INDEX

Name SANJANA SURESH

Standard Section Roll No.

Subject AI LAB

School / College

S. No.	Date	Title	Page No.	Teacher's Sign
01	24.09.24	LAB - 01 Tic-tac-toe game	01	
02	01.10.24	LAB - 02 Vacuum Cleaner Problem	02 - 03	
03	8.10.24	LAB - 03 8 puzzle problem - BFS	04 - 05	
04	15.10.24	LAB 8 puzzle problem - A* algorithm	6 - 10	
05	22.10.24	LAB ① Iterative Deepening Search ② Hill climbing Search	11 - 15	

S. No.	Date	Title	Page No.	Teacher's Sign
	29.10.24	LAB - 6		
		Simulated Annealing		
		LAB		
	12.11.24	Propositional Logic - Truth Table Enumeration		
	19.11.24	LAB - 7		
		First Order Logic - unification		
	26.11.24	LAB 8		
		First Order Logic - Forward Chaining		
	2.12.24	LAB - 09.		
		First Order Logic - Resolution		
	16.12.24	LAB-10		
		Alpha Beta Pruning		

24/09

LAB-01

Implement Tic-tac-toe game

Algorithm

1. Create a 3 X 3 board using a two dimensional array
2. Display the empty board.
3. Define a function called evaluate() to check whether the board is filled or not.
4. Create a function called called row_win() to chek whether the rows are filled by winner.
5. Similarly, create column_win() and diag_win().
6. Randomly choose a player to begin playing
7. Display output (the board) after each move.
8. The player inputs the x coordinate and y-coordinate to move.
9. Repeatedly check if any of the winning arrangement returns true.
10. If the current player has won, the game, then print a winning ~~game~~ message and break the loop.
11. If the game is a draw, print draw message.

Qual
24-9-24

01/10/24

LAB-02 (Program - 4)

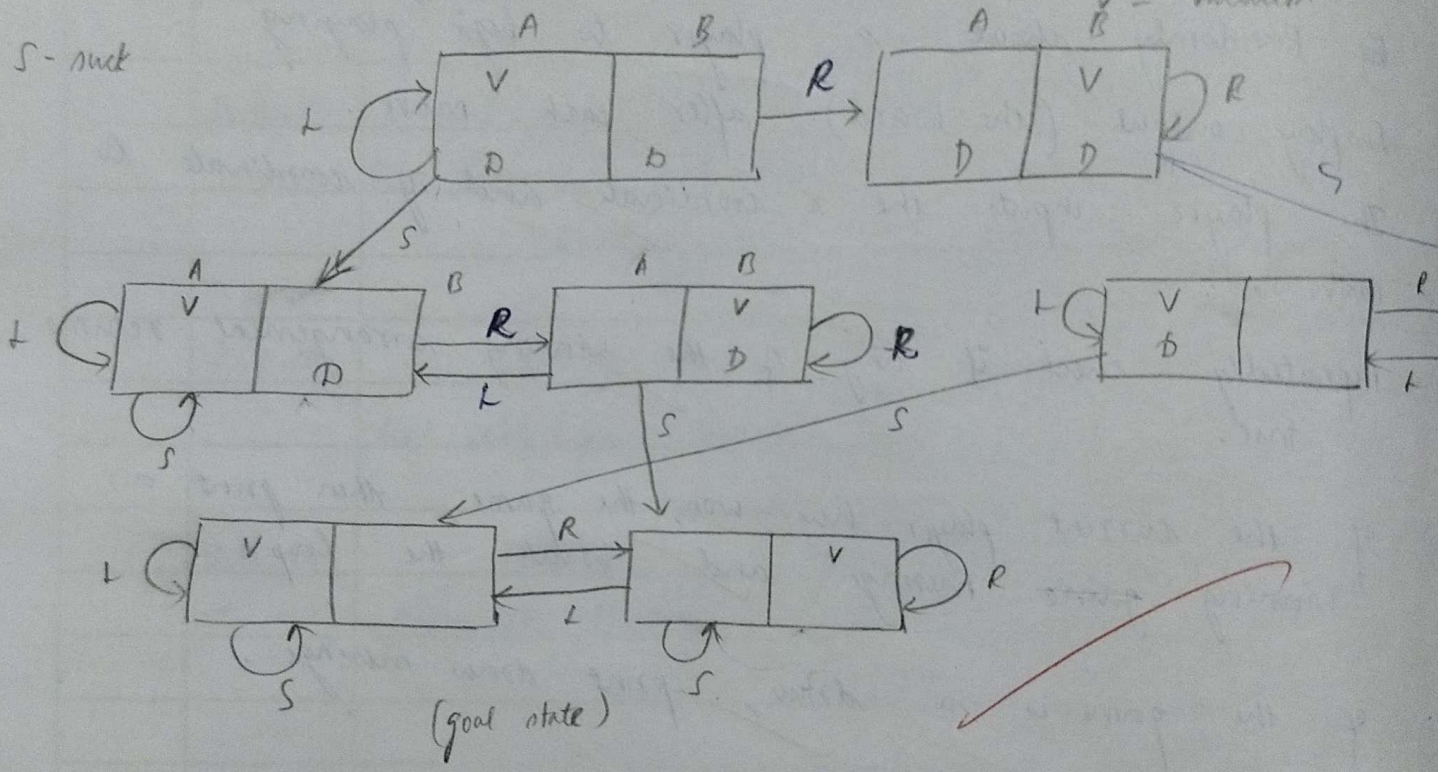
Implement Vacuum Cleaner world

function REFLEX - VACUUM - AGENT ([location, status]) returns an action

if status = Dirty then return Suck
else if location = A then return right
else if location = B then return Left.

L - Left
R - Right
D - Dust
V - Vacuum

State space for the vacuum world



Simple model

0's and 1 input

① Ask location
② Status
③ Is dust available in other room? // enter location
enter status : 0 / 1
Is dust available in room?
④ Cost (every time you suck)

ALGORITHM

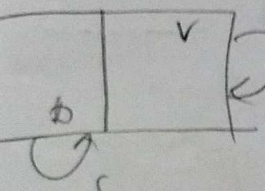
① Define a function called vacuum_world() where user input of the location of the vacuum, the status of the room and status of other room.

② The goal state is defined as [A, 0, B, 0] and cost is initialized to 0.

③ Depending on the user's input, the following steps are followed -

(i) Location is specified as A / B and status is 0 when clean, 1 when room is dirty.

(ii) If the input location is A and the room is dirty, then vacuum has to 'suck' and the goal state for room A is changed to zero.



(iii) Increment the cost variable

(iv) If the other room's status is dirty, then the vacuum has to move right and suck. ~~and goal~~

(v) If the input location is A and the room is clean, then depending on the other room's status the vacuum will remain in room A or move right to room B.

④ Goal attained. and cost calculated.

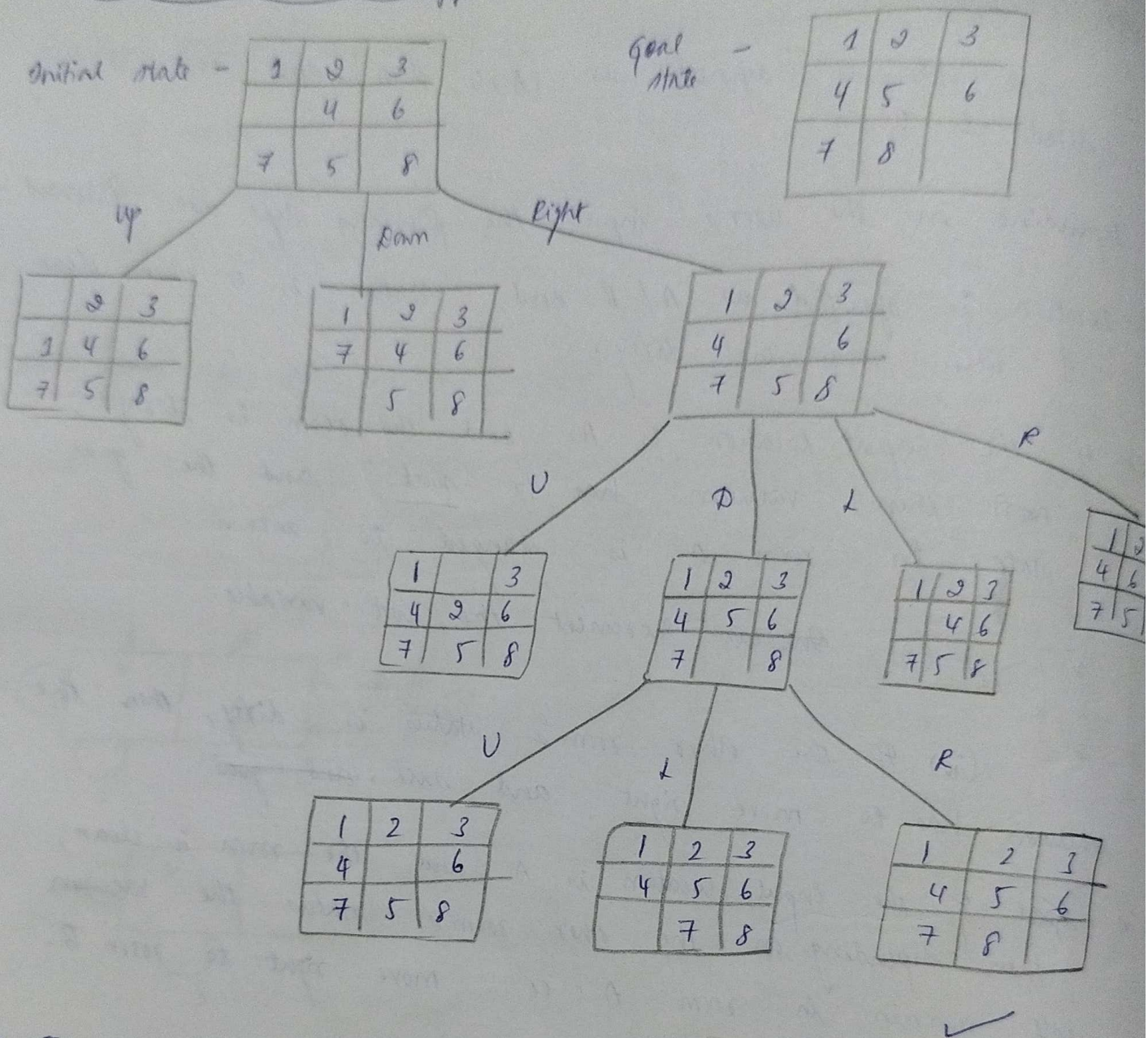
3/10/24

08/10

LAB - 03

Solve 8 puzzle problem using BFS and DFS -

* BFS - Non heuristic approach



① Input : Initial state and final goal state

② Depending on blank space, check for possible actions - UP, DOWN, RIGHT, LEFT

③ Implement a function to locate blank tile in the current state.

No. of sta

④ Implement a function to check if the current state matches the goal state.

(5) Create a function to check if the current state matches the goal state.

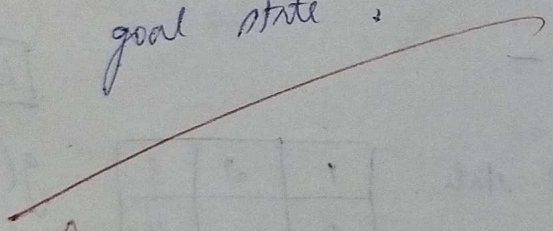
(6) Implement a function to generate a new board configuration by applying a move.

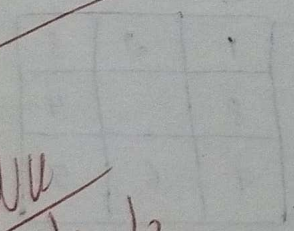
(7) Initialize a queue and add starting state —

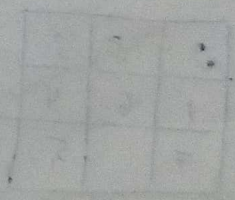
- Use a set to ~~track~~ track visited states to avoid cycles.
- Each entry in the queue consists of current state and the path taken to reach it.

(8) Implement a function to print the configuration once the goal state is reached.

(9) Print the series of moves that moved from initial state to goal state.







13/10/25

15/10

LAB - 03

for 8 puzzle problem using A* implementation to calculate f(n) using –

a) g(n) = depth of a node

h(n) = heuristic value ⇒ no. of misplaced tiles

$$f(n) = g(n) + h(n)$$

b) g(n) = depth

h(n) = heuristic value ⇒ Manhattan distance

$$f(n) = g(n) + h(n)$$

a) **NUMBER OF MISPLACED TILES**

minimum no of displaced t

Draw the state space diagram for –

Level

g(n) =

h(n) =

Initial state

2	8	3
1	6	4
7		5

Goal state

1	2	3
8		4
7	6	5

Left

2	8	3
1	6	4
	7	5

g(n) = 1
h(n) = 5
f(n) = 6

Up

2	8	3
1		4
7	6	5

g(n) = 1
h(n) = 3
f(n) = 4

Right

2	8	3
1	6	4
7	5	

g(n) = 1
h(n) = 5
f(n) = 6

Level

g(n) =

L

2	8	3
	1	4
7	6	5

g(n) = 2
h(n) = 3
f(n) = 2+3 = 5

U

2		3
1	8	4
7	6	5

g(n) = 2
h(n) = 3
f(n) = 5

R

2	8	3
1	4	
7	6	5

g(n) = 2
h(n) = 4
f(n) = 6

D

2	8	3
1	6	4
7		5

×

same as parent state

Left

2	8	3
	1	4
7	6	5

$g(n) = 0$
$h(n) = 3$
$f(n) = 5$

U

	8	3
2	1	4
7	6	5

$g(n) = 3$
$h(n) = 3$
$f(n) = 6$

R

2	8	3
1		4
7	6	5

✗

D

2	8	3
7	1	4
	6	5

$g(n) = 1$
$h(n) = 4$
$f(n) = 7$

Up

2		3
1	8	4
7	6	5

L

	2	3
1	8	4
7	6	5

$g(n) = 1$
$h(n) = 2$
$f(n) = 5$

D

2	8	3
1		4
7	6	5

$g(n) = 1$
$h(n) = 3$
$f(n) = 6$

R

2	3	
1	8	
7	6	5

$g(n) = 1$
$h(n) = 3$
$f(n) = 6$

R

2		3
1	8	4
7	6	5

$g(n) = 4$ $h(n)$ ✗

D

1	2	3
	8	4
7	6	5

$g(n) = 4$
$h(n) = 1$
$f(n) = 5$

U

	2	3
1	8	4
7	6	5

✗

D

1	2	3
7	8	4
	6	5

R

1	2	3
8		4
7	6	5

✓

$g(n) = 5$

Goal state

* Algorithm

Step 1 : Place the starting node in the open list

Step 2 : Check if the open list is empty or not if the list is empty then return failure and stops. // No input

Step 3 : Select the node from the OPEN list which has the smallest value of evaluation function ($g(n) + h(n)$) if node n is goal node, then return success and stop otherwise

Step 4 : Expand node n and generate all of its successors and put n into the closed list. For each successor n, check whether 'n' is already in the open or closed list. If not then compute the evaluation function for 'n'. and place it in open list.

Step 5 : Else if node n is already in open and closed, then it should be ~~attended~~ attached to the back pointer which reflects the lowest $f(n)$ value.

10/4/22

b) MANHATTAN DISTANCE

Initial state

2	8	3
1	6	4
7		5

$g(n) = 0$

Goal state

1	2	3
8		4
7	6	5

L

2	8	3
1	6	4
	7	5

$g(n) = 1$
$h(n) = 6$
$f(n) = 7$

U

2	8	3
1		4
7	6	5

$g(n) = 1$
$h(n) = 4$
$f(n) = 5$ ✓

R

2	8	3
1	6	4
7	5	

$g(n) = 1$
$h(n) = 6$
$f(n) = 7$

L

2	8	3
	1	4
7	6	5

$g(n) = 2$
$h(n) = 2+1+2 = 5$
$f(n) = 7$

U

2		3
1	8	4
7	6	5

$g(n) = 2$
$h(n) = 1+1+1 = 3$
$f(n) = 5$ ✓

R

2	8	3
1	4	
7	6	5

$g(n) = 2$
$h(n) = 1+1+1+2 = 5$
$f(n) = 7$

D

2	8	3
1	6	4
7		5

✗

L

	2	3
1	8	4
7	6	5

$g(n) = 3$
$h(n) = 1+1 = 2$
$f(n) = 5$ ✓

D

2	8	3
1		4
7	6	5

$g(n) = 3$
$h(n) =$
✗

R

2	3	
1	8	4
7	6	5

$g(n) = 3$
$h(n) = 1+1+1+1 = 4$
$f(n) = 7$

R

2	3	
1	8	4
7	6	5

✗

D

1	2	3
	8	4
7	6	5

$g(n) = 4$
$h(n) = 1$
$f(n) = 5$

U

	2	3
1	8	4
7	6	5

R

1	2	3
8		4
7	6	5

✓

D

1	2	3
7	8	4
	6	5

$g(n) = 5$
$h(n) = 0$
$f(n) = 5$

4

2

8

Manhattan distance for $g(n) = 1$

1	2	3	4	5	6	7	8
1	1	0	0	0	1	1	2

Left

Sum = 1+1+1+1+2 = 6

1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	2

Up

Sum = 4

1	2	3	4	5	6	7	8
1	1	0	0	1	1	0	2

Right

Sum = 6

Manhattan distance for $g(n) = 2$

Dev
15/10/18

LAB

Implement iterative deepening search algorithm — (IDS)

Algorithm –

1. For each child of the current node.
2. If it is the target node, return the node.
3. If the current maximum depth is reached, return.
4. Set the current node to this node and go back to 1.
5. After having gone ~~all~~ through all children, go to the next child of the parent (the next sibling)
6. After having gone through all children of the start node, increase the maximum depth and go back to 1.
7. If we have reached all leaf (bottom) nodes, the goal node doesn't exist.

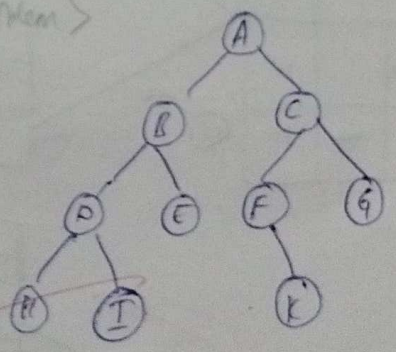
function ITERATIVE-DEEPENING-SEARCH (problem) returns a solution or failure

for depth = 0 to ∞ do

result ← DEPTH-LIMITED-SEARCH (problem, depth)

if result ≠ cutoff then return result.

<Explore with 8 puzzle problem>



– Level 0

– Level 1

– Level 2

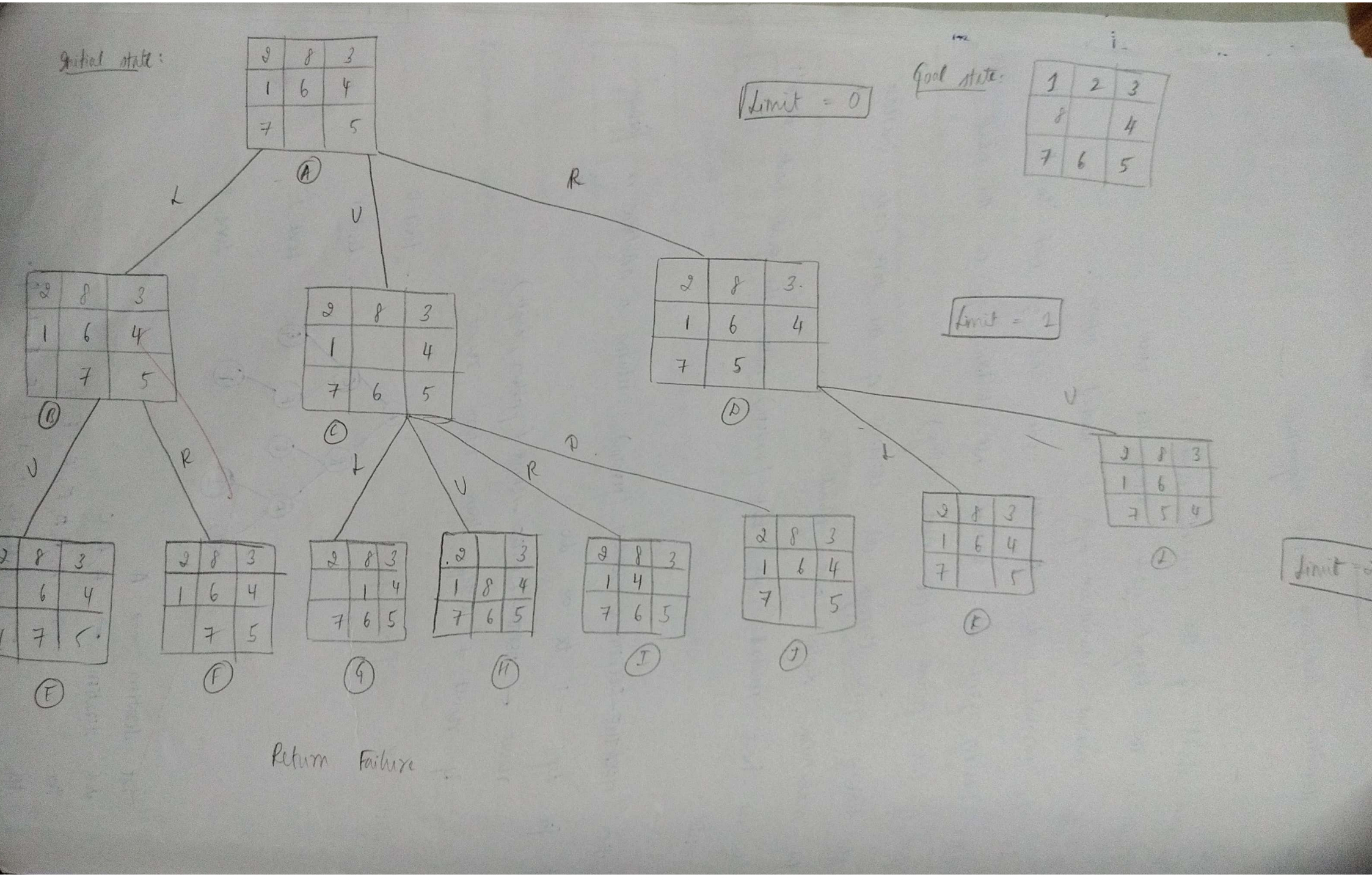
– Level 3

1st iteration → A

2nd iteration → A, B, C

3rd → A, B, D, E, C, F, G

4th → A, B, D, H, I, E, C, F, K, G.


Initial state:
Goal state.
Limit = 0
Limit = 1
L
U
R
A
B
C
D
E
F
G
H
I
J
Return Failure

1st iteration — A

2nd iteration — A, B, C, D,

3rd iteration — A, B, E, F, C, G, H, I, J, D, K, L

22/12/~

Implement Hill climbing search algorithm to solve N-queens problem

Algorithm –

function HILL-CLIMBING (problem) returns a state that is a local maximum.

current ← MAKE-NODE (problem, INITIAL-STATE)

loop do

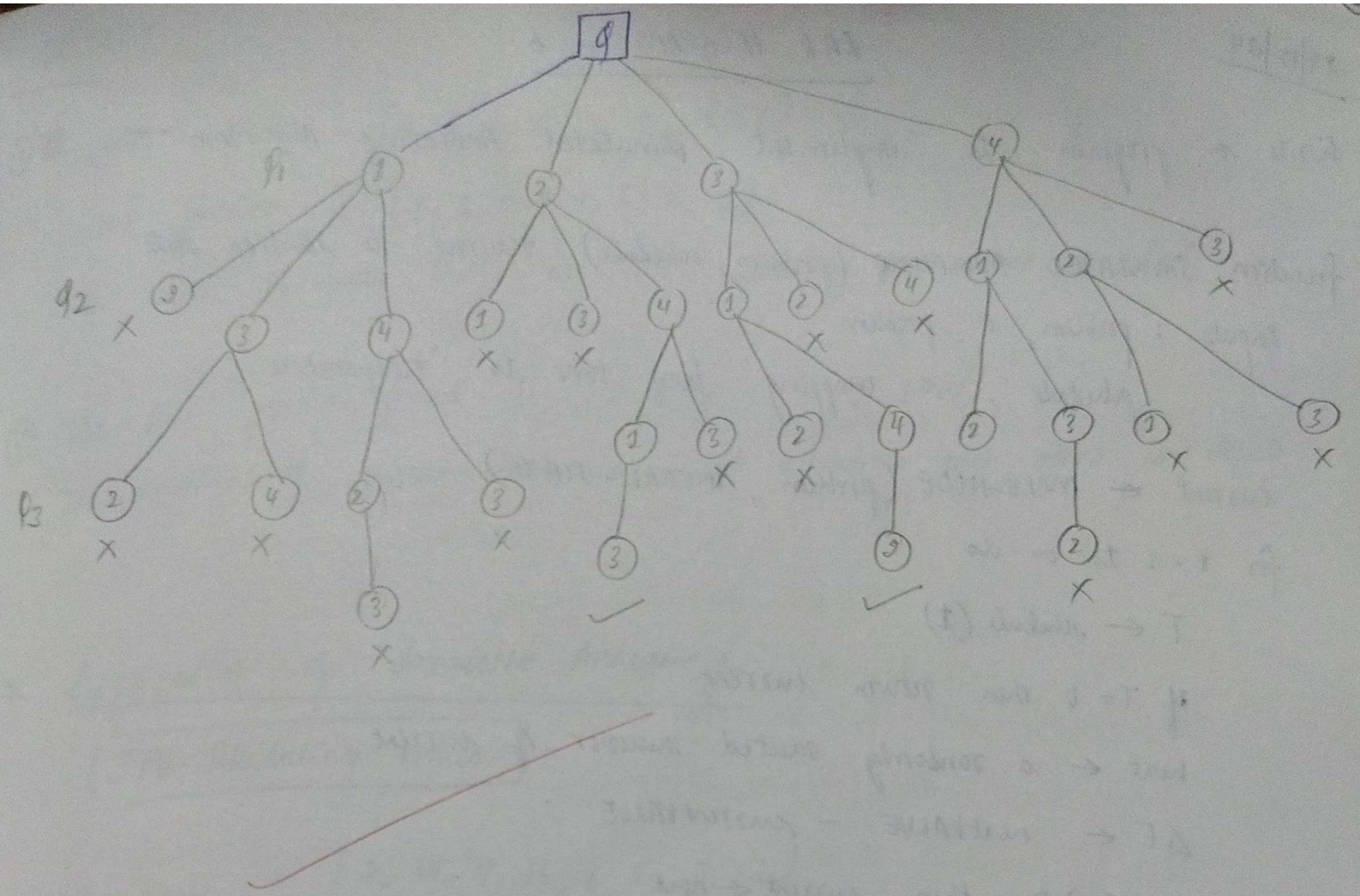
neighbour ← a highest valued successor of current

if neighbour.VALUE ≤ current.VALUE then return current.STATE

current ← neighbour

Problem Formulation –

- State : 4 queens on the board. One queen per column.
  - Variable : x_0, x_1, x_2, x_3 where x_i is the ~~position~~ row position of the queen in column i.
    : Assume that there is one queen per column.
  - Domain for each variable : $x_i \in \{0, 1, 2, 3\}, \forall i$
- Initial state – a random state
- Goal state – 4 queens on the board. * No pair of queens are attacking each other.
- Neighbour relation – Swap the row positions of two queen.
- Cost function – The number of pairs of queens attacking each other, directly or indirectly.



Due
22/10/24

29/10/24

LAB PROGRAM - 6

Write a program to implement Simulated Annealing Algorithm —

function SIMULATED-ANNEALING (problem, schedule) returns a solution state
  inputs : problem, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE (problem, INITIAL-STATE)
  for t = 1 to ∞ do
    T ← schedule (t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← nextVALUE − currentVALUE
    if ΔE > 0 then current ← next
    else current ← next only with probability $e^{\Delta E/T}$

* 8 Queen problem using Simulated Annealing

mlrose - Python package

mlrose for simulated annealing

① Import the mlrose and numpy libraries

②

$$P(x, x_j, T) = \begin{cases} 1 & F(x_j) \geq F(x) \\ e^{\frac{F(x_j) - F(x)}{T}} & F(x_j) < F(x) \end{cases}$$

OUTPUT

① initial position = [4, 6, 1, 5, 2, 0, 3, 7]

best position = [0, 6, 4, 7, 1, 3, 5, 2]

The number of queen that are not attacking each other is 8.0.

② The best position found is [2 5 7 1 3 0 6 4]

The number of queens that are not attacking each other is : 28.0

* Application of Simulated Annealing

Job Scheduling Problems

job_times = [2, 14, 4, 16, 6, 5, 3, 12]

num_machine = 3

O/P Best job-to-machine assignment : [1 2 0 1 2 0 1 0]

Minimum makespan : 21.0

Machine 1 jobs : [(2, 4), (5, 5), (7, 12)] Total time : 21

Machine 2 jobs : [(0, 2), (3, 16), (6, 3)] Total time : 21

Machine 3 jobs : [(1, 14), (4, 6)] Total time : 20

29-10-24

WEEK - 07

13/11

Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not

Algorithm :

function TT-ENTAILS?(KB, α) returns true or false

inputs : KB, - the knowledge base, a sentence in propositional logic
α - the query, a sentence in propositional logic

symbols ← a list of the proposition symbols in KB and α
return TT-Check-All (KB, α, symbols, {})

function TT-Check-All (KB, α, symbols, model) returns true or false

if EMPTY? (symbols) then
if PL-TRUE? (KB, model) then return PL-TRUE? (α, model)
else return true // when KB is false, always return true

else do
P ← FIRST (symbols)
rest ← REST (symbols)
return (TT-CHECK-ALL (KB, α, rest, model $\cup$ {P = true})
and
TT-CHECK-ALL (KB, α, rest, model $\cup$ {P = false}))

Truth table

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Leftrightarrow Q$	$P \Rightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	false	true
true	false	false	false	true	false	false
true	true	false	true	true	true	true

$\wedge$ - and $\Rightarrow$ implies if P then Q
$\vee$ - or

Example

$\alpha = A \vee B$ $\quad KB = (A \vee C) \wedge (B \vee \neg C)$

Checking $KB \models \alpha$

A	B	C	A ∨ C	B ∨ ¬C	KB	α
F	F	F	F	T	F	F
F	F	T	T	F	F	F
F	T	F	F	T	F	T
F	T	T	T	T	T	T
T	F	F	T	T	T	T
T	F	T	T	F	F	T
T	T	F	T	T	T	T
T	T	T	T	T	T	T

O/P Seen

SEO

12/11/24

LAB PROGRAM - 7.

19/11/24

Implement unification in First order Logic

Algorithm : Unify (Ψ_1, Ψ_2)

Step 1 : If Ψ_1 or Ψ_2 is a variable or constant, then :

a) If Ψ_1 or Ψ_2 are identical, then return NIL.

b) Else if Ψ_1 is a variable,

a. then if Ψ_1 occurs in Ψ_2, then return FAILURE.

b. Else return {(Ψ_2 / Ψ_1)}.

c) Else if Ψ_2 is a variable,

a. If Ψ_2 occurs in Ψ_1, then return FAILURE,

b. Else return {(Ψ_1 / Ψ_2)}.

d) Else return FAILURE.

Step 2 : If the initial Predicate symbol in Ψ_1 and Ψ_2 are not the then return FAILURE.

Step 3 : If Ψ_1 and Ψ_2 have a different number of ~~arguments~~ arguments, then return FAILURE.

Step 4 : Set substitution set (SUBST) to NIL.

Step 5 : For i = 1 to the number of elements in Ψ_1,

a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2 and put the result into S.

b) If S = failure, then return failure.

c) If S ≠ NIL, then do –

a. Apply S to the remainder of both L1 and L2.

b. SUBST = APPEND (S, SUBST).

Step 6 : Return SUBST.

Eg: I $P(x, F(y)) \longrightarrow ①$

$P(a, F(g(z)) \longrightarrow ②$

① predicate

② no of arguments

① and ② are identical if x is replace with 'a' in ①

$P(a, F(y))$

if y is replaced with $g(z)$.

$P(a, F(g(z))$

II $\varphi(a, g(z, a), f(y)) \longrightarrow ①$

$\varphi(a, g(f(b), a), z) \longrightarrow ②$

Replace x in ① with $f(b)$ $\Rightarrow$ $\varphi(a, g(f(b), a), f(y))$

Replace $f(y)$ in ② with z $\Rightarrow$ $\varphi(a, g(f(b), a), z)$

Q1) $\psi_1 = P(f(a), g(x))$

$\psi_2 = P(x, x)$

If the predicates f and g are distinct and not same, they cannot be substituted as the same variable x.

Unification fail

Q2) $\psi_1 = P(b, x, f(g(z)) \longrightarrow ①$

$\psi_2 = P(z, f(y), f(y)) \longrightarrow ②$.

Replace b with z in ① $P(z, x, f(g(z))$

x with $f(y)$ in ① $P(z, f(y), f(g(z))$

$g(z)$ with y $P(z, f(y), f(y))$

OUTPUT –

Case 1 : P(f(a), f(b)) and P(z, z).

Unification unsuccessful!

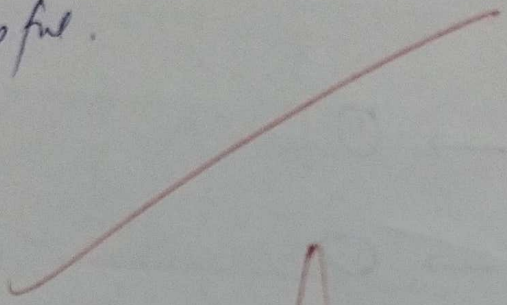
Case 2 : P(b, x, f(g(z))) and P(z, f(y), f(y))

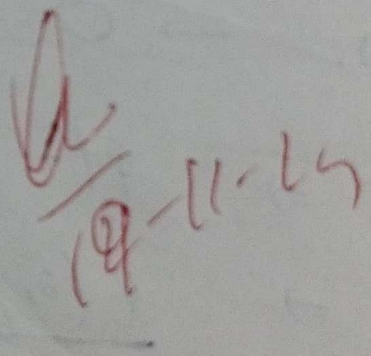
Replace b with z

Replace x with f(y)

Replace y with g(z)

Unification successful.





LAB PROGRAM - 8

26/11

For a given case study you should create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

Algorithm : FORWARDREASONING

function FOL - FC - ASK (KB, α) returns a substitution or false

inputs : KB, the knowledge base, a set of first order definite clauses.

α, the query, an atomic sequence

local variables : new, the new sentences inferred on each iteration

repeat until new is empty

new $\leftarrow$ { }

for each rule in KB do

$(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE - VARIABLES (rule)

for each θ such that SUBST (θ, $p_1 \wedge \dots \wedge p_n$) = SUBST ($\theta$, $p_1' \wedge \dots \wedge p_n'$)

for some $p_1', \dots p_n'$ in KB

$q' \leftarrow$ SUBST (θ, q)

if q' does not unify with some sentence already in KB or new then

add q' to new

$\phi \leftarrow$ UNIFY (q', α)

if ϕ is not fail then return ϕ

add new to KB

return false.

o/p

Query {'c': 'john'} is TRUE with substitution {'c': 'John'}

kb = [[{"A": "x"}, {"B": "x"}, {"C": "x"}], # A(?) and B(?) ⇒

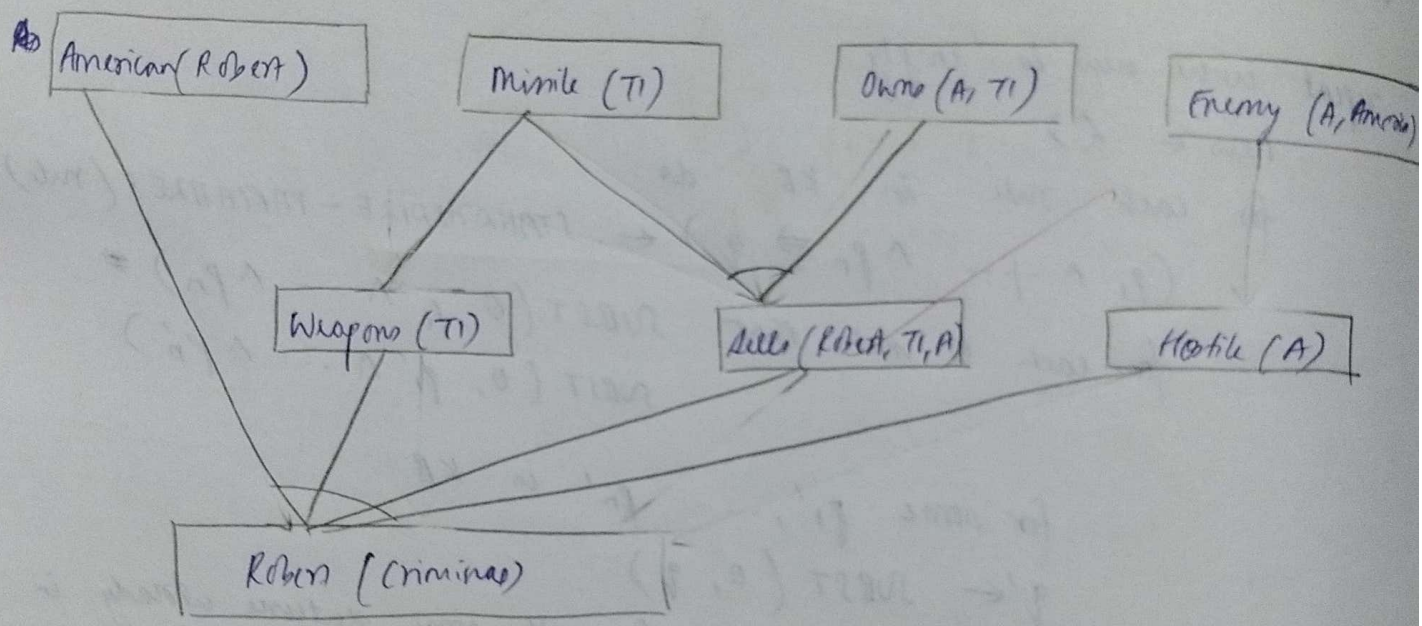
{"A": "John"}, #Fact : A(John)

{"B": "John"} #Fact : B(John)

]

query {"C": "John"}.

o/p 2



- It is a crime for an American to sell weapons to hostile nation.

Let's say p, q, r are variable.

Rule 1

American(p) ∧ weapon(q) ∧ sells(p, q, r) ∧ Hostile(r) ⇒ Criminal(p

- Country A has some missiles.

∃x Owns(A, x) ∧ Missile(x)

Owns(A, T1)

Missile(T1)

• All of the missiles were sold to country A by Robert.

rule 2 $\forall x$ Missile $(x) \wedge$ Owns $(A, x) \Rightarrow$ Sells (Robert, x, A).

• missiles are weapons.

rule 3 missile $(x) \Rightarrow$ weapon (x)

• Enemy of America is known as hostile

rule 4 $\forall x$ Enemy $(x,$ America$) \Rightarrow$ Hostile (x)

To prove : Robert is criminal

Facts
① Robert is an American.
② Country A is an enemy of America.

Rules

④

Done
26-11-24

LAB PROGRAM - 9

Convert a given first order logic statement into resolution -

Basic steps for proving a conclusion S given premises -

Premise$_1$, ... premise$_n$

(all expressed in FOL)

1. Convert all sentences to CNF.
2. Negate conclusion S and convert result to CNF
3. Add negated conclusion S to the premise clauses.
4. Repeat until contradiction or no progress is made -
   a) Select 2 clauses (call them parent clauses)
   b) Resolve them together, performing all required unifications.
   c) If resolvent is the empty clause, a contradiction has been found (i.e) S follows from the premises)
   d) If not, add resolvent to the premises.

If we succeed in step 4, we have proved the conclusion.

I/P :- John eats peanuts

Example –

Given the KB or premise :

① John like all kind of food.

② Apple and vegetable are food.

③ Anything anyone eats and not killed is food.

④ Anil eats peanuts and still alive

⑤ Harry eats everything that Anil eats.

⑥ Anyone who is alive implies not killed.

⑦ Anyone who is not killed implies alive.

Prove by resolution that – ⑧ John like peanut.

① $\forall x : food(x) \rightarrow like(John, x)$

② $food(Apple) \wedge food(vegetable)$

③ $\forall x \forall y : eats(x,y) \wedge \neg killed(x) \rightarrow food(y)$

④ $eats(Anil, Peanuts) \wedge alive(Anil)$

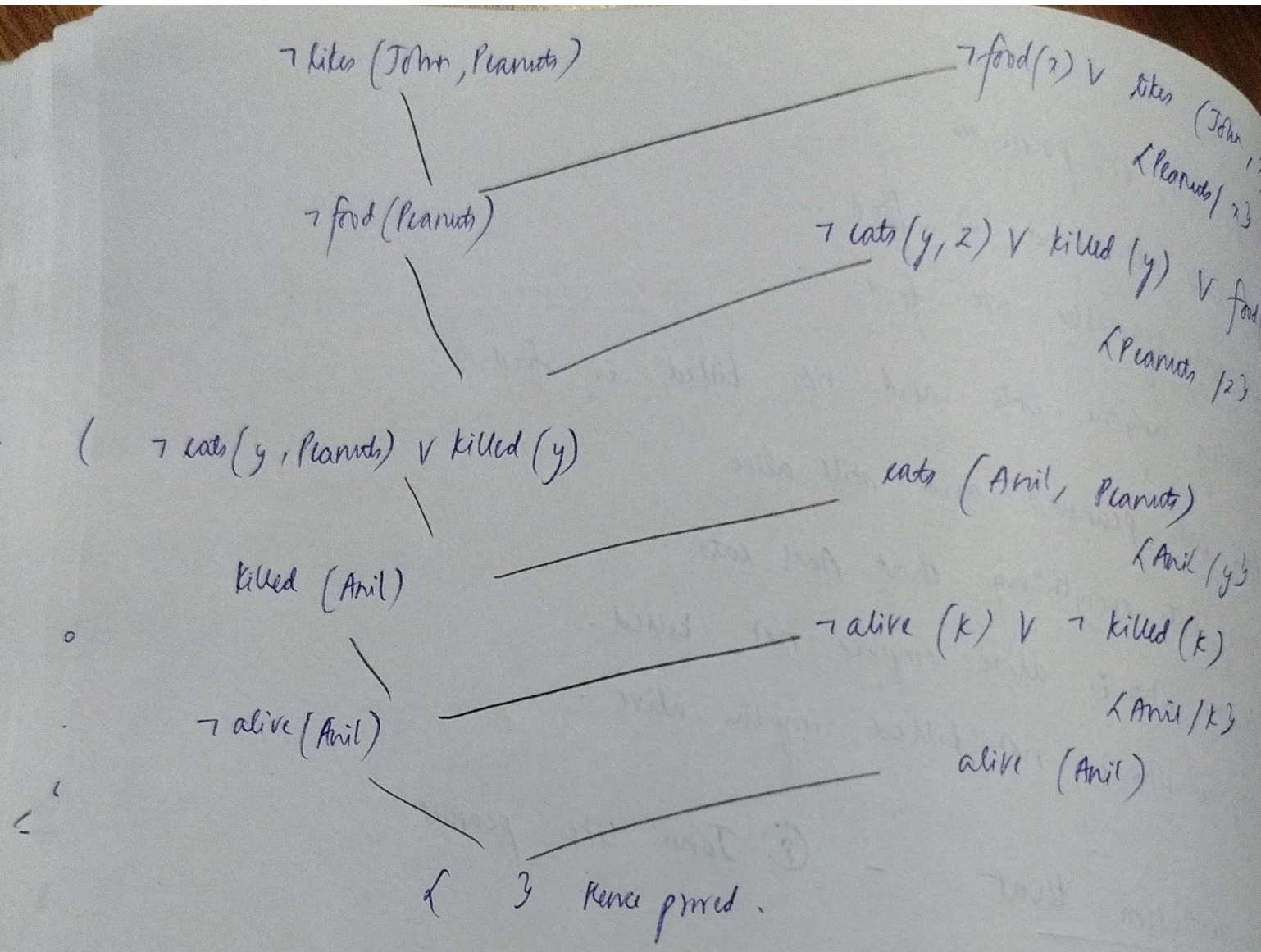
⑤ $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$

⑥ $\forall x : \neg killed(x) \rightarrow alive(x)$

⑦ $\forall x : alive(x) \rightarrow \neg killed(x)$

⑧ $like(John, Peanuts)$

Rules ① $\alpha \Rightarrow \beta$ with $\neg \alpha \vee \beta$


¬ likes (John, Peanuts)
¬ food (Peanuts)
¬ food (x) V likes (John
{Peanuts/x}
¬ eats (y, z) V killed (y) V food
{Peanuts/z}
(¬ eats (y, Peanuts) V killed (y)
killed (Anil)
eats (Anil, Peanuts)
{Anil/y}
¬ alive (Anil)
¬ alive (k) V ¬ killed (k)
{Anil/k}
alive (Anil)
{ } Hence proved.