=n

1,2,3,4,5,6,7,8,10,11,12,13

3. x=n, y=n

1,2,3,4,5,6,7,8,9,12,13

x=5 , y=10

-> cover all the statement

x=10, y=5

2. Logical coverage Testing.

Two direction

one is true another one is false

TCID	T I/P		ER
	x	y	
1.	5	10	y is Big
2.	10	2	x is Big

Logic is Line no: 8

3. conditional Testing.

checking condition statement is Line no:6 nd 8

TCID	T I/P		ER
	x	y	
1.	5	10	while block (6) go to line 7 else line 13
2	10	2	if condition block true -> Line no 9 Lfalse -> line no 10

multiconditional Testing: →more than one condition

Back program code is not suitable to multiconditional.
① ②

8. if (x>y && x!=0) $2^2=4$

TC ID	T I/P		ER
	x>y	x!=0	Line no:
1	T	T	9
2	T	F	10
3	F	T	10
4	F	F	10

True means go to Line no:9 or false means go to Lino no:10

4. Loop Based Testing:

```
while (x!=y)
{

}
```

Tc id	T I/P	ER
	x!=y	
1.	True	Block-entry
2.	false	Block-exist

5 path- Basis Testing

step1 : Develop the program & line no <statement

step2 : Draw D-D Graph

step3 : calculate cyclometric complexity

step4 : Find independent paths in the given program

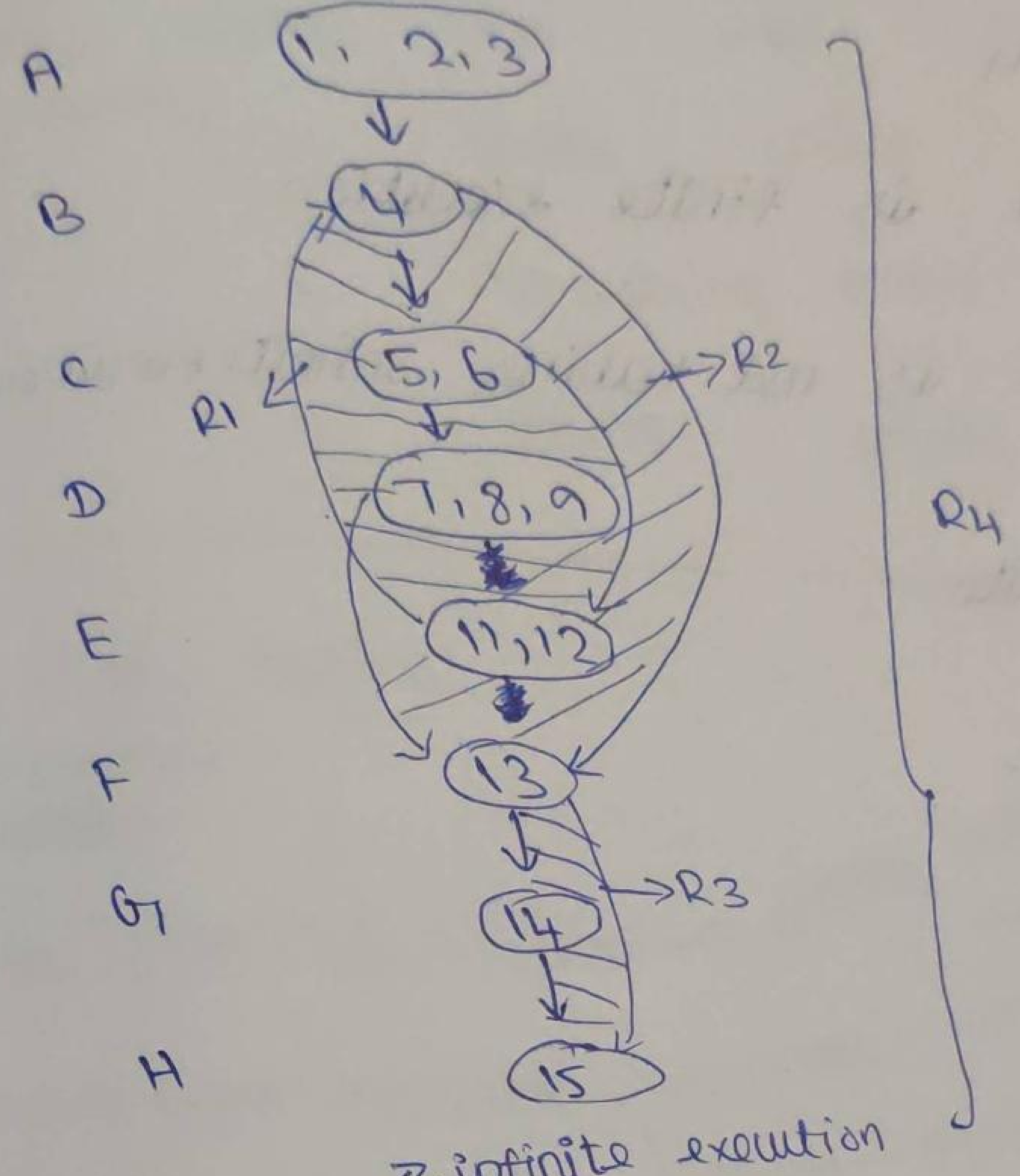
step5 : Remove invalid paths

step6 : create TC Table

Example : step1 :

```
1. printf ("Enter a number");
2. scanf ("%d, &number);
3. index=2;
4 while (index <=number-1)
 5. {
 6. if(number%index==0)
 7. {
 8. printf ("not a prime number");
9. break;
10. }
11. index++;
12. }
13 if (index== number)
14. printf (" prime number ");
15. }
```

step2: D-D Graph.



step3:- cyclometric complexites:

1. $v(G) = e - n + 2*P$ → node, ↳ predicate node, ↓ edge

$e \rightarrow$ noof

$n \rightarrow \bigcirc$

$= 10 - 8 + 2 * 1$

$= 4$

$d \rightarrow$ statemen

② $v(G) = d + 1$

$= 3 + 1$

$= 4$

③ $v(G) =$ n(regions)

$= n$ (R1, R2, R3, R4)

$= 4$

① ② ③

4 = 4 = 4

=> This program is finite execution

(or)

This program is not having infinite execution

ex:

4 ≠ 3 ≠ 4 -> infinite

④ Independent path:

A -> H

1. A-B-F-G-H
2. A-B-F-H
3. A-B-C-D-F-G-H
4. A-B-C-D-F-H
5. A-B-C-E-B-F-G-H
6. A-B-C-E-B-F-H

5. Remove invalid path

A-H

1. A-B-F-G-H ✓
2. ~~A-B-F-H~~ ->invalid variable
3. ~~A-B-C-D-F-G-H~~ Iv
4. A-B-C-D-F-H ✓
5. A-B-C-E-B-F-G-H ✓
6. ~~A-B-C-E-B-F-H~~

only one will come not both

8. In code

8. printf(not a prime)->D

12. printf(Prime)->G

check G and D

6- TC Table:

TCID	TI/P	ER	AR	status	Path coverage
1	Prime no < 2	prime	prime	pass	ABFGIH
2	4	not a prime	prime	fail	ABCDFH
3	3	prime	prime	pass	ABCEBFGIH

✓ valid

Data flow Testing:
↓ variable:
3 - states

1- Define (D/d) / initialize → getting value of the variable → which line no

2- usage (U/u) variable
- c-use - computational
- p-use - predicate

3- Kill (K/k) → last line → end of the code

Ex:

1. main()
2. {
3. int a,b;
4. char c;
5. cin>>a>>b; → define line no → (1)
6. a=a+10; → use
7. b = a ; → use
8.
9.
10. count<<a<<b
    use
11. } → Kill state

Killstate - free (a);

steps. Data Flow Testing:

1- Create a program and assign a program for every statement in a program.

2. Draw d-d graph

3. convert d-d graph into control flow graph

~~4- convert d-d graph~~

4- Create anamoly table each variable.

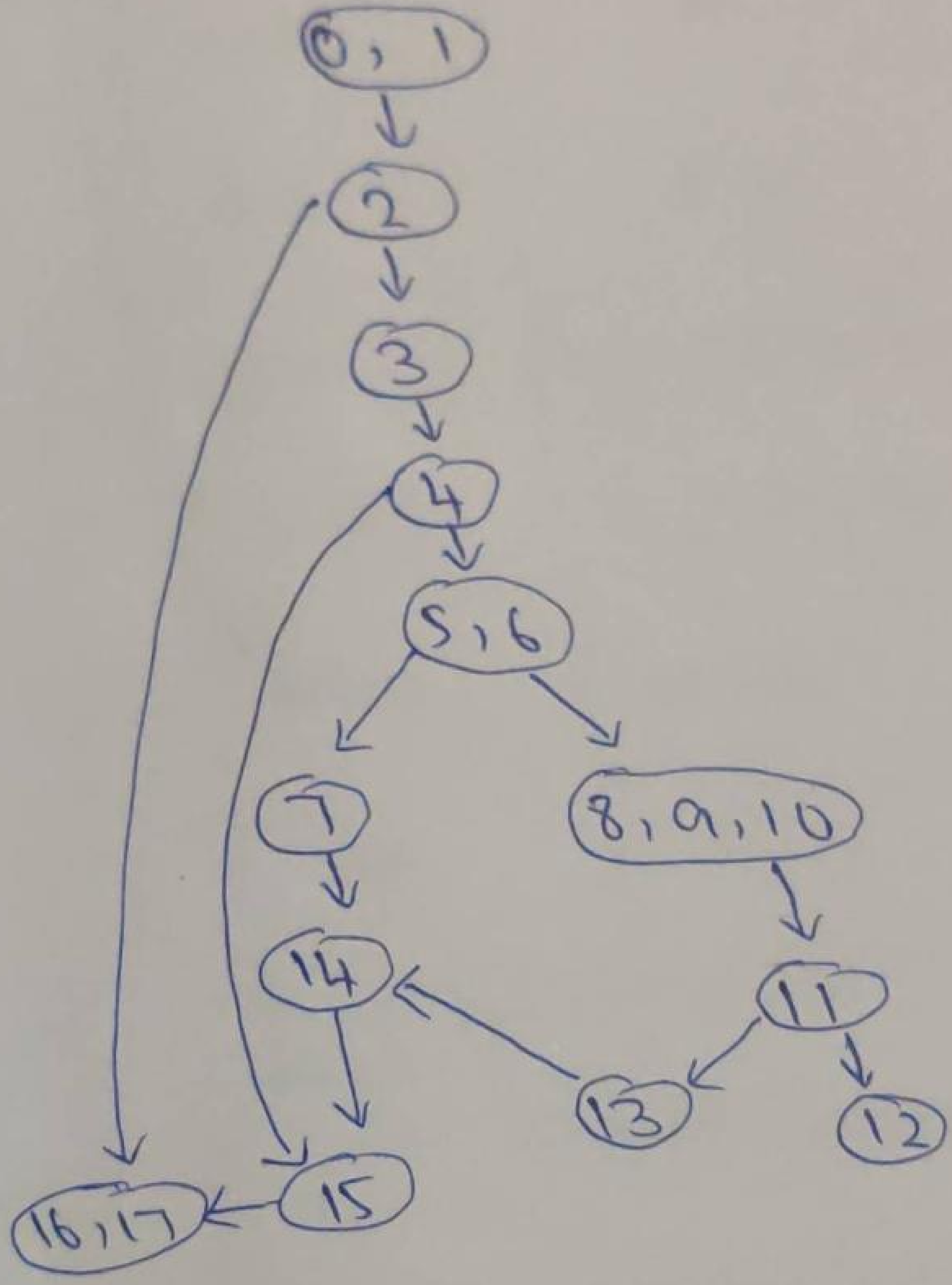
step1:

```
    main()
    {
    int work;
0- double payment =0;
1. scanf("%d",work);
2- if(work>0) {
3. payment =40;
4. if(work >20)
5- {
6- if (work <=30)
7- payment =payment + (work-25)* 0.5;
8- else
9- {
10- payment =payment + 50+(work-30)*0.1;
11. if (payment >=300 )
12. payment =payment *0.9;
13. }
14 }
```

15- }

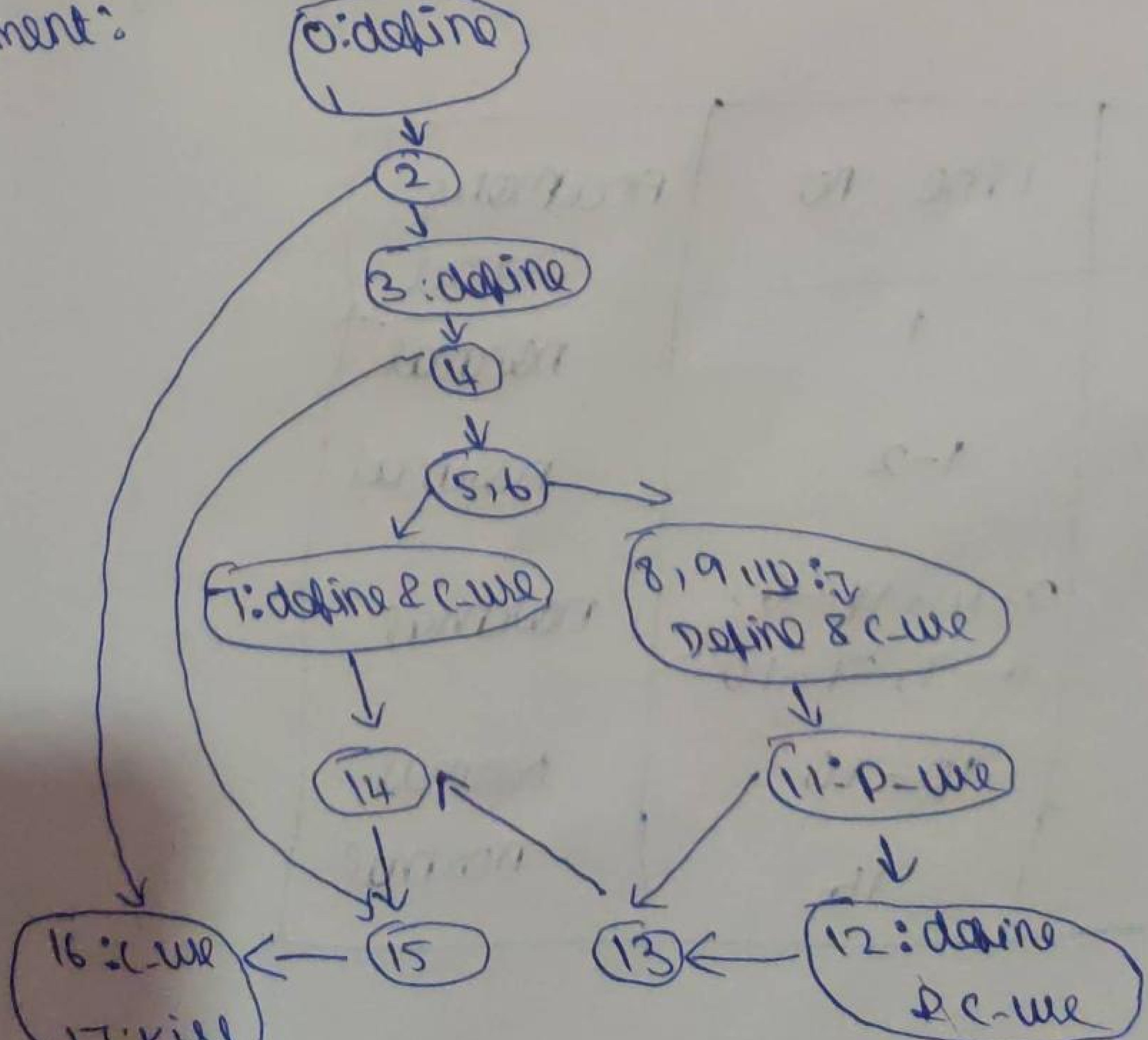
16- printf (" Final payment", payment),

step2: d-d Graph:

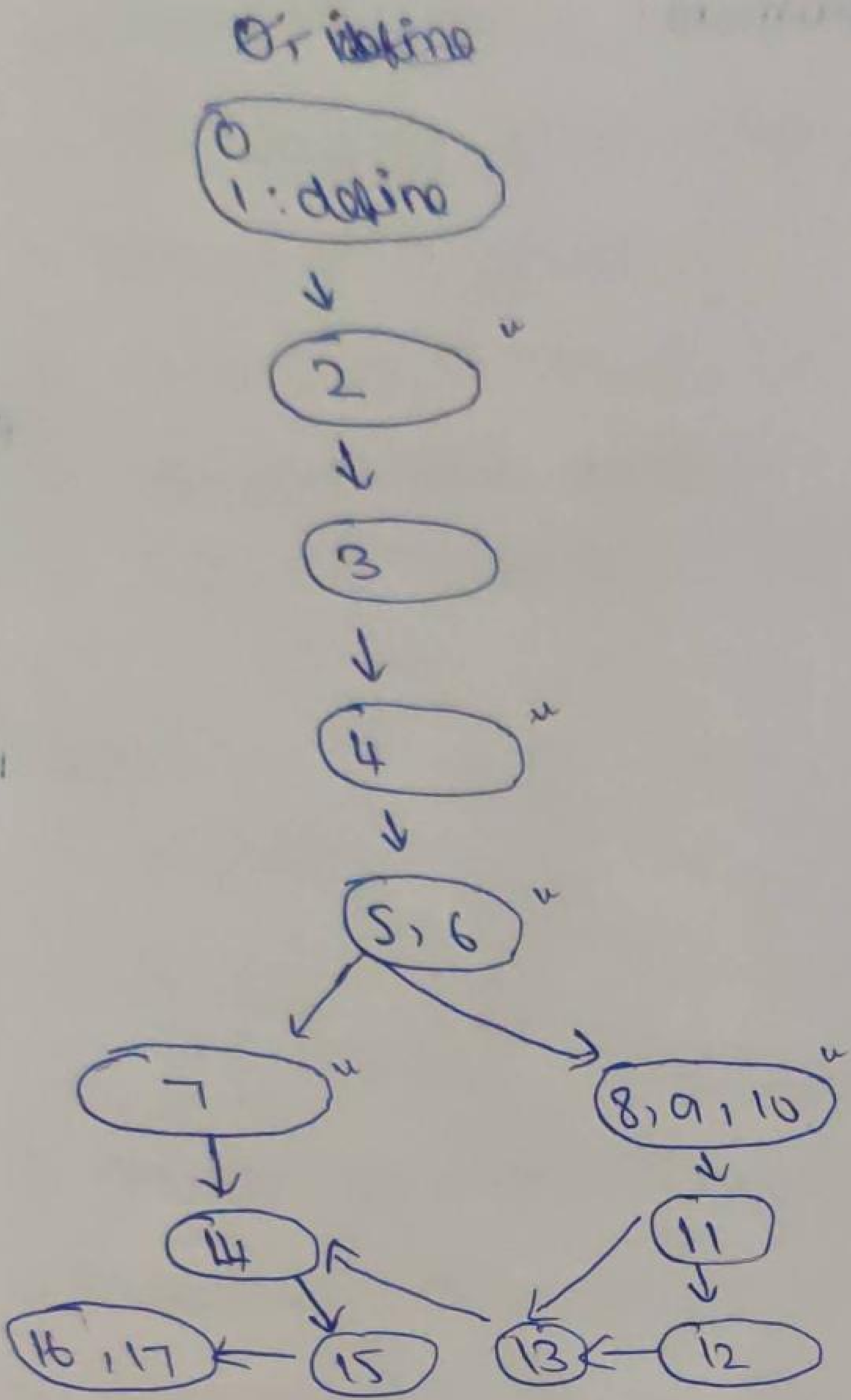


step3: Dataflow or control flow Graph.

Forpayment:



For work:



step4: Anamoly table -> for each variable separately

(i) work

Anamoly	Line no	Anamoly Impact
~d	1	normal
du	1-2	normal
uu	2-4, 4-6, 6-7, 7-10	normal
uk	10-16	normal
k~	16	normal

⑤ payment:

Anamaly	Lino no	Anamaly Impact
~d	0	normal
dd	0-3	harmless bug
du	3-7, 7-10, 10-11, 12-16	normal
ud	7-7, 10-10, 12-12	normal
~~d~~uu	11-12	normal
uk	16-16	normal
k~	16	normal

Anamolly Table:

~d -> normal
~u -> potential
~k -> harmless
d~ -> "
u~ -> normal
k~ -> "

dd -> harmless
du -> normal
dk -> harmlessbug /potential
ud -> potential
uu -> normal
uk -> normal
kd -> serious
ku -> serious or potential
kk -> normall

Mutation Testing:

mutense -> some statements are working fine

change the code or change the Logic

Test ID	T I/P		actual expected result	mutual Result
	x	y		
1.	2	2	4	0
2	4	3	7	12