INDEX

Name Shilpa K.M

Standard Section Roll No.

Subject

SL No.	Date	Title	Page No.	Teacher Sign / Remarks
01	24/9/24	Tic - Tac - Toe	01-3	9/10/2024
02	1/10/24	lab - 2 (Vaccum cleaner)	04-6	
03	8/10/24	lab - 3 (8 puzzle DFS, BFS)	7 - 8	
04	15/10/24	8 puzzle by using A*	9 - 12	15/10/2024
05	22/10/24	Iterative deepening Search Algorithm	14 - 15	
06	29/10/24	Simulated Annealing Algo.	16-17	29/10/2024
07	12/11/24	Truth table enumeration Algo	18-19	14/11
08	19/11/24	First order logic (unification)	20-21	19/4
09	26/11/24	Forward reasoning	22	
10	26/11/24	Convert CNF to resolution	23-24	
11	26/11/24	Alpha beta pruning Algo	25-26	3/12/2024

Lab-01

Bafna Gold
Date: Page: 01

Algorithm for tic tac toe

Step 1 : define print board (board)
* Create 2D list array to representing the Tic tac toe grid
* Join the element of each row with " | "
* print the separate line of dashes "---" after each row for clarity

Step 2 :- define check winner (board)
if row[0] == row[1] == row[2] != " "
return row[0]

→ * This checks each row to see if all 3 cells contain same symbol (x/o) and not empty if winner is found, it returns winning symbol.

→ * if board[0][col] == board[1][col] == board[2][col] != " ";
return board[0][col].

→ This checks each column for 3 matching symbol, returning winning symbol if a match is found.

→ * if board[0][0] == board[1][1] == board[2][2] != " "
return board[0][0]

This checks main daigonal for 3 matching symbol & returns if it is found.

→ if board[0][2] == board[1][1] == board[2][0] != " "
return board[0][2]

This checks anti-diagonal for winning symbol & returns it if found

→ return none
This checks if no winning condition is met after all checks it returns none. it indicates there is no winner.

Step 4 :- define tic tac toe ()
* Initialize 3×3 board with empty space
* Set current player to x
* Prompts the current player for row and column input (0, 1, 2)

Step 5 :- checks if selected cell is empty
if occupied print an error message & restart it
if board [row][col] != " "
print ("Invalid move")

Step 6 :- calls check_winner(board) to check for wining condition
* if a winner is found, prints the board and announces the winner, then exists the function.

* current_player = "O" if current_player = "X" else "X"

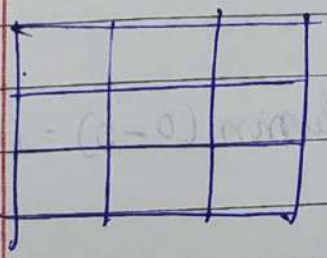
→ Switches the current player for the next turn
* After 9 turns if no winner prints the board and declares a draw

24/9/2024

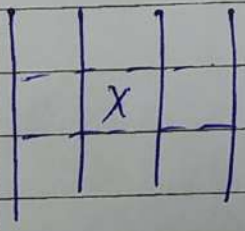
Step 7 : if name == "main"
tictactae()

* calls the tictactoe in main function & then start the game.

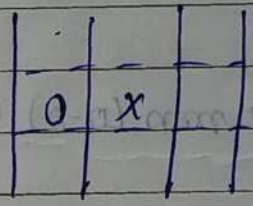
O/P



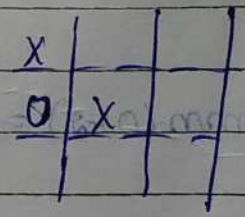
Player X : Enter row and clounm (0-2) = (1, 1)



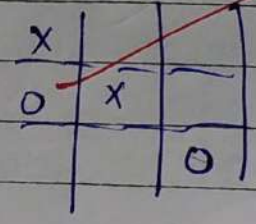
player O enter row & clounm (0-2) = (1, 0)



player X enter row & column (0-2) = (0, 0)



player O enter row & column (0-2) = (2, 2)



player
~~enter~~ ~~the~~ x : enter row and colum (0-2) : (0,2)

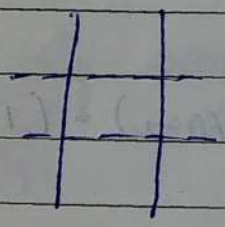
X	-	X
O	X	
		O

player O : enter row & column (0-2) = (0, 1)

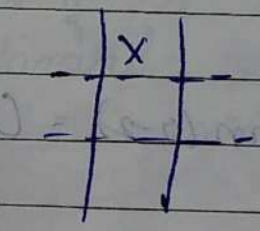
X	X	X
O	X	
		O

X is winning

O/P



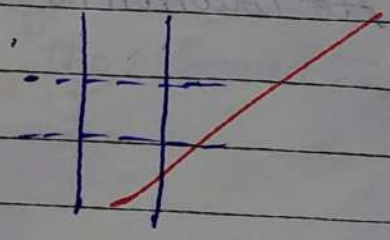
Player X : enter row and column (0-2) = (0, 1)



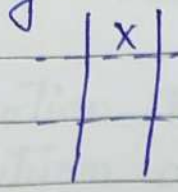
player Y : Enter row and column (0-2) = (0, 3)

Invalid move

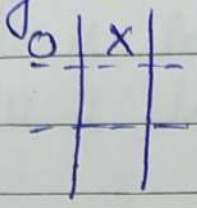
O/P



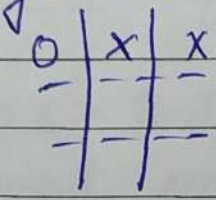
player X: Enter row and column (0-2) = (0,1)



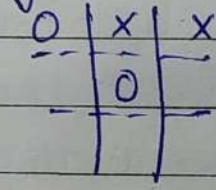
player O: Enter row and column (0-2) = (0,0)



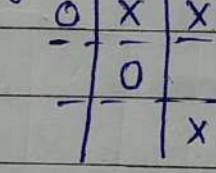
player X: Enter row and column (0-2) = (0,2)



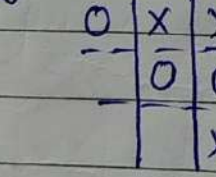
player O: Enter row and column (0-2) = (1,0)



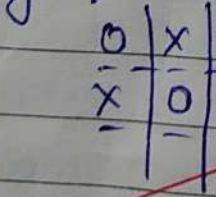
player X: Enter row and column (0-2) = (2,2)



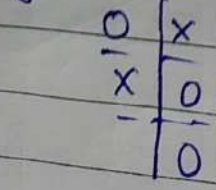
player O: Enter row and column (0-2) = (1,2)



player X: Enter row & column (0-2) = (1,0)



player O: Enter row & column (0-2) = (2,1)



player 0 : Enter row and column (0-2) = (2,0)

O	X	X
X	O	O
O	O	X

No Winner

9/10/2024

Lab-02

Rufna Gold
Page: 4

Lab 2: Implement Vaccum world cleaner

function REFLEX_VACCUM_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

Algorithm for two quadrents

Step 1:- Initialization
* Input current room (either A or B)
* Input the status of the room clean or dirty
* Initialize a cost to 0
Step 2: Display Initial room status
Step 3: cleaning loop
* While either of the room is dirty:
1) if current room is A and is dirty then clean & increase cost by one
2) If current room is B and is dirty then clean and increase cost by one
3) If current room is not dirty move to next room
Step 4: Display cost
Step 5: Stop

Output:
Enter current room either A or B: A
Is room A dirty? (Yes: 1/no: 0): 0
Is room B dirty? (Yes: 1/No: 0): 0

Initial Status of rooms:

Room A : clean
Room B : Dirty

Moving to Room B...
current status:
Room A : clean
Room B : Dirty
cleaning Room B...
current status
Room A : clean
Room B : clean

Both rooms are now clean! Total cost : 1

Output:
Enter current room either A or B : A
Is Room A dirty? (Yes : 1 / NO : 0) : 1
Is Room B dirty? (Yes : 1 / NO : 0) : 0

Initial status of rooms:
Room A : Dirty
Room B : clean

cleaning Room A...
current status:
Room A : clean
Room B : clean

Both rooms are now clean! Total cost : 1

output:
Enter current room eitheir A or B : A

Is Room A dirty? (Yes: 1 / no: 0): 1
Is Room B dirty? (Yes: N / NO: 0): 1

Initial status of rooms:
Room A: Dirty
Room B: Dirty

Cleaning Room A...
current status:
Room A: clean
Room B: Dirty
Moving to Room B...
current status:
Room A: clean
Room B: Dirty
cleaning Room B...
current status:
Room A: clean
Room B: clean

Algorithm for Four Quandrants

Step 1: Start
* Accept status of each room (either clean or dirty)
Step 2: Display the initial status of all rooms
Step 3: cleans rooms
* while all rooms are clean
1) If first room is dirty clean and go to next room
2) If second room is dirty clean and go to next room
3) If third room is dirty clean and go to next room
4) If fourth room is dirty clean and go to next room
5) Increase cost when cleaning and decreases count
Step 4: Show the last status
Step 5: End

Output:

Enter current room (A, B, C, D) : B
Is room A dirty ? (Yes : 1 / NO : 0) : 1
Is room B dirty ? (Yes : 1 / NO : 0) : 0
Is room C dirty ? (Yes : 1 / NO : 0) : 0
Is room D dirty ? (Yes : 1 / NO : 0) : 0

Initial status of rooms :
Room A : Dirty Room B : clean Room C : clean
Room D : clean

Moving to Room C...
moving to Room D...
Moving to Room A ...
cleaning Room A...

Current status
Room A : clean Room B : clean
Room C : clean Room D : clean

All rooms are now clean : Total cost : 1

Output :

Enter current room (A, B, C, D) : A
Is Room A dirty ? 1
Is Room B dirty ? 1
Is Room C dirty ? 1
Is Room D dirty ? 1

Initial rooms status:
Room A : Dirty Room B : Dirty Room C : Dirty
Room D : Dirty

cleaning Room A...
Moving to Room B ...
cleaning Room B...
Vaccum is recharging
Moving to Room C...
cleaning Room C...
Moving to Room D...
cleaning to Room D...

Current status :
Room A : clean Room B : clean Room C : clean
Room D : clean

All rooms are now clean ! Total cost : 4

9/10/2024

Solve 8 puzzle problems using DFS and BFS

State Space diagram:

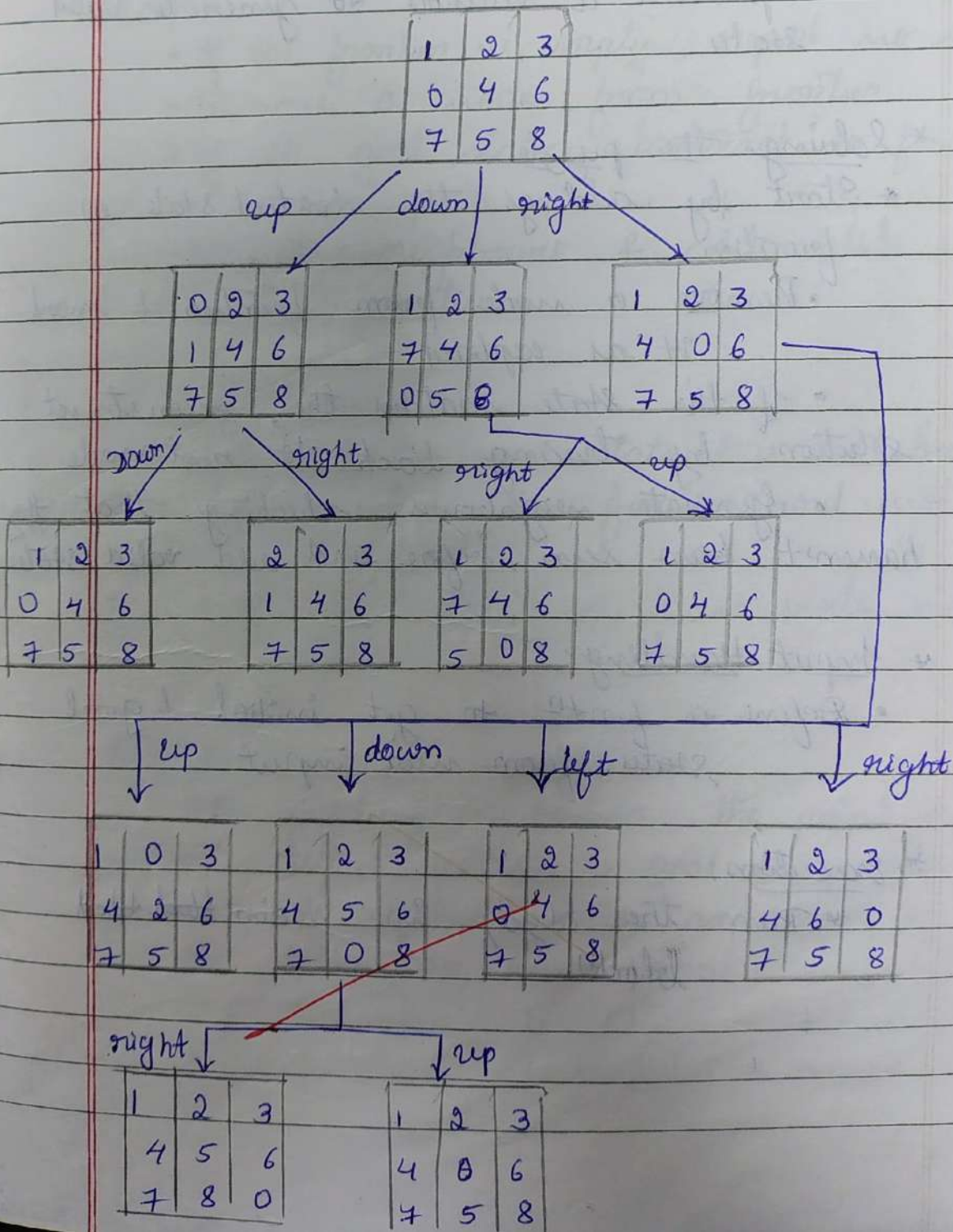
1	2	3
0	4	6
7	5	8

Start State

⇒

1	2	3
4	5	6
7	8	0

Goal state



Algorithm for 8 puzzle problems by using BFS

* Node Stack frontier:
  * Implement a Stackfrontier class to manage the frontier of states using Stack.
  *
* Puzzle class:
  * Initialize with start and goal States
  * Implement a method to generate valid States

* Solving the puzzle:
  * Start by adding the initial state of frontier
    * Remove a node from frontier & mark it as explored
    * If the state matches the, reconstruct solution by tracing back to root node
    * Generate neighbours, checking that they haven't been seen before and add valid state

* Input Handling:
  * Define a funtn to get initial & goal state from user input

* Execution;
  * Run the algo/ in main ~~block~~ blocks

Algorithm for 8 puzzles problems by using DFS

* Input : Read the start and goal states from users
*
* Initialization : Set up the start node and frontier.
* Loop:
  - If the frontier is empty, report no solution
  - Remove a node from frontier
  - If its goal state, backtrack to find the solution path.
  - Generate neighbours & add valid new states to frontier.
* Solution process:
  - check if the new state has not been seen before [not in the frontier and not explored].
  - If valid create a new node and add it to frontier.
* Output:
  - After finding a solution, the print_solution method output initial & goal states. the number of explored states and series of moves taken to reach goal.

9/10/2024

Lab - 4

15/10/23

For 8 puzzle problem using A* star implementation 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)

ⓑ g(n) = depth of a node
h(n) = Heuristic value
⇓
Manhattan distance
f(n) = g(n) + h(n)

@ Number of misplaced tiles
Draw the state space diagram for

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5

g(n) = 0 h(n) = 4. goad state

2	8	3
1	6	4
7		5

L U R

2	8	3
1	6	4
	7	5

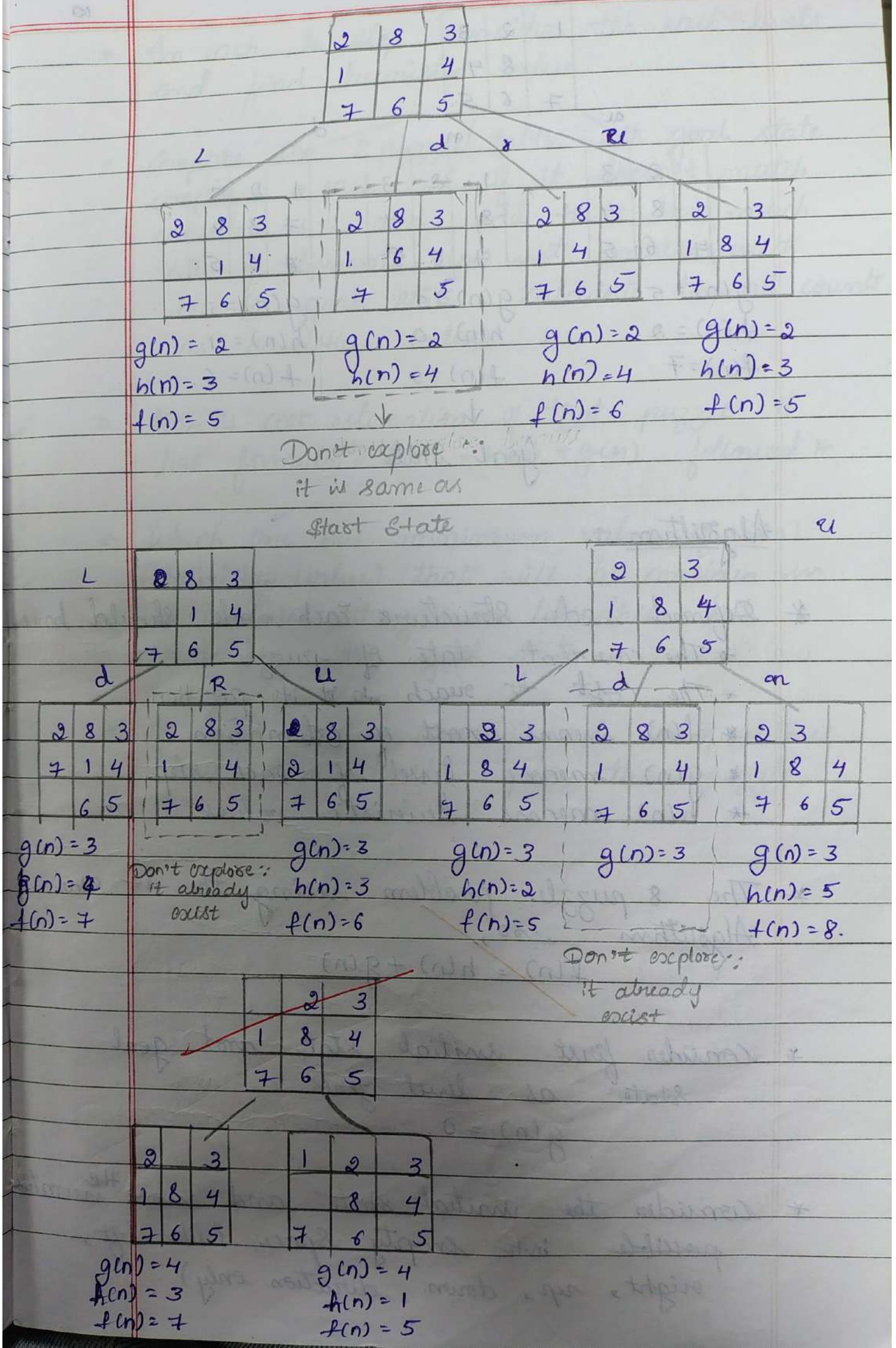
g(n) = 1
h(n) = 5
f(n) = 1+5 = 6

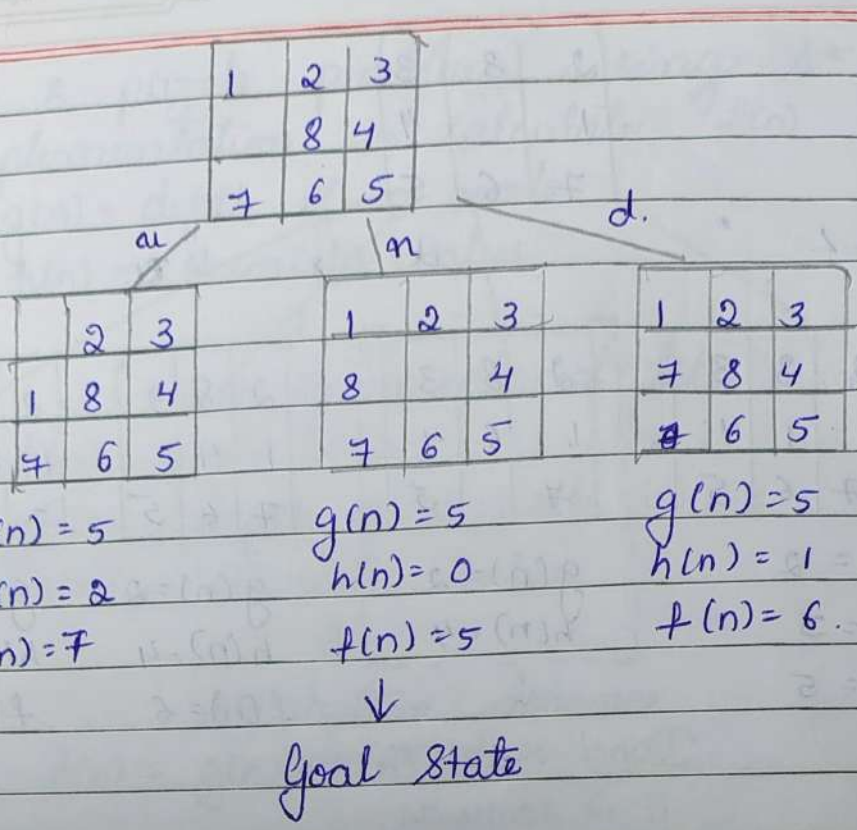
2	8	3
1		4
7	6	5

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

2	8	3
1	6	4
7	5	

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


2 8 3
1 4
7 6 5
L
d
R
g(n) = 2
h(n) = 3
f(n) = 5
g(n) = 2
h(n) = 4
Don't explore ∴ it is same as Start State
g(n) = 2
h(n) = 4
f(n) = 6
g(n) = 2
h(n) = 3
f(n) = 5
L
d
R
U
g(n) = 3
g(n) = 4
f(n) = 7
Don't explore ∴ it already exist
g(n) = 3
h(n) = 3
f(n) = 6
L
d
U
g(n) = 3
h(n) = 2
f(n) = 5
Don't explore ∴ it already exist
g(n) = 3
h(n) = 5
f(n) = 8.
g(n) = 4
h(n) = 3
f(n) = 7
g(n) = 4
h(n) = 1
f(n) = 5



Goal State

Algorithm :

* Defined node structure Each node should have
  * The current state of puzzle
  * The cost to reach to it its position
  * f(n) means cost of estimation
  * g(n) means level of each step
  * h(n) means heuristic values.

* The 8 puzzle problem using the A* Star Algorithm . so,

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

* Consider first initial state and goal state as level zero

$$g(n) = 0.$$

* Consider the initial state and move the number into empty space. (like left, right, up, down direction only)

* In each & step consider the each levels and find heuristic value.

* Compare the 8 puzzle table with goal state compare @ each cell if it doesn't match consider as 1 and add the 1 How much cells of elements does not match with cell of goal state. calculate the all counts as heuristic value.

* f(n) is cost estimation of that puzzle it has formula $f(n) = h(n) + g(n)$ followed #

* which one has minimum value of h(n). [heuristic value] that will be consider in next level, it goes on like this till reaching the goal state

* If any puzzles already exist in previous level. then don't explore those puzzle

(b) Manhattan distance

2	8	3
1	6	4
7		5

Initial state

1	2	3
8		4
7	6	5

goal state

$g(n) = 0$ $h(n)$ 1 2 3 4 5 6 7 8.

$1+1+0+0+0+1+0+1 = 4$

2	8	3
1	6	4
7		5

l u r

2	8	3
1	6	4
	7	5

$g(n) = 1$

$h(n) = 1+1+1+1+2$

$h(n) = 6$

$f(n) = 7$

2	8	3
1		4
7	6	5

$g(n) = 1$

$h(n) = 1+1+2$

$h(n) = 4$

$f(n) = 5$

2	8	3
1	6	4
7	5	

$g(n) = 1$

$h(n) = 1+1+1+1+2 = 6$

$f(n) = 7$

2	8	3
1		4
7	6	5

L u d r

2	8	3
	1	4
7	6	5

$g(n) = 2$

$h(n) = 2+1+2$

$= 5$

$f(n) = 7$

2		3
1	8	4
7	6	5

$g(n) = 2$

$h(n) = 1+1+1$

$h(n) = 3$

$f(n) = 5$

2	8	3
1	6	4
7		5

$g(n) = 2$

$h(n) = 1+1+1+2$

$h(n) = 5$

$f(n) = 7$

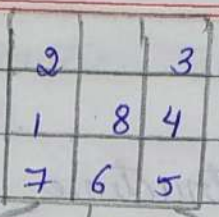
2	8	3
1	4	
7	6	5

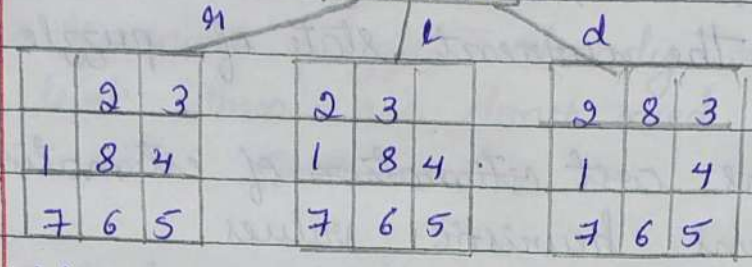
$g(n) = 2$

$h(n) = 1+1+1+2$

$= 5$

$f(n) = 7$

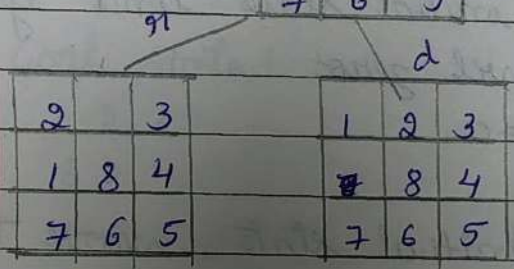




$g(n) = 3$
$h(n) = 1+1 = 2$
$f(n) = 5$

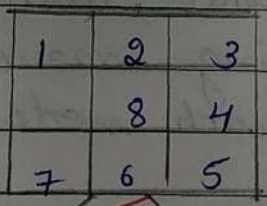
$g(n) = 3$
$h(n) = 1+1+1 = 3$
$f(n) = 6$

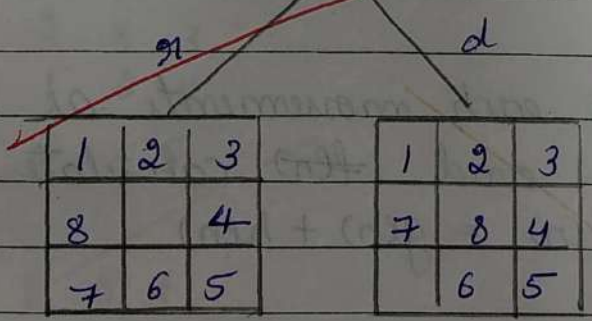
$g(n) = 3$
$h(n) = 1+1+2 = 4$
$f(n) = 7$



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

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





$g(n) = 5$
$h(n) = 0$
$f(n) = 5$
↑
goal State

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

Algorithm:

* Defined node structure each node should have the current state of puzzle

g(n) means cost ~~estimation~~ of estimation
h(n) means heuristic values
f(n) = g(n) + h(n).
g(n) = means level of each ~~puzz~~ step

* The 8 puzzle problem using the A* Algorithm
f(n) = g(n) + h(n)

* Consider first initial state and goal state as level zero
g(n) = 0

* Consider the Initial state. move the possible nodes into empty space.
(like left, right, up, down direction)

* In Manhatten distance, calculate heuristic value as is comparing moves of each node ~~node~~ with ~~go~~ each node of goal state.

* we count the each movements as ~~two~~ heuristic value and ~~f(n)~~ calculate the f(n) = g(n) + h(n)

* Consider the minimum value of h(n) as to calculate the 8 puzzles problem

* It goes iteration till it reaches the goal State.

* If any puzzles already exit in previous level. then we don't need it explore that State

Output: of misplaced tiles

Initial state:

2 8 3
1 6 4
7 0 5

Goal state:

1 2 3
8 0 4
7 6 5

Solution found with cost: 5

steps:

2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
8 0 4
7 6 5

Output for Manhatten

Enter Initial State
2 8 3
1 6 4
7 0 5

Enter goal State
1 2 3
8 0 4
7 6 5

level 1
2 8 3
1 0 4
7 6 5

level 1
2 8 3
1 6 4
0 7 5

level 1
2 8 3
1 6 4
7 5 0

level : 2

2 8 3
1 6 4
7 5 0

level : 2

2 8 3
1 6 0
7 5 4

level : 2

2 0 3
1 8 4
7 6 5

level : 3

2 3 0
1 8 4
7 6 5

level 4

2 0 3
1 8 4
7 6 5

level 5

1 2 3
8 0 4
7 6 5

15/10/2024

Lab - 5

22/10/24

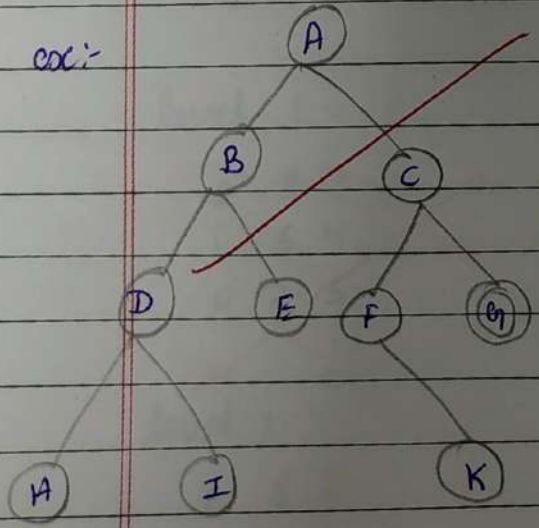
Implement Iterative Deepening Search Algorithm

Algorithm:-

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

ex:-



1st iteration → A
2nd iteration → A, B, C
3rd iteration → A, B, D, E, C, F, G
4th iteration → A, B, D, E, H, I, E, F, K, G

Initial State

1	4	3
7		6
5	8	2

Goal State

1	4	3
7	6	2
5	8	

level 0

up left Down right

level 1

1		3
7	4	6
5	8	2

1	4	3
	7	6
5	8	2

1	4	3
7	8	6
5		2

1	4	3
7	6	
5	8	2

left right up down left right

level 2

	1	3
7	4	6
5	8	2

	1	3
7	4	6
5	8	2

	4	3
1	7	6
5	8	2

1	4	3
7		6
5	8	2

1	4	3
7	8	6
	5	2

1	4	3
7	8	6
5	2	

up down

1	4	
7	6	3
5	8	2

1	4	3
7	6	2
5	8	

goal state

exit

22/10/24

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)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.value ≤ current.value then
            return current.State
        current ← neighbor

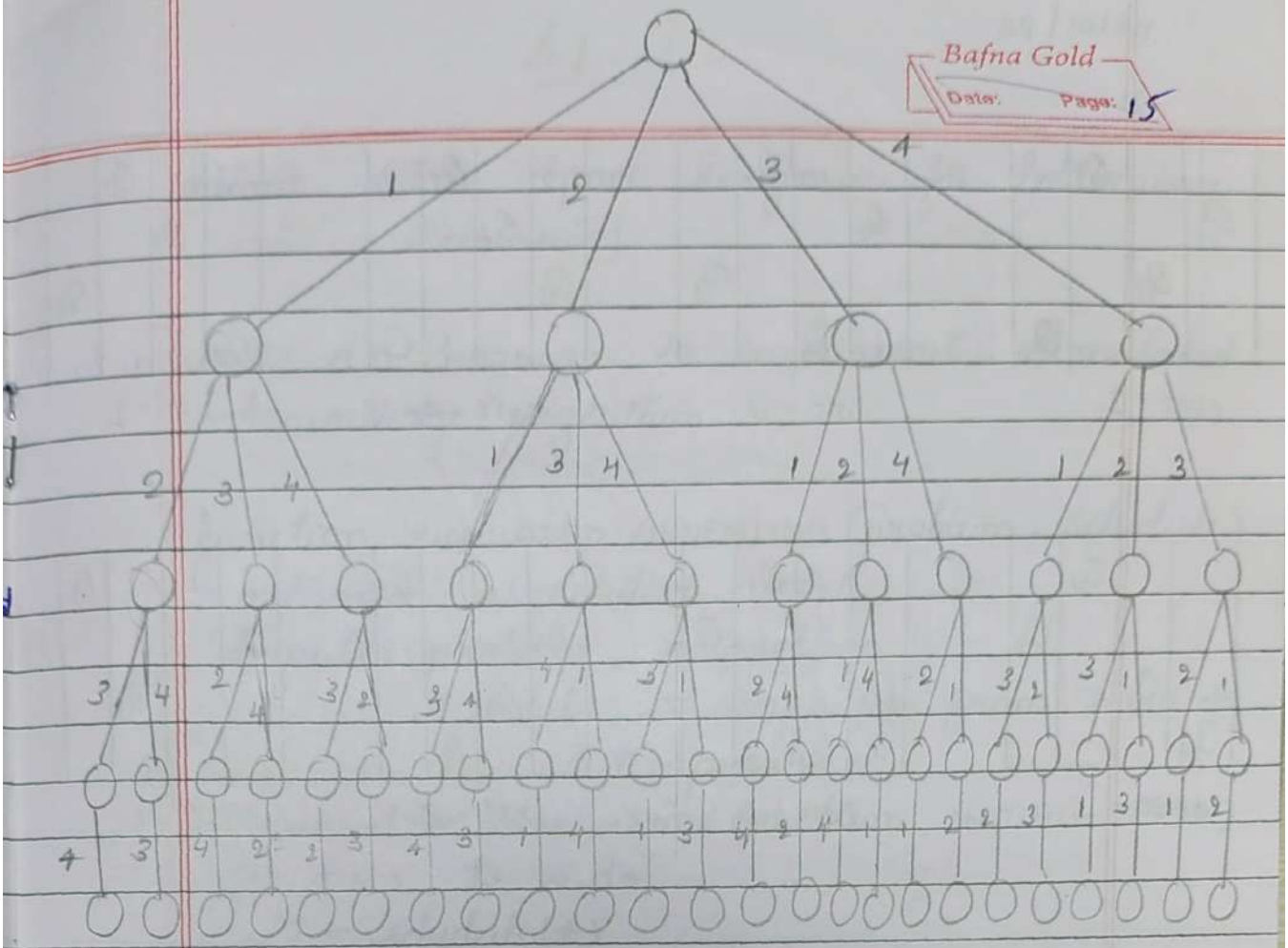
* State: 4 queens on the board. One queen per column.
  - variables: x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column.
  - Domain for each variable: $x_i \in \{0,1,2,3\}, \forall i$.

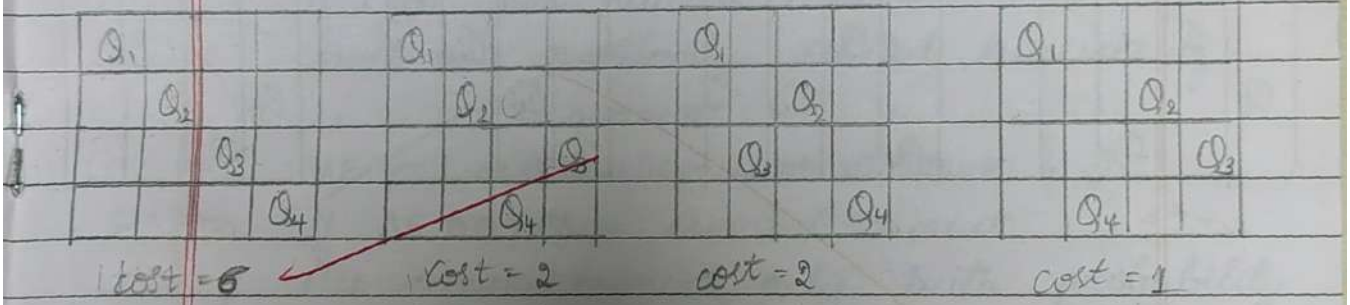
* Initial State: a random state

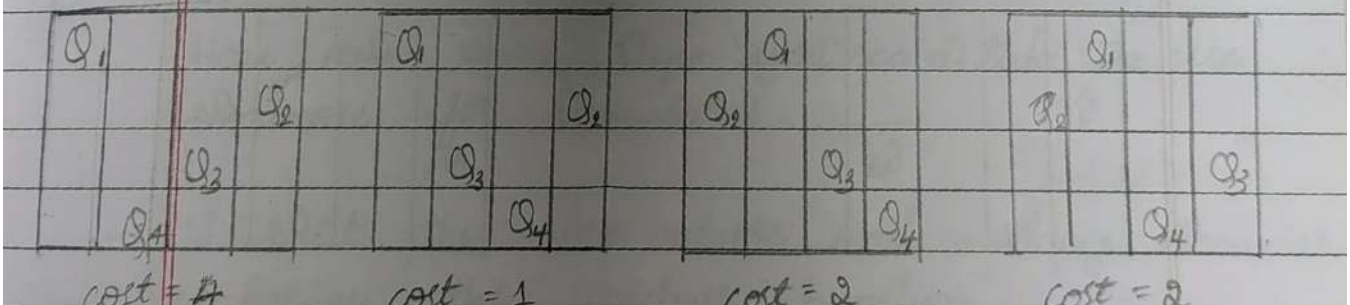
* Goal state: 4 queens on the board. No pair of queens are attacking each other.

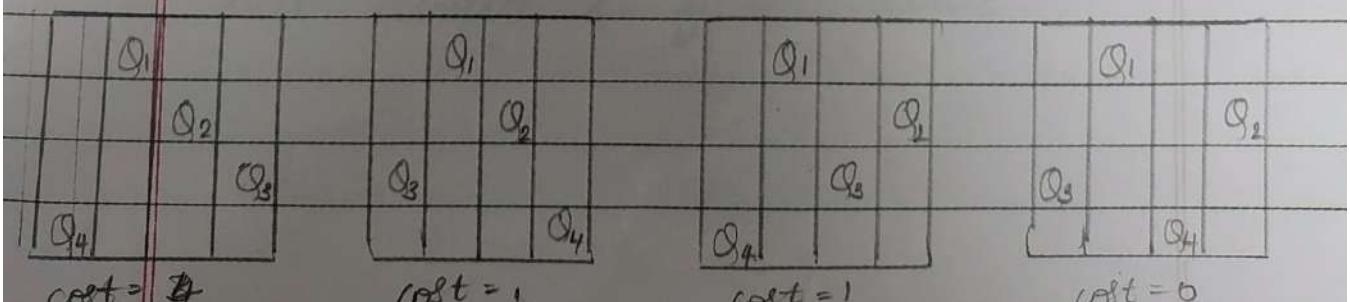
* Neighbour relation:
  Swap the row positions of two queens

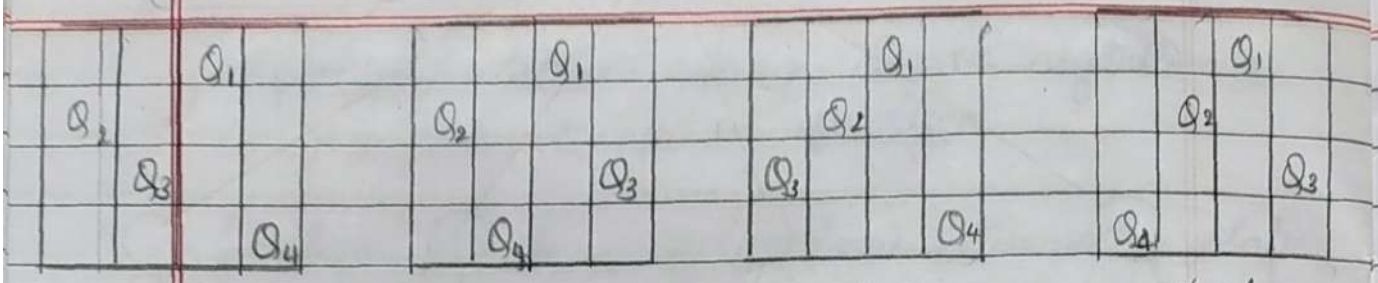
* cost function: The number of pairs of queens attacking each other, directly & Indirectly



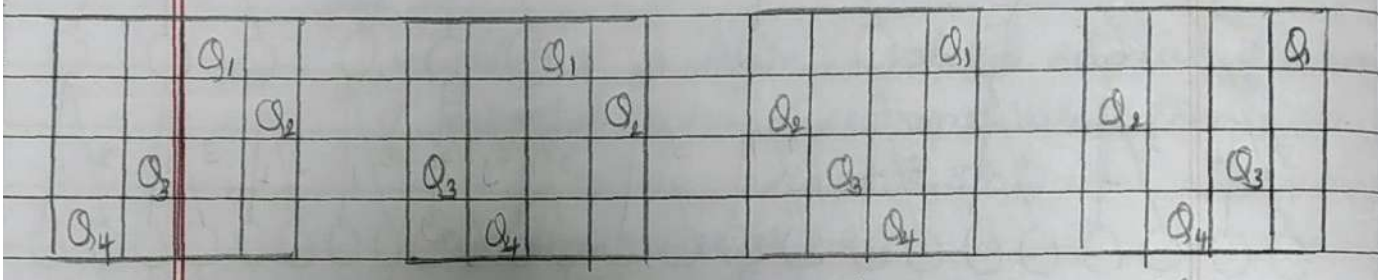

cost = 6
cost = 2
cost = 2
cost = 1


cost = 4
cost = 1
cost = 2
cost = 2

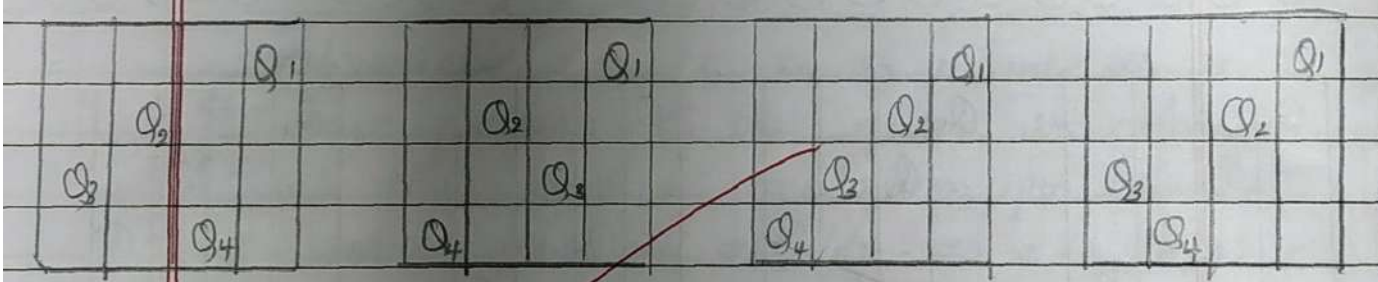

cost = 4
cost = 1
cost = 1
cost = 0



cost = 1 cost = 0 cost = 4 cost = 1

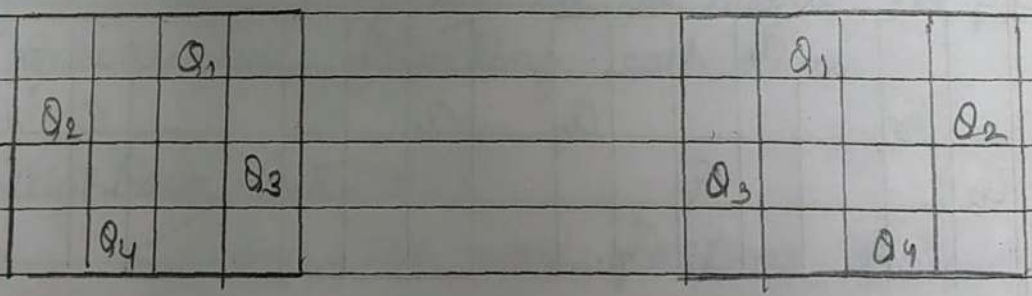


cost = 2 cost = 2 cost = 4 cost = 1



cost = 1 cost = 2 cost = 6 cost = 2

Best Solution :-



Lab - 06

29/10/24

Bafna Gold
Date: Page: 16

Draw State Space Diagram for following Scenario

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 $e^{\Delta E/T}$

Here, are some Steps you can take to use mlrose for Simulated

1. Import the mlrose and numpy libraries
2. Define the objective function.
3. use the mlrose algorithms to get best state, best fitness & fitness curve

The Stimulated Annealing Algorithm can be decomposed in 4 Simple Steps

1. Start at a random point x.
2. Choose a new point x_j on a neighbour $N(x)$.
3. Decide whether or not to move to 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

Python code

```
import mlrose_hiive as mlrose
import numpy as np

def queens_max(position):
    queen_no_attacking = 0
    for i in range(len(position) - 1):
        no_attack = 0
        for j in range(i+1, len(position)):
            if (position[j] != position[i]) and
               (position[j] != position[j-i]) and
               (position[j] != position[i] - (j-i)):
                no_attack += 1
            if no_attack == len(position) - 1 -
                            i:
                queen_no_attacking += 1
    return queen_no_attacking
```

```
try:
    user_input = input("Enter the initial position
    as 8 comm-separated integers: ")
    initialpos = np.array([int(x) for x in user_input
                          .split(',')])
    if len(initialpos) != 8 or any(x<0 or x>=8
        for x in initialpos):
            raise ValueError("Please enter exactly 8
            integer between 0 & 7.")
except ValueError as e:
    print(e)
    exit()
objective = mlrose.CustomFitness(queen_max)
problem = mlrose.DiscreteOpt(length=8, fitness_fn
    =objective, maximize=True, max_val=8)
T = mlrose.ExpDecay()

result = mlrose.simulated_annealing(problem=
    problem, schedule=T, max_attempts=500,
        max_iter=5000, init_state=initialpos)

best_state = result[0]
best_fitness = result[1]

print("The best position found is:", best_state)
print("The no. of queens that are not attacking
        each other is:", best_fitness)
print("\n Best state Diagram:")
board = [['.' for _ in range(8)] for _ in range(8)]
for row, col in enumerate(best_state):
    board[col][row] = 'Q'
```

```
for row in board:
    print(" ", join(row))
```

Output

Enter the initial position as 8 comma-separated integers : 1, 2, 4, 6, 7, 5, 3, 0.
The best position found is : [3 7 0 2 5 1 6 4]
The no. of queens that are not attacking each other is : 7.0

Best State Diagram :

```
- - Q - - - - -
- - - - - Q - -
- - - Q - - - -
Q - - - - - - -
- - - - - - - Q
- - - Q - - - -
- - - - - Q - -
- Q - - - - - -
```

29/10/2024

Lab - 06

12/11/24

Bafna Gold
Date: Page:

Implementation of truth table enumeration algorithm for deciding propositional

TT - Truth table
PL - True? returns true if sentence holds within a model.
The variable model represents a partial model - an assignment to some of symbols.
The keyword "and" is a used here as a logical operator on its two arguments, returning true or false.

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 propositional symbol in KB and α
return TT-CHECK-ALL(KB, α, symbol, {})

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, model ∪ {P = false}))

Example
α = A ∨ B KB = (A ∨ C) ∧ (B ∨ ¬C)
KB ⊨ α

A	B	C	A ∨ C	B ∨ ¬C	KB
false	false	false	false	true	
false	false	true	true	false	
false	true	false	false	true	
false	true	true	true	true	
true	false	false	true	false	
true	false	true	true	false true	
true	true	false	true	true	
true	true	true	true	true	

KB	α
false	false
false	false
true	true
true	true
false	true
true	true
true	true

Output

A	B	C	KB	Alpha	KB entails Alpha?
false	false	false	false	false	NO
false	false	true	false	false	NO
false	false	false	false	true	NO
false	false	true	true	true	Yes
true	false	false	true	true	Yes
true	false	true	false	true	NO
true	true	false	true	true	Yes
true	true	true	true	true	Yes

Does KB entails alpha ? True

14/11/2024

Lab-07

19/12/24

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 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 & Ψ_2 are not same, then return FAILURE.

Step 3 : If Ψ_1 & 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 number of elements in Ψ_1

(a) call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, & put result into S.

(b) If S = failure then returns FAILURE

(c) If S ≠ NIL then do,

a. Apply S to the remainder of

both L1 & L2
b. SUBST = APPEND(3, SUBST)

Step 6 : Return SUBST

* example 1:

P(X, F(Y)) - ①
P(a, F(g(X)) — ②

① & ② are identical if X is replaced with a in ①
a/X

P(a, F(Y)) — ①

if Y is replaced with g(X)
P(a, F(g(X)) - ①.

Now, ① & ② are equal so it is give as output "TRUE"

* example 2:
Q(a, g(X, a), f(y)) - ①
Q(a, g(f(b), a), X) - ②

if X is replaced with f(b) in ①. f(b)/X

Q(a, g(f(b), a), f(y)) — ①

~~if f(y) is replaced with X in ① ;~~
~~Q(a, g(f(b), a), X) — ①. X/f(y)~~

~~now, ① & ② are equal~~

Cannot able to replace with x in ① because the same variable ca didn't replaced twicely

so, it is not unified. it is FAILURE

* example 3 =

$\psi_1 = p(b, x, f(g(z)))$ —①
$\psi_2 = p(z, f(y), f(y))$ —②

if b^z is replaced with b in ②

$\psi_2 = p(b, f(y), f(y))$ —②

if x is replaced with f(y) in ①

$\psi_1 = p(b, f(y), f(g(z)))$ —①

if y is replaced with g(z) in ②

$\psi_2 = p(b, f(y), f(g(z)))$ —②

Now, ① & ② are equal so, it is unified

* example 4 =

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

x cannot replaced twice so it is failure.

Output :

Enter the first term : ['p', 'b', '?x', ['f', ['g', '?z']]]
Enter the second term : ['p', '?z', ['f', '?y'], ['f', '?y']]
unification Successfull !
Substitution : {'?z': 'b', '?x': ['f', '?y'], '?y': ['g', '?z']}

Enter the first term : ['p', ['f', ['a']], ['g', ['?y']]]
enter the second term : ['p', '?x', '?x']
unification failed : Predicate Symbols don't matth : g != f

8/11/21

Lab-08

26/11/24

Routing Information Protocol [RIP]-

Topology:-

Create KB consisting of FOL statements & parse the given query using forward reasoning

Algorithm:-

function FOL-FC-ASK(KB, α) returns a substitution or false
input: KB, the knowledge base, a set of first order definite clauses α, the query, an atomic sentence.
local variable: new, the new sentence inferred on each iteration
repeat until new is empty
new ← {}
for each rule in KB do
$(p_1 \wedge \cdots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES (rule)
for some $p_1' \cdots p_n'$ 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

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)
Proven : Criminal(Robert)

Example :-

It is a crime for an American to sell weapons to hostile nations
Let's say p, q and r are variable
~~American~~
Country a has some missiles.
&
Existential Instantiation, Introducing a new constant T1 :

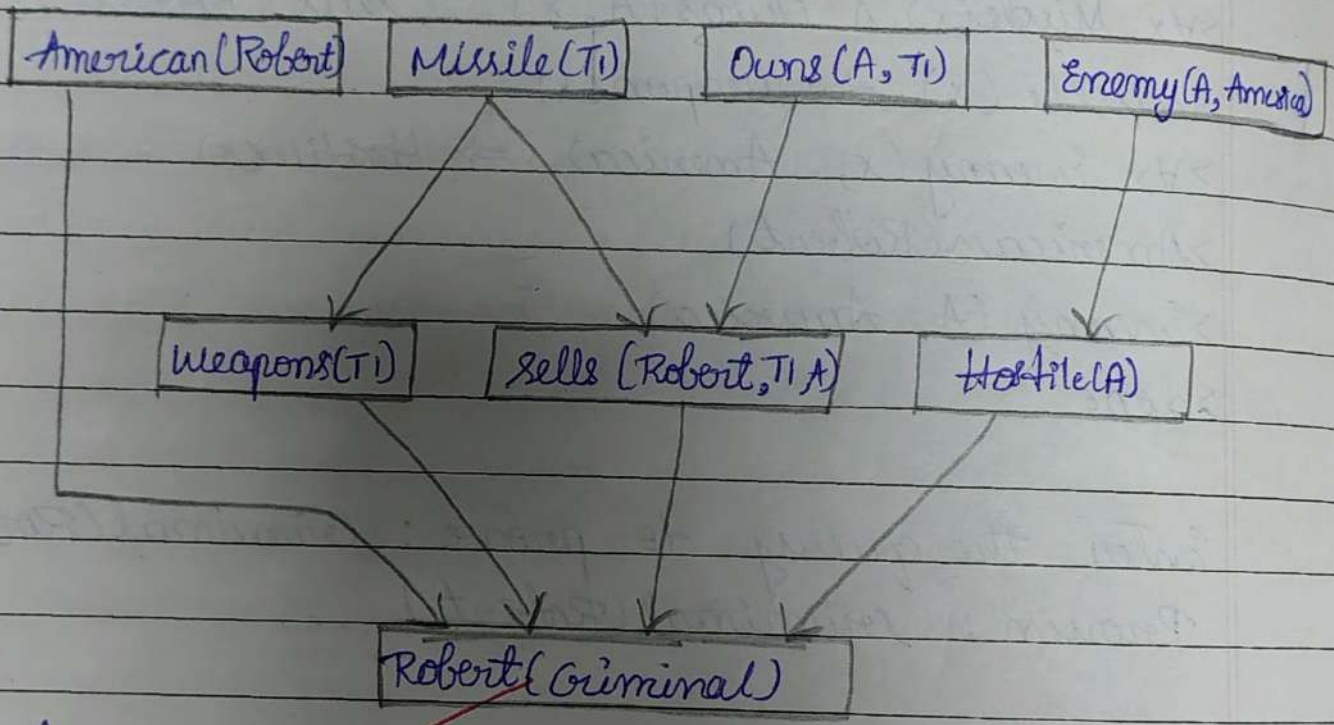
All of the missiles were sold to country A by Robert

missiles are weapons

Enemy of America is known as hostile

Robert is an American

The country A, an enemy of America.



American(p) ∧ weapon(q) ∧ sells(p, q, x) ∧ Hostilion(x)
⇒ Criminal

3/12/2024

26/11/24

Lab - 09

Bafna Gold
Date: Page: 23

Convert a given first order logic Statements into conjunctive normal form (CNF) to Resolution

Basic steps for providing a conclusion S given premises

Premises, ..., Premisen

Call expressed in FOL):

1. Convert all Sentences to CNF
2. Negate conclusion S & convert result to CNF
3. Add negated conclusion S to the premises clauses
4. Repeat until Contradiction or no progress is made:
   - @ Select 2 clauses (call them parent clauses)
   - ⓑ Resolve them together, performing all required unification
   - © If resolvent is empty clause, a contradiction has been found (i.e., S follows from premises).
   - ⓓ If not, add ~~note~~ resolvent to premises

If we Succeed in Step 4, we have proved the conclusion

Give the KB of 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 everything that Anil eats peanuts & 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

Prove by resolution that:

Representation in FOL

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

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

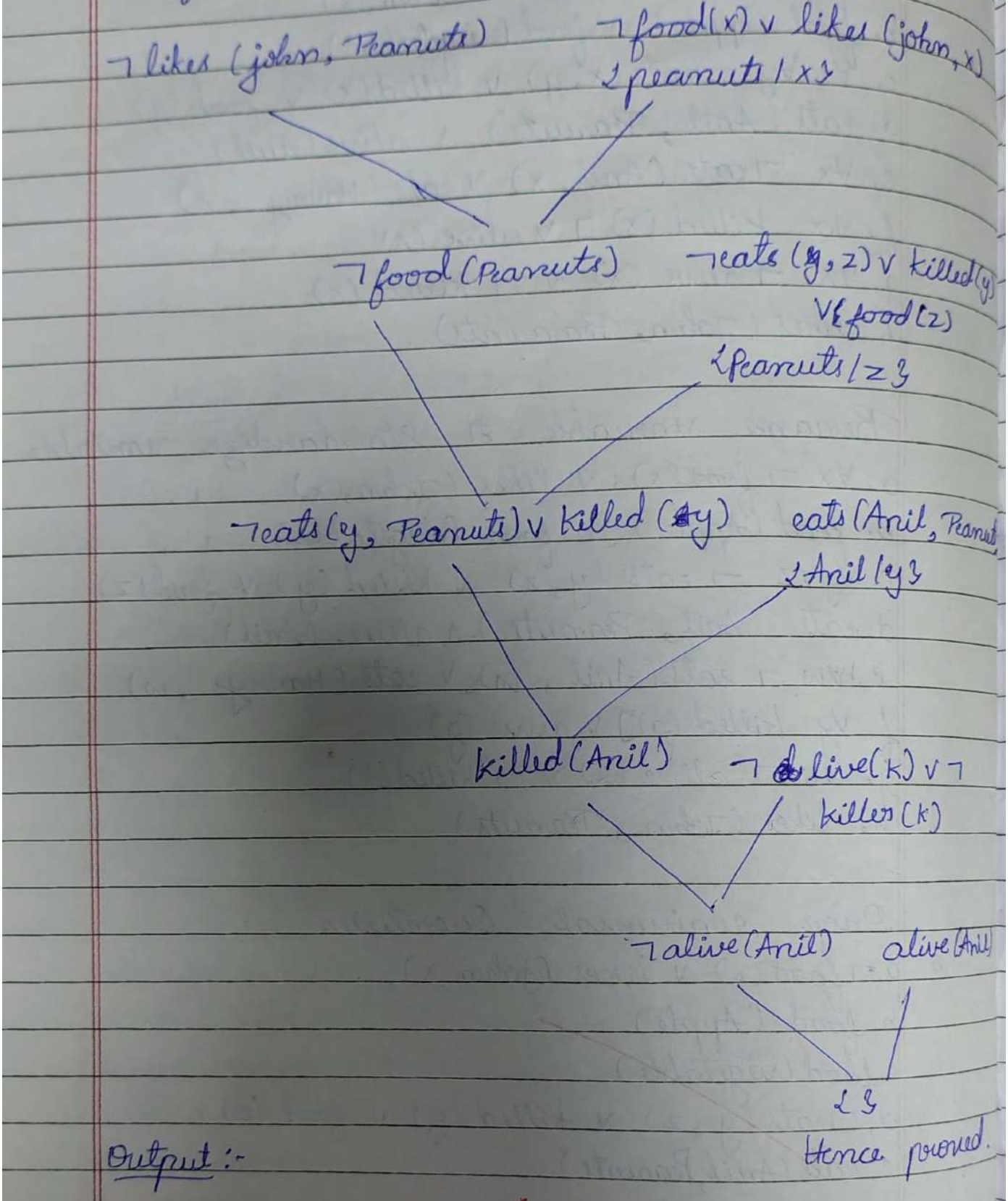
a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall x \forall y \neg [eats(x, y) \wedge \neg killed(x)] \vee food(y)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$
f. $\forall x : \neg killed(x) \rightarrow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

Move negation (¬) inwards & 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 variable or Standardize variable
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 Quantifier
6. a. $\neg food(x) \vee likes(john, x)$
b. $food(Apple)$
c. $food(vegetable)$
d. $\neg eats(y, z) \vee killed(y) \vee food(z)$
e. $eats(Anil, Peanuts)$
f. $alive(Anil)$
g. $\neg eats(Anil, w) \vee eats(Harry, w)$
h. $killed(g) \vee alive(g)$
i. $\neg alive(k) \vee \neg killed(k)$
j. $likes(john, Peanuts)$

Proof by Resolution



Hence proved.

Output :-

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

31/3/2022

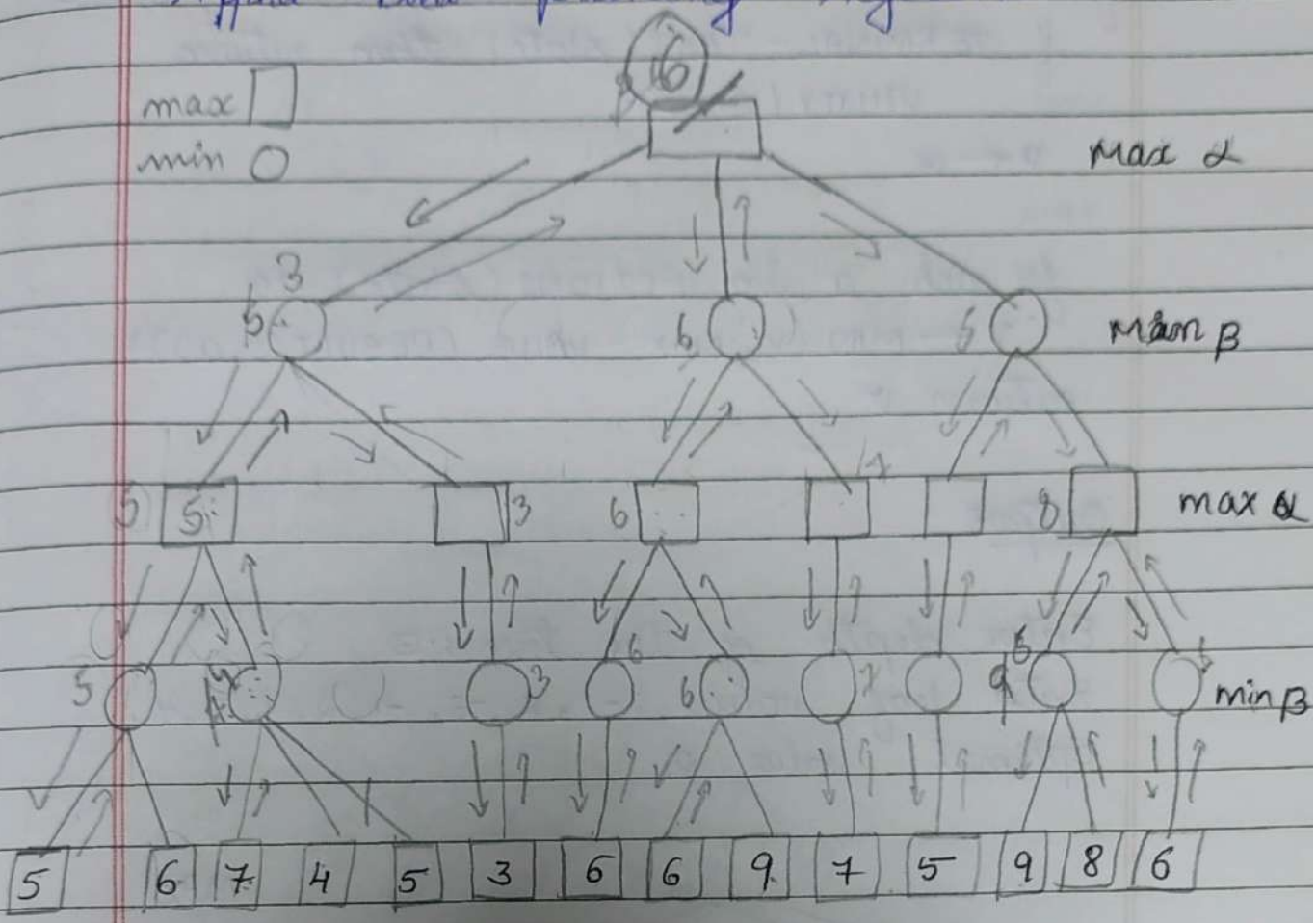
Lab-10

26/11/24

Bafna Gold

Date: Page: 25

Alpha Beta pruning Algorithm



$\alpha >= \beta$

Algorithm :-

function MINIMAX-DECISION (state) returns an action
return any max a ∈ ACTIONS (s)
MIX-VALUE (RESULT (state, a))

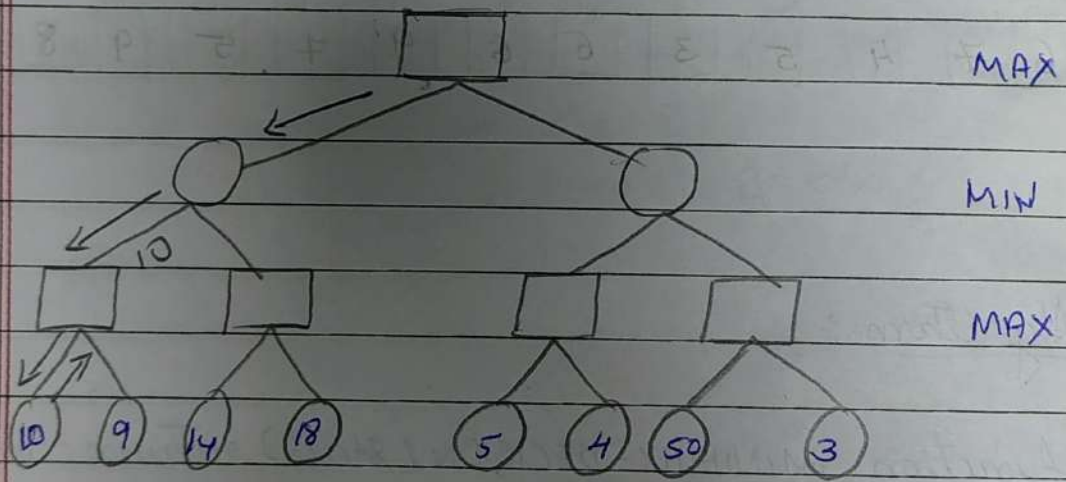
function MAX-VALUE (State) returns a utility value
if TERMINAL-TEST (State) then return
UTILITY (State)
$v \leftarrow -\infty$
for each a in ACTION (State) do
$v \leftarrow$ MAX (v, MIN-VALUE (RESULT (s, a)))
return v

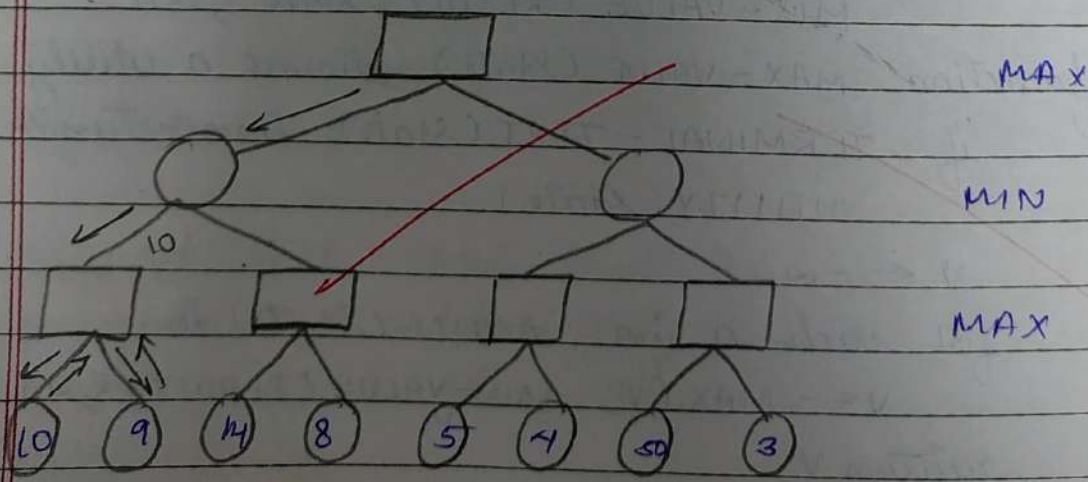
function MIN-VALUE (State) return a utility value
if TERMINAL-TEST (state) then return
UTILITY (State)
$v \leftarrow \infty$

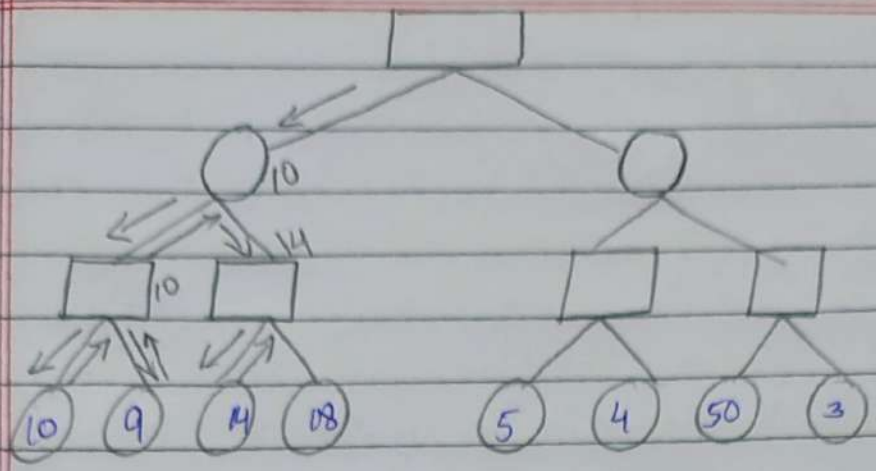
for each a in ACTIONS (state) do
$v \leftarrow$ MIN (v, MAX-VALUE (RESULT (s, a)))
return v

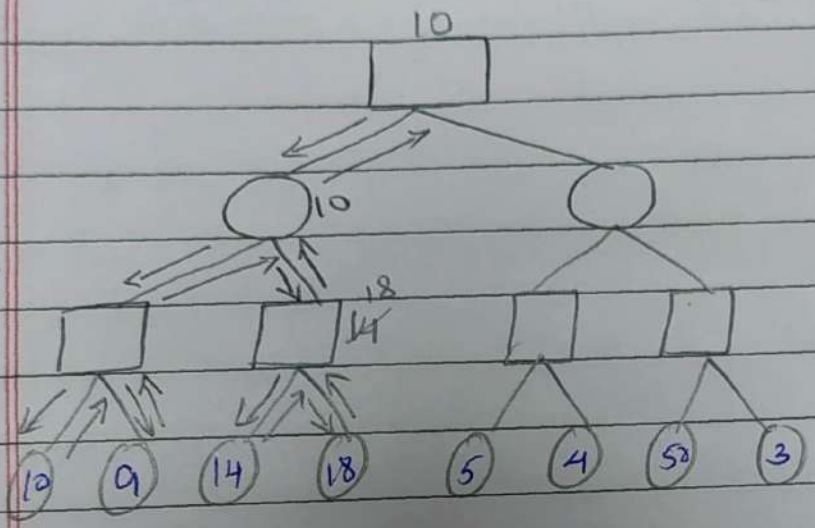
Output

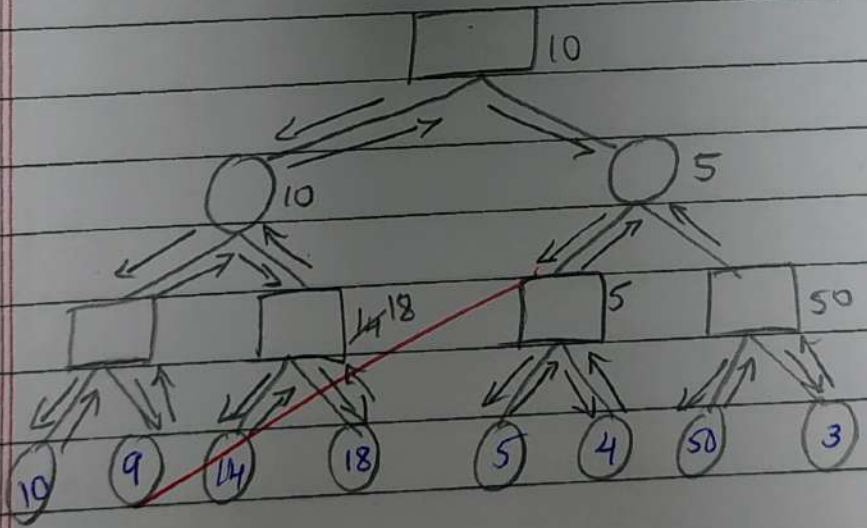
Enter depth of the tree : 3
Enter leaf values : -1, 8, -3, -1, 2, 1, 3, 4
Optimal value : 2






MAX
MIN
MAX
10
14
10
10
9
14
18
5
4
50
3


10
MAX
10
MIN
18
14
MAX
10
9
14
18
5
4
50
3


MAX
10
MIN
10
5
MAX
14
18
5
50
10
9
14
18
5
4
50
3

3/12/2024