INDEX

Name : Sakshi . B.R

Std : Div : Roll No :

Sub :

School / College :

Sr. No.	Date	Title	Remark	Pg. No.
1.a)	24/9/24	Tic Tac Toe	1-2	
1.b)	1/10/24	Vacuum World Cleaner	3-4	
2.	8/10/24	8 puzzle problem		
3 & 4	15/10/24	Depth limited search algorithm & Uniform cost search algorithm		
5 & 4	22/10/24	Iterative Deepening Hill Climbing		
5	29/10/24	Annealing Problem		
6	12/11/24	Propositional logic		
7	19/11/24	Unification in FOL		
8	26/11/24	First Order Logic.		
9	26/11/24	FOL to CNF		
10	26/11/24	Alpha Beta Pruning		

GoodLuck | Page No.
Date 24 | 3 | 24

Lab - 1

Tic Tac Toe

→ Initialize the 2D array of 3 row and 3 column with empty space

→ Create the function "board" for display

→ Take the input from the user as 'X' or 'O'

→ Handle the I/P of the user by checking the row, column, diagonal wise by the condition.

→ Create the function victory to check
Iterate and check if all the row/col as occupied by same icon
if occupies → True

If not → False

→ Create a function for draw
Iterate over the board and check whether all the cells are filled or not
if filled → break

→ Create a function for draw
Iterate over the board and check whether all the cells are

→ Create the main game function drawBoard:
loop the game

- Ask current player to make move
- Update the move
- Check for the victory and print message
- Check for draw & print draw message
- Ask for draw & print draw message
- Ask for next player move
- Update the move and it goes on.

GoodLuck | Page No.
Date 1 | 10 | 24

Lab - 2

Implement vacuum world cleaner

function Reflex - vacuum - Agent ([Location, Status)]
return an action

if Status = Dirty then
return suck
else if location = A: then
return Right
else if location = B then
return left

Input :: Location A
Status 0
Dust in another = 1.

Algorithm :-

- Take input:
  Take input from user for the initial location of the vacuum
  Take input from user for the status of the room (0 for clean, 1 for dirty).
- Check if the current room is dirty
  If dirty:
  clean the room
  Increase the total cost by 1
  If already clean, print a message that the room is already clean.
- Move to the next room

- Clean the room
- Check goal state
- Print total cleaning cost.

Output

Current state: {'A':0, 'B':0}
Vacuum is in Room A
Total cost so far: 0
Enter initial location of the vacuum (A or B): A
Enter the status for Room A : 1
Enter the status for Room B : 0
Suck at A
Move Right to B.
Move Left to A
Total cost : 1
Goal Reached.

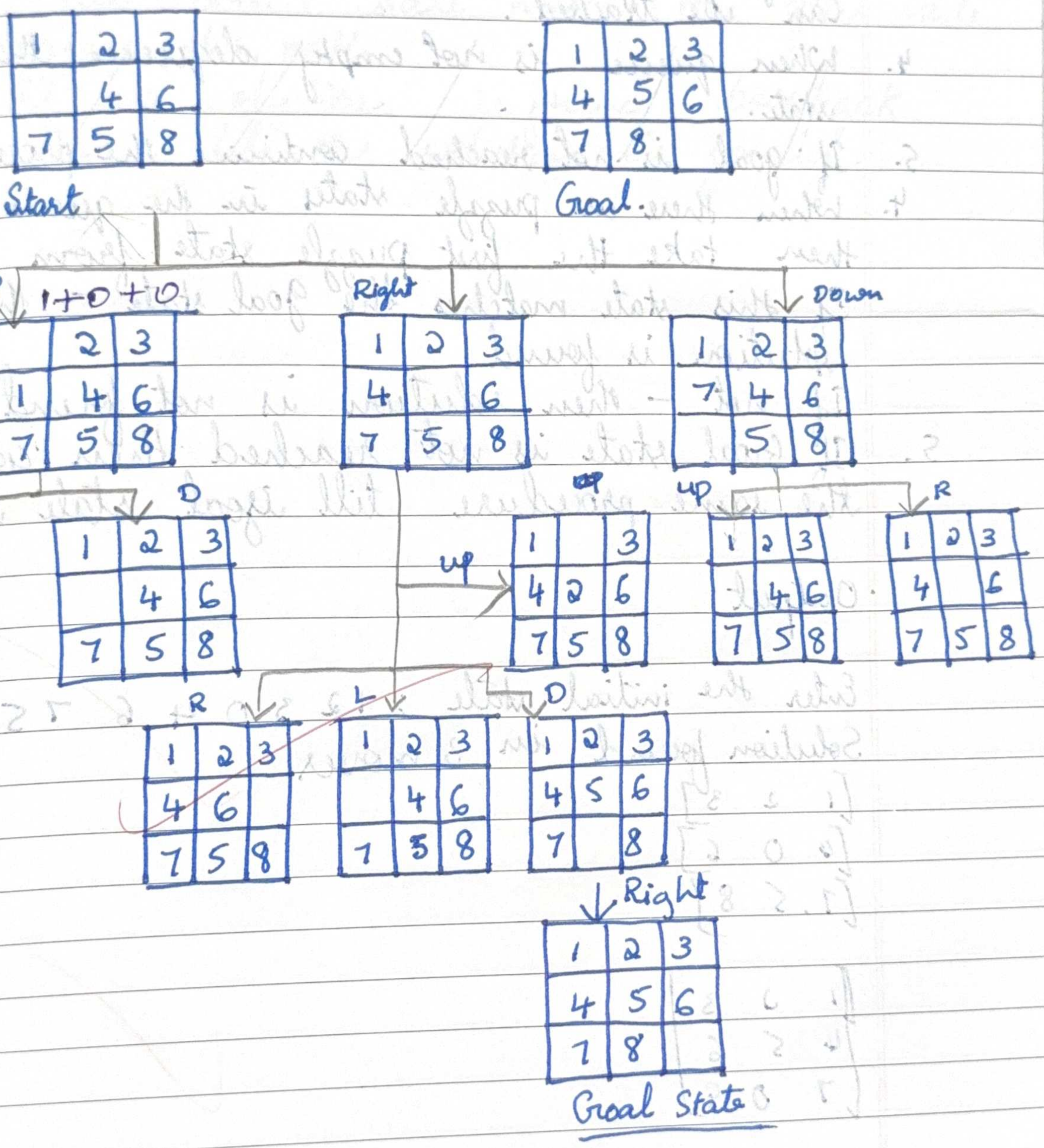
11/10/24

GoodLuck | Page No.
Date 8 | 10 | 24.

Lab - 2

Solve 8 puzzle using DFS and BFS.

State Space Diagram.



BFS

1. Defining of start state and goal state.
2. Use queue to track the states to reach goal state.
3. By using set visited and non visited states can be tracked.
4. When queue is not empty dequeue the first state.
5. If goal is not reached continue the procedure.
4. When there are puzzle states in the queue then take the first puzzle state from the queue
   If this state matches the goal state - then solution is found
   If not - then solution is not found
5. If Goal state is not reached then continue the same procedure till goal state is reached.

Output

Enter the initial state : 1 2 3 0 4 6 7 5 8
Solution found in 3 moves
[1 2 3]
[4 0 6]
[7 5 8]

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

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

GoodLuck | Page No.
Date

DFS Algorithm.

1. Set the goal configuration of the puzzle
2. Locate the empty space
3. Generate valid state by moving adjacent tiles into empty space
4. If the current state matches the goal state return the path
5. If no solution is found - backtrack. Continue till the solution is found.

GoodLuck | Page No.
Date 15 | 10 | 24

Lab - 3

For 8 puzzle problem using A* implementation

a) to calculate f(n) using
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).

Draw the state space diagram for

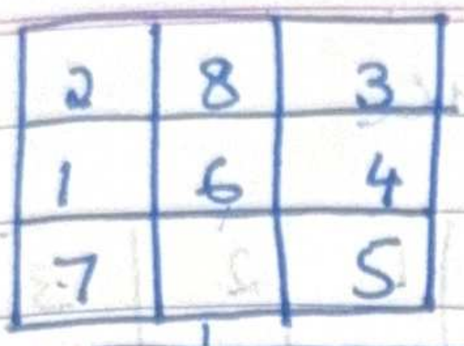
2	8	3
1	6	4
7		5

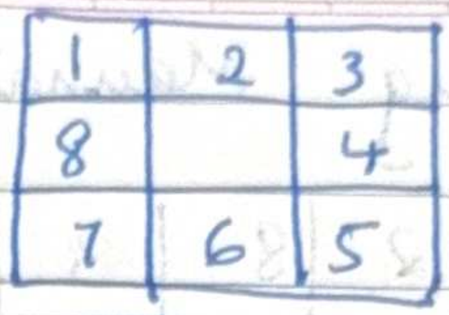
Initial state

1	2	3
8		4
7	6	5

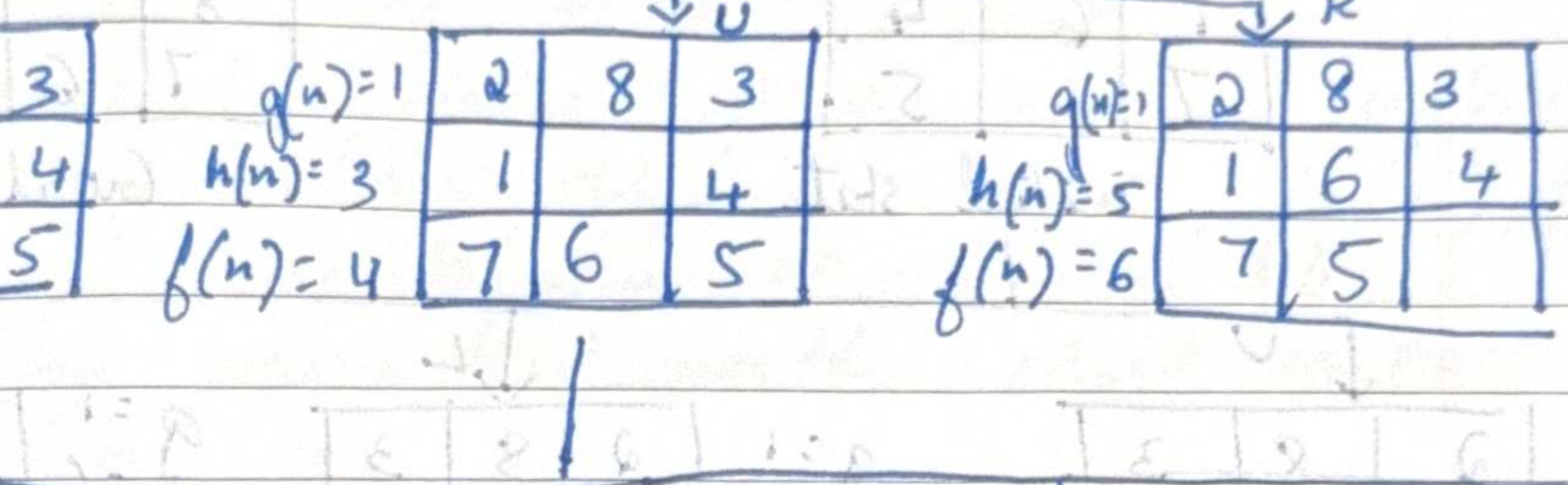
Goal state.

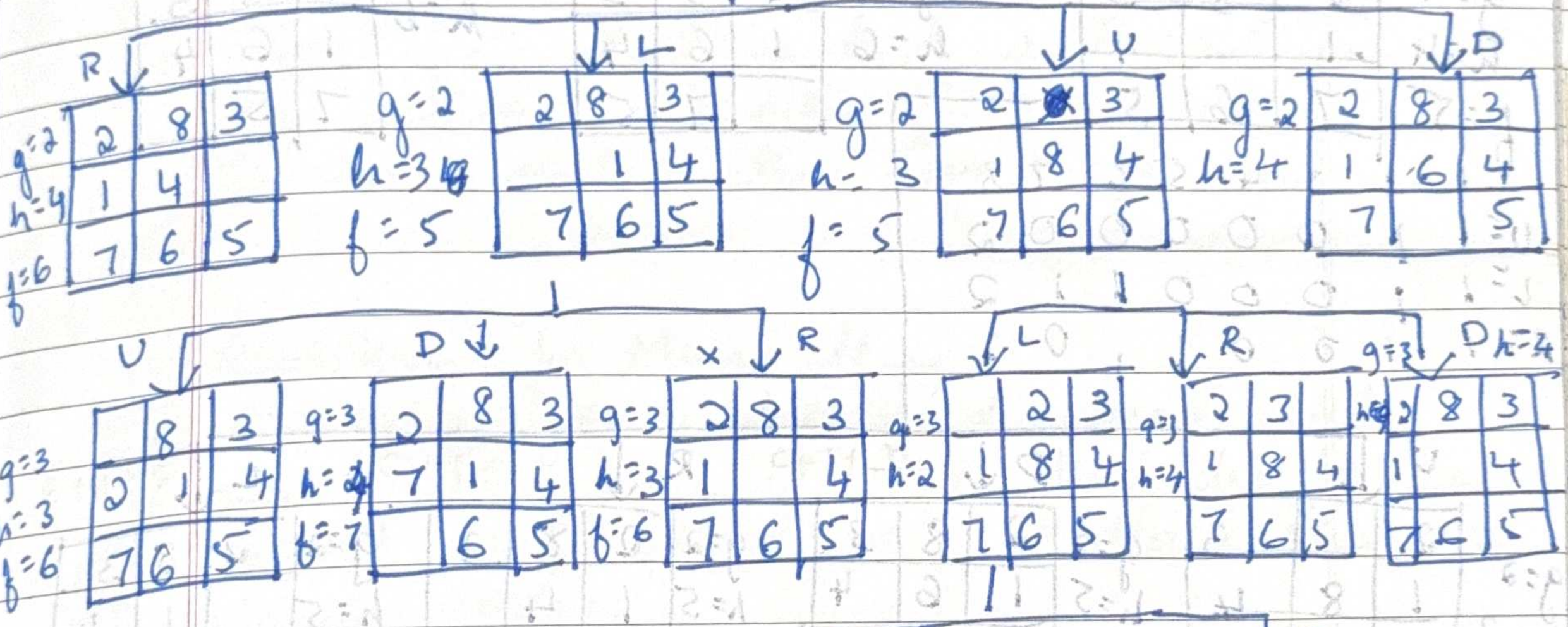
a)

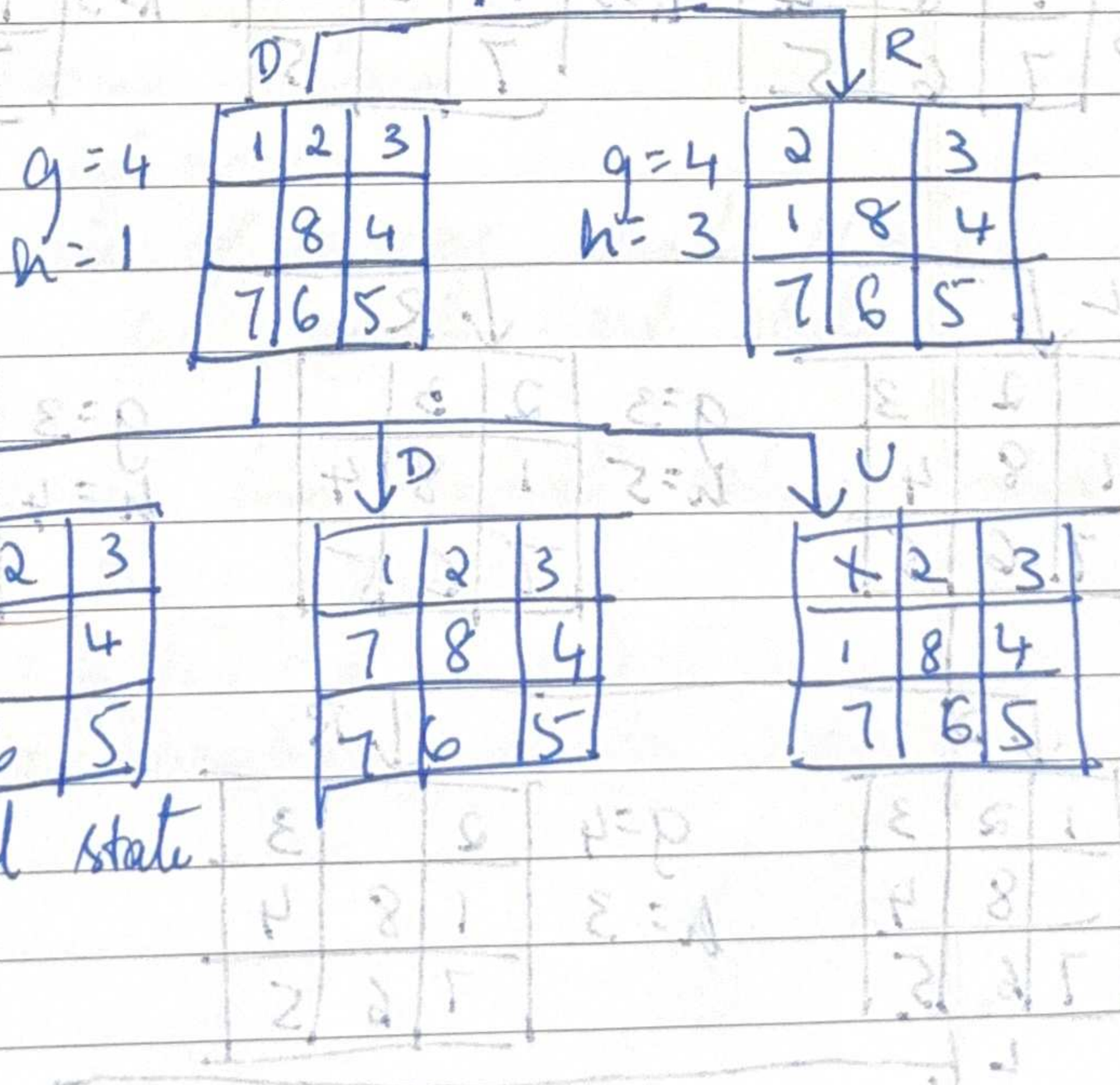




Goal state







Goal state

b) Using Manhattan Distance

2	8	3
1	6	4
7		5

Initial state

1	2	3
8		4
7	6	5

Goal state

1

U

g=1, h=4, f=5

2	8	3
1		4
7	6	5

L

g=1, h=6

2	8	3
1	6	4
	7	5

R

g=1, h=6

2	8	3
1	6	4
7	5	

	1	2	3	4	5	6	7	8
U=1	1	0	0	0	0	0	0	2
L=1	1	0	0	0	1	1	0	2
R=1	1	0	0	0	1	1	0	2

U 1+1+1

g=2, h=3

2		3
1	8	4
7	6	5

D 1+2+1+0

g=2, h=5

2	8	3
1	6	4
7		5

R 1+1+1+0

g=2, h=5

2	8	3
1	4	
7	6	5

L 2+1+2

g=2, h=5

2	8	3
	1	4
7	6	5

L

g=3, h=2

	2	3
1	8	4
7	6	5

R

g=3, h=5

2	3	
1	8	4
7	6	5

D

g=3, h=4

2	8	3
1		4
7	6	5

D

g=4, h=1

1	2	3
	8	4
7	6	5

R

g=4, h=3

2		3
1	8	4
7	6	5

L

g=5, h=0

1	2	3
8		4
7	6	5

Goal state //

U

g=5, h=3

	2	3
1	8	4
7	6	5

D

g=5, h=2

1	2	3
7	8	4
	6	5

GoodLuck | Page No.
Date

Algorithm for heuristic :

1. Initialize open list and closed list.
   open list - States to be explored.
   Closed list - States that are already explored.
2. Define function that is $g(n)$, $h(n)$, f and $f(n)$ where $h(n)$ is no misplaced tiles.
3. Initialize open list with start state.
   Start $g(n) = 0$.
4. Select $f(n)$ which is least and continue with the procedure. Where $f(n) = g(n) + h f(n)$
5. If goal is reached then stop the procedure.

Algorithm for Manhatten.

1. Initialize open list and closed list where open list is states that are to be explored and closed list is states that are already explored.
2. Define functions where
   $g(n)$ is depth of node
   $h(n)$ is number of moves and $f(n)$
3. Initialize open list with start state
   Start $g(n) = 0$.
4. Select $f(n)$ which has lowest value and continue with the procedure
   where $f(n) = g(n) + h(n)$
5. If the goal is reached then stop the procedure

GoodLuck | Page No.
Date

(a)/(b) 0/1

Initial State

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix}$$

Goal state

$$\begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

Sol found with cost : 5

Steps

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 & 3 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

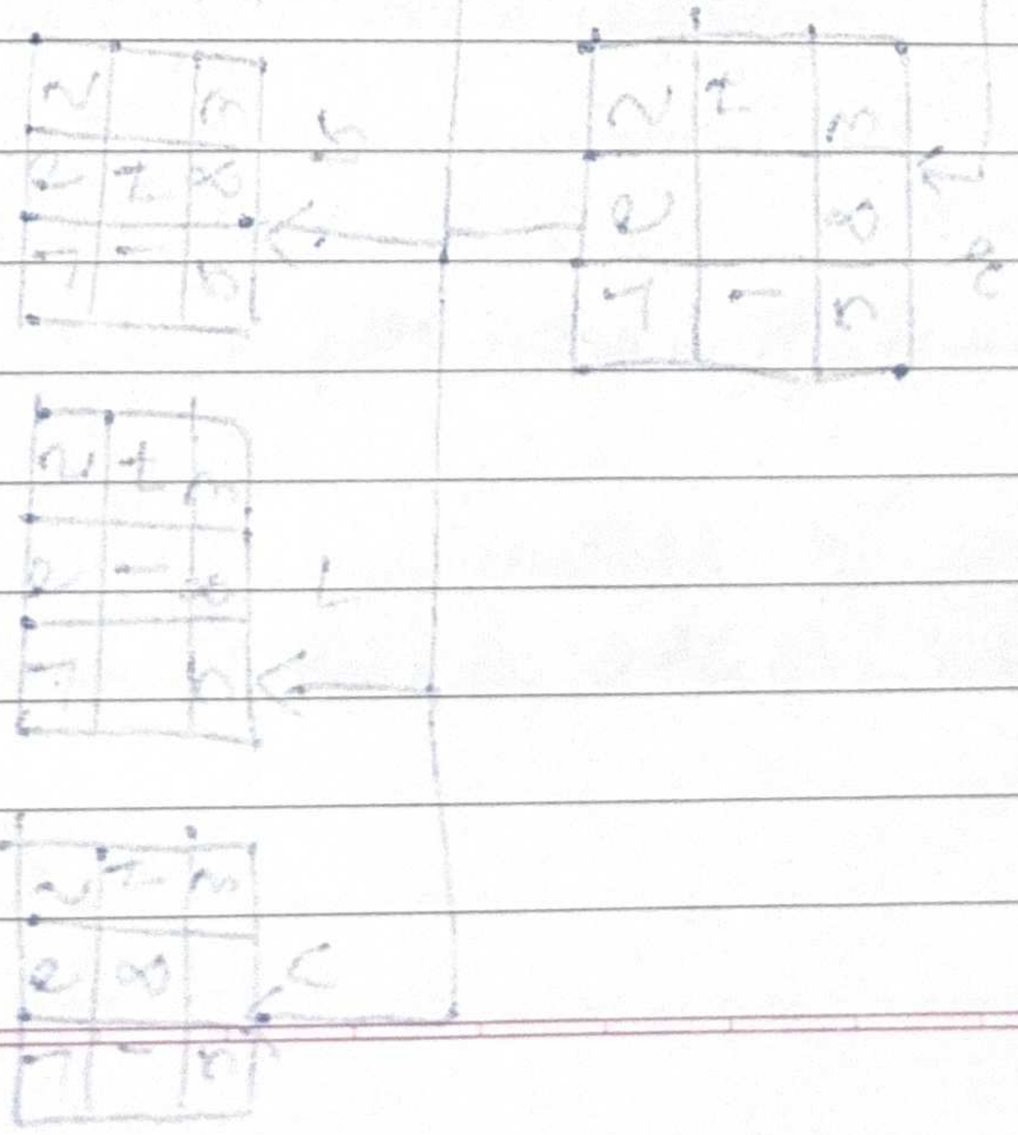
GoodLuck | Page No.
Date 22 | 10 | 24

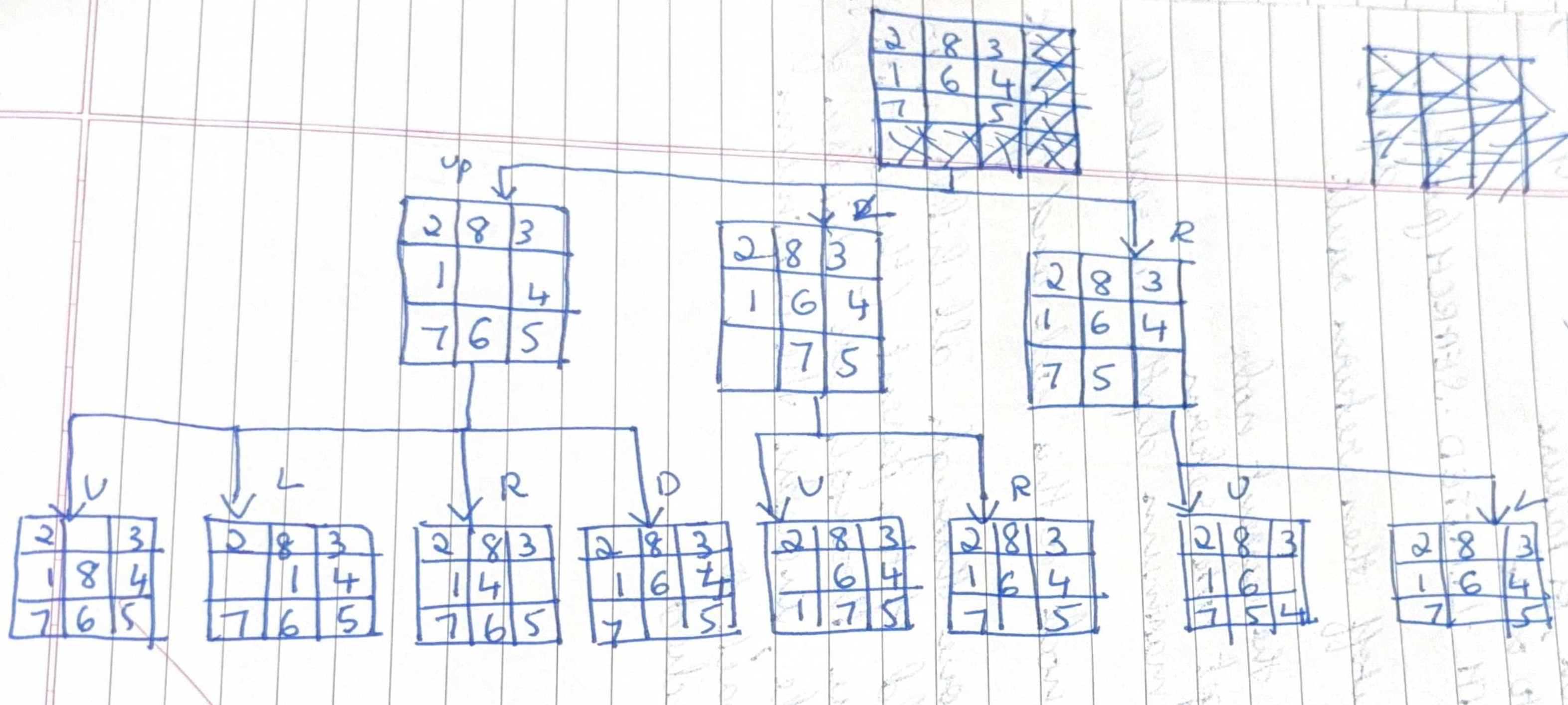
Lab Program : 5.

Implement Iterative Deepening search algorithm.

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.

1. For each child of the current node
2. If it is the target node, return
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 through all children, go to the next child of the parent (the next sibling)
6. After having gone thorough all children of the start node, increase the maximum depth and go back to
7. If we have reached




Up
L
R
D
U
L
R
U
R
U

Lab - 5

Implement Hill Climbing search algorithm to solve N-Queens problem.

function HILL-CLIMBING (problem) returns a state that is a local maximum
  current ← MAKE-NODE (problem.INITIAL-STATE)
  v loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≤ current.VALUE then returns current.STATE current ← neighbor

→ • State: 4 queens on the board. One queen per column
  - Variables: x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column
  - Domain for each variable: $x_i \in \{0, 1, 2, 3\}$, $\forall i$.

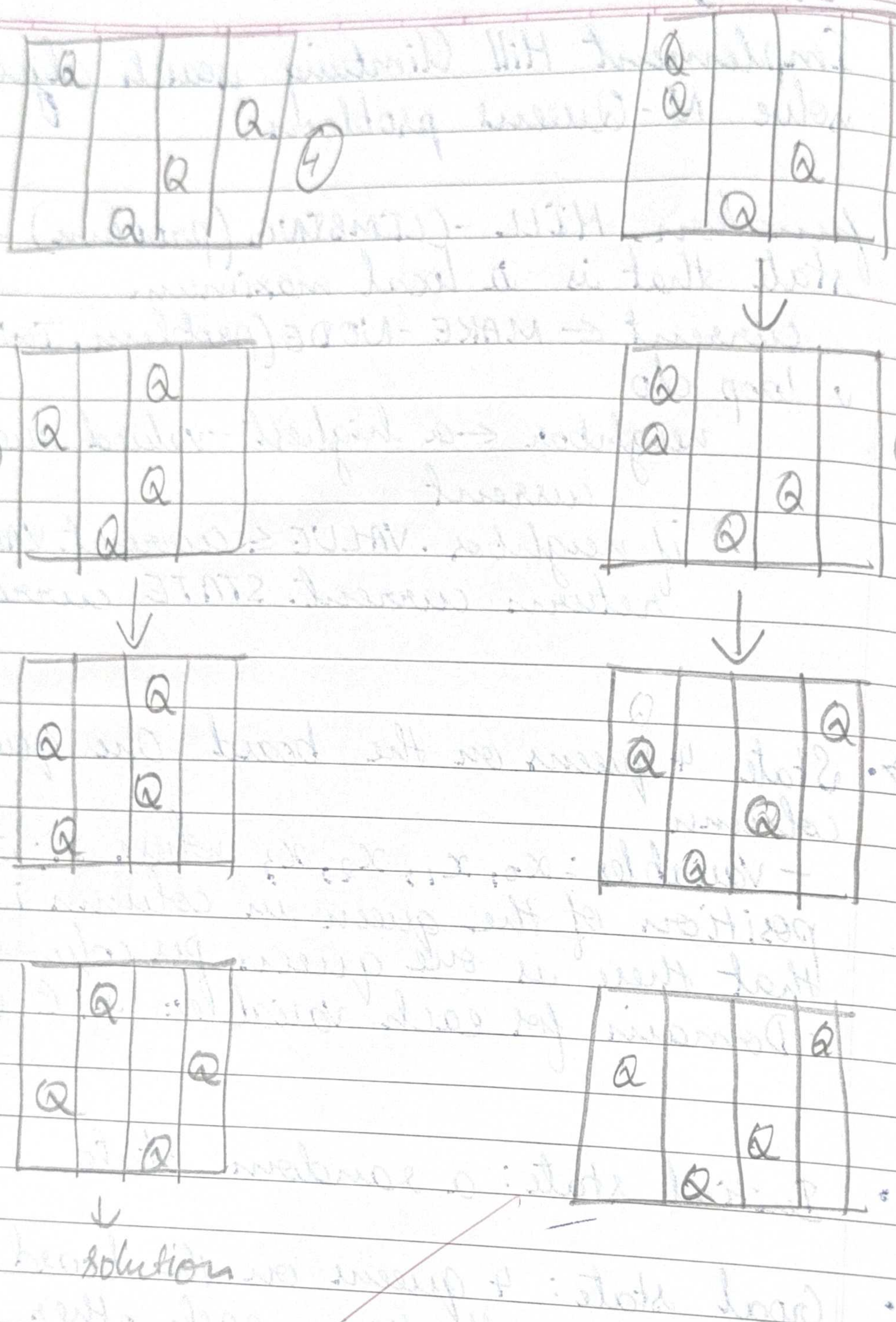
• Initial state: a random state

• Goal state: 4 queens on the board. No pair of queens are attacking each other

• Neighbours relation:
  Swap the row position of two queens

• Cost function: The number of pairs of queens attacking each other, directly or indirectly.

GoodLuck | Page No.
Date



GoodLuck | Page No.
Date 29 | 10 | 24

Lab - 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 ← next.VALUE - current.VALUE
if ΔE > 0 then current ← next
else current ← next only with probability $e^{\Delta E/T}$

The Simulated Annealing Algorithm
The alg can be decomposed in 4 simple steps

1. Start at a random point x
2. Choose a new point x_j on a neighbourhood N(x)
3. Decide whether or not to move to the new point x_j. The decision will be made based on the probability function $P(x, x_j, T)$

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

4. Reduce T

Output

8 queens

The best position found is [0 8 5 2 6 3 7 4]

The no of queens that are not attacking each other is 0

MST

Edges in Minimum Spanning Tree

0 -- 2 (weight = 1)

2 -- 3 (weight = 3)

2 -- 1 (weight = 2)

Total weight

Sudoku using simulated annealing

[[5 3 4 6 7 8 1 9 2
6 7 2 1 9 5 8 3 4
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 8 3
2
]]

GoodLuck | Page No.
Date 12 | 11 | 24

Lab - 6

Program - 6

Implementation of Truth-Table enumeration algorithm for deciding propositional entailment i.e., create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

Algorithm:

function T-T-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 propositional 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)
      ∪ {P = true})

and
TT-CHECK-ALL(KB, α, rest, mode U
{P = false4})

=> Propositional Inference Enumeration Method.

α = AVB

KB = (AVC) ∧ (BV ¬C)

A	B	C	AVC	BV¬C	KB	α
false	false	false	false	true	false	false
false	false	true	true	false	false	false
false	false	false	false	true	false	true
false	~~false~~ true	false	false	true	true	true
false	True	true	true	true	true	True
True	~~false~~ false	false	true	true	false	true
True	false	true	true	false	false	true
True	~~false~~ true	false	true	true	true	true
True	true	true	true	true	true	true

Combination where both KB and α (AVB) are true:

A	B	C
0	1	1
1	0	0
1	1	0
1	1	1

GoodLuck | Page No.
Date 19 | 11 | 24

Lab - 7

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 $\{(\psi_2 / \psi_1)\}$
- c) Else if ψ_2 is a variable,
  - a. If ψ_2 occurs in ψ_1, then return FAILURE,
  - b. Else return $\{(\psi_1 / \psi_2)\}$.
- d) Else return FAILURE

Step 2 : If the initial Predicate symbol in ψ_1 and ψ_2 are not same, then return FAILURE.

Step 3 : If ψ_1 and ψ_2 have a different number of 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 returne Failure
- c) If S ≠ NIL then do,
  - a. Apply S to the reminder of both L1 and L2
  - b. SUBST = APPEND(S, SUBST).

Step 6 : Return SUBST

Eg :

$P(x, F(y)) \rightarrow (1)$

$P(a, F(g(x))) \rightarrow (2)$

$P(a, F(y)) \rightarrow P(a, F(g(x)))$

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

(1) & (2) are identical if x is replaced initial

Ex → P(x, F(y)) → ①
P(a, F(g(n)) → ②
① and ② are identical if x is replaced with a in ① a/x

P(a, F(y)) → ①
if y is replaced with g(n)
P(a, F(g(n))) → ②

Now ① & ② are same, so they are unified

→ Eg: Eats(x, Apple)
Eats(Riya, y)
x is replaced with Riya
Riya/x
Eats(Riya (Apple)

y is replace with Apple
Apple/y
Eat(Riya, Apple).

→ Eg: g(x, a), f(y)
① Q(a, (n, a), f(y)) → ①
② Q(a, g(f(b), a), x) → ②

replace n with f(n) → ①
① Q(a, g(f(n), a); f(y)

replace x with f(y) → ②
x(f(y)
② Q(a, g(f(n), a), f(y))

Now both are same, they are unified

GoodLuck | Page No.
Date

Lab - 8

Forward Reasoning Algorithm

Q $\psi_1 = P(f(a), g(y))$
$\psi_2 = P(x, x)$

1st one is fail

Q $\psi_1 = P(b, x, f(g(z)))$
$\psi_2 = P(z, f(y), f(a))$.

2nd one is pass

Iteration 1:
Attempting to unify : $P(f(a), g(y))$
& $P(x, x)$
current substitution : empty.

Iteration 2:
Attempting to unify : $f(a)$ & x
current substitution : empty
Added substitution : $x \rightarrow f(a)$

Attempting to unify : $g(y)$ & $f(a)$
current substitution : $x \rightarrow f(a)$
unification failed : Diff
Predicates and argument length.

GoodLuck | Page No.
Date

Output:
Unifying $P(b, x, f(g(z)))$ with $P(z, f(y), f(y))$
Unifying b with z
Substitution: $b \rightarrow z$
Unifying $f(g(z))$ with $f(y)$
Substitution: $f(g(z)) \rightarrow f(y)$

Final Result:
Unification successful
Substitution: $b \rightarrow z, x \rightarrow f(y), f(g(z)) \rightarrow f(y)$.

19-12-24

GoodLuck | Page No.
Date 26 | 11 | 24

Lab - 8

→ First Order Logic

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 sentence.

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

repeat until new is empty

new ← { }

for each rule in KB do

$(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIAB

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

for some $p_1' \ldots p_n'$ 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.

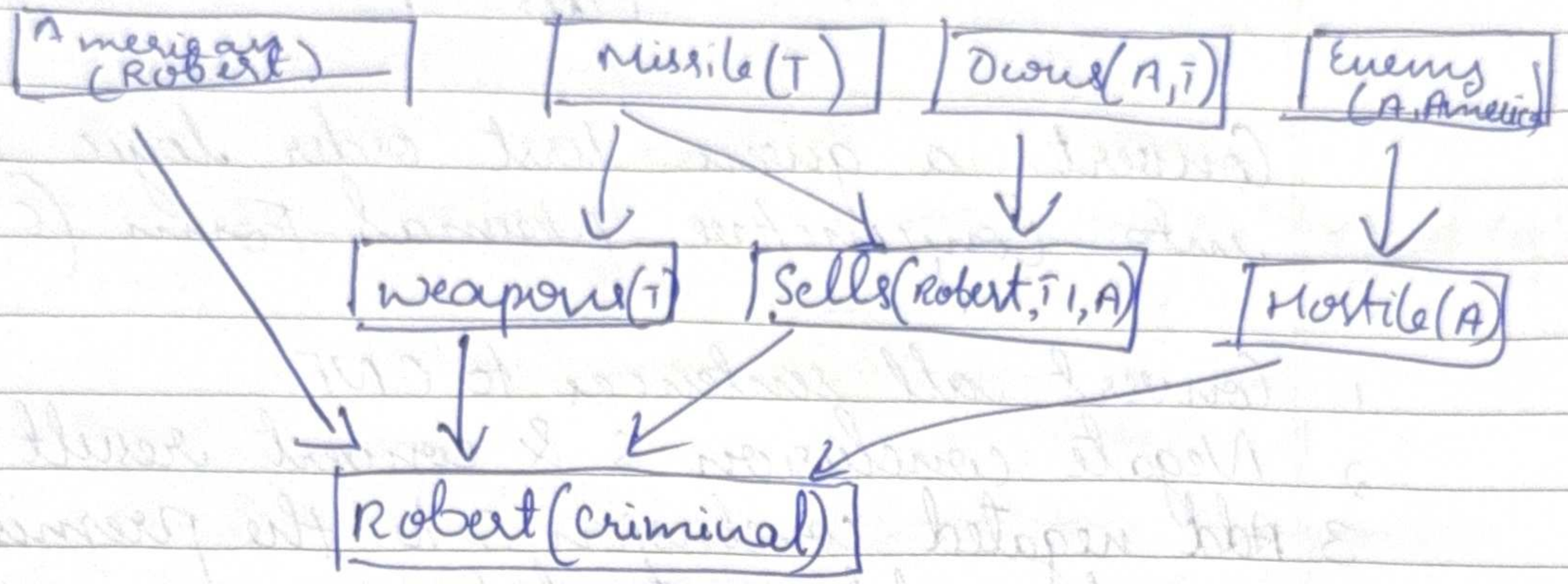
26/11/24

Q4
As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, were sold to it by Robert, who is an American citizen".

Prove that "Robert is criminal"

Representation in FOL

It is a crime for American to sell, weapons to hostile nations
~~Let's say~~
$American(p) \wedge Weapon(q) \wedge Sells(p,r,q) \wedge Hostile(r) \Rightarrow Criminal(p)$
Country A has some missiles
$\exists x \; Owns(A,x) \wedge Missiles(x)$
Existential Instantiation, introducing a new constant T1:
$Owns(A,T1)$
$missile(T1)$
All of the missiles were sold to country A by Robert
$\forall x \; Missiles(x) \wedge Owns(A,x) \Rightarrow Sells(Robert,x,A)$
Missiles are weapons
$Missile(x) \Rightarrow weapon(x)$
Enemy of America is known as hostile
$\forall x \; Enemy(x,America) \Rightarrow Hostile(x)$



Output:

Enter your FOL statements

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

> ∀x owns (A, x) ∧ Missile (x)

> owns (A, T1)

> Missile (T1)

> ∀x Missile (x) ∧ Owns (A, x) ⇒ Sells (Robert, x, A)

> Missile (x) ⇒ Weapon (x)

> ∀x Enemy (x, America) ⇒ Hostile (x)

> American (Robert

> Enemy (A, America)

> done

Enter the query to prove: Criminal (Robert)

Proves: Criminal (Robert

7-11-24

GoodLuck | Page No.
Date 26 | 11 | 24

Lab - 9

Convert a given first order logic statement into Conjunctive Normal Form (CNF)

1. Convert all sentences to CNF
2. Negate conclusion S & 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 ~~resolved~~ 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.

Given KB or Premises:

a. John likes all kind of food
b. Apple and vegetables are food
c. Anything anyone eats and not killed is food
d. Anil eats peanuts and still alive
e. Harry eats everything that Anil eats
f. Anyone who is alive implies not killed
g. Anyone who is not killed implies alive
h. Proof by res that: Representation in FOL

a. $\forall x : food(x) \rightarrow likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall g x \forall y : eats(x, y) \wedge \neg killed(x) \rightarrow food(y)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$
f. $\forall x : \rightarrow killed(x) \rightarrow alive(x)$
g. $\forall x : alive(x) \rightarrow killed(x)$
h. $likes(John, Peanuts)$

Eliminate implication $\alpha \Rightarrow \beta$ with $\neg \alpha \vee \beta$

a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall x \forall y \neg [eats(x,y) \wedge \neg killed(x)] \vee food(y)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$
f. $\forall x \neg [\neg killed(x)] \vee alive(x)$
g. $\forall x \neg alive(x) \vee \neg killed(x)$
h. $likes(John, Peanuts)$.

Move negation ($\neg$) inwards and rewrite

a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall x \forall y \neg eats(x,y) \vee killed(x) \vee food(y)$
d. $eats(Anil, peanuts) \wedge alive(Anil)$
e. $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$
f. $\forall x killed(x) \vee alive(x)$
g. $\forall x \neg alive(x) \vee \neg killed(x)$
h. $likes(John, Peanuts)$.

Rename variables or standardize variables

a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall y \forall z \neg eats(y, z) \vee killed(y) \vee food(z)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall w \neg eats(Anil, w) \vee eats(Harry, w)$
f. $\forall g killed(g) \vee alive(g)$
g. $\forall k \neg alive(k) \vee \neg killed(k)$
h. $likes(John, Peanuts)$.

Drop Universal Quantifiers

a. ¬ food(x) ∨ likes(John, x)
b. food(Apple)
c. food(vegetables)
d. ¬ eats(y, z) ∨ killed(y) ∨ food(z)
e. eats(Anil, Peanuts)
f. alive(Anil)
g. ¬ eats(Anil, w) ∨ eats(Harry, w)
h. killed(g) ∨ alive(g)
i. ¬ alive(k) ∨ ¬ killed(k)
j. likes(John, Peanuts)

Proof by Resolution:

Hence proved

GoodLuck | Page No.
Date

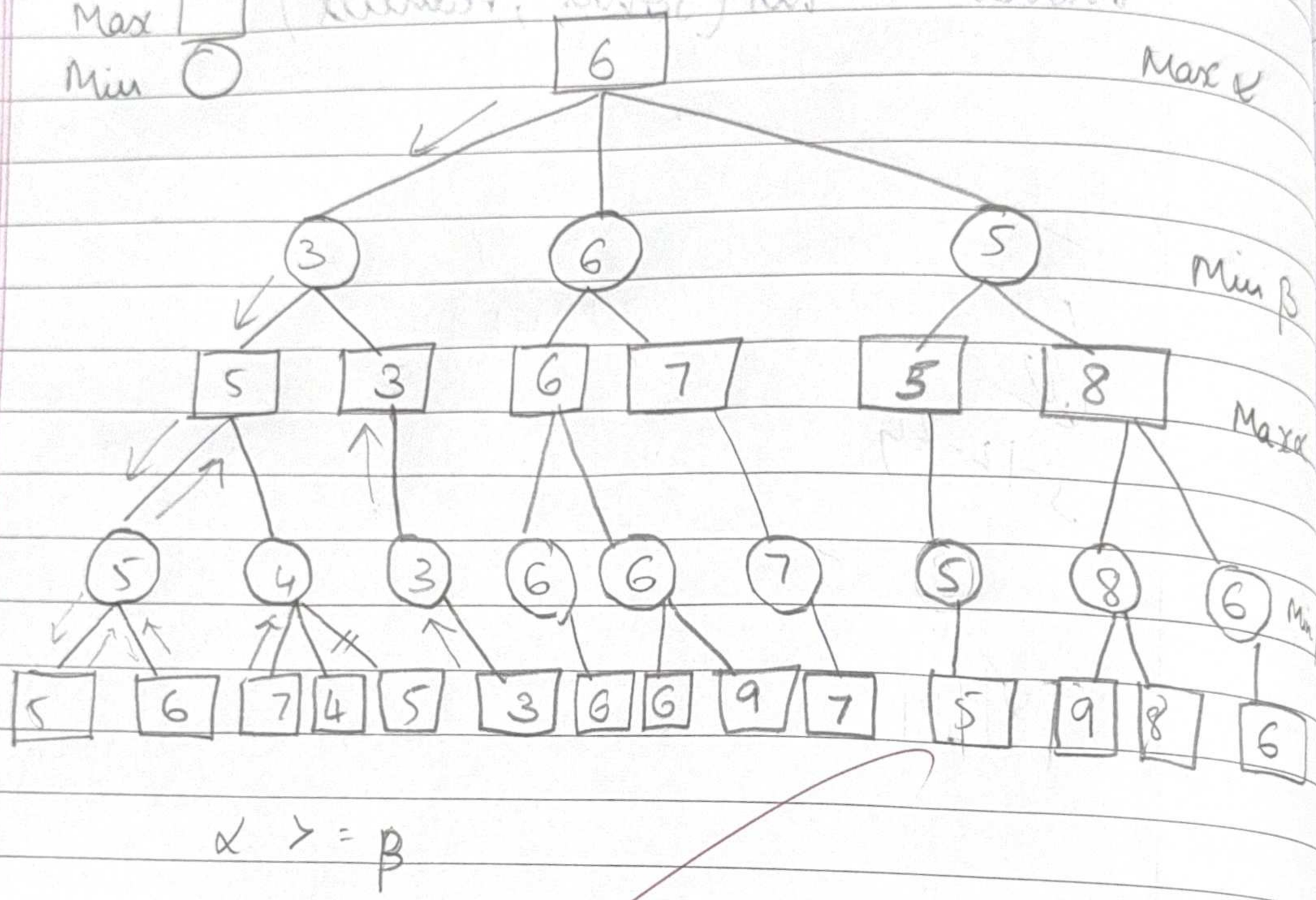
Output:
Derived Fact : food(Peanuts)
Derived Fact : likes(John, Peanuts)
Proven : Likes(John, Peanuts)

3-12-24

GoodLuck | Page No.
Date 26/11/24

Lab - 10.

Implement Alpha Beta Pruning Algorithm.



$\alpha >= \beta$

Algorithm

function MINIMAX-DECISION(state) returns an action

return arg max $a \in$ ACTIONS(s)
MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

$v \leftarrow -\infty$

for each a in ACTIONS(state) do

$v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a)))

return v

function MIN-VALUE(state) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

$v \leftarrow \infty$

GoodLuck | Page No.
Date

for each a in ACTIONS (state) do
    v ← MIN (v, MAX-VALUE (RESULT(s,a)))
return v

Output :-
Enter depth of the tree : 3
Enter leaf values : -1, 8, -3, -1, 2, 1, 3, 4.
Optimal value : 2.