INDEX

NAME: PRANAV SRINIVAS STD.: _____ SEC.: 1BM22CS203 SUB.: ARTIFICIAL INTELLIGENCE

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1)	24-09-2024	TIC-TAC-TOE AMPLEMENTATION		10
2)	01-10-2024	VACUUM CLEANER AGENT		10
3)	08-10-2024	8-PUZZLE PROBLEM		10
4)	15-10-2024	ITERATIVE DEEP SEARCH ALGORITHM A* ALGORITHM (Solution for 8-puzzle problem)		9
5)	22-10-2024	SIMULATED ANNEALING		10
6)	29-10-2024	N QUEENS PROBLEM → A* ALGORITH → HILL CLIMBING ALGORITHM		10
7)	12-11-2024	KNOWLEDGE BASE ENTAILMENT		10
8)	19-11-2024	FIRST ORDER LOGIC		10
9)	3-12-2024	FORWARD CHAINING BACKWARD CHAINING ALPHA BETA PRUNING (Similar questions as above)		9
10)	17-12-2024	ADDITIONAL QUESTIONS - FOL, ALPHABETA PRUNING Forward & Backward Chaining		

24th September, 2024 Tuesday — Laboratory 1

1) Implement the Tic-Tac-Toe Game, after having written an algorithm for the same.

Algorithm for the Tic-Tac-Toe Game

Step 1: Initialize the Game Board:
Create a 3×3 matrix (two-dimensional array) with empty values

Step 2: Display Board Function
Function prints the current state of the board.

Step 3: User Input Function
Function allows the user to input their move.

Step 4: Win Checker Function
Function to check if a player has won the game

Step 5: Draw Checker Function
Function to check if there is a condition for draw — the board if full and there can be no winner.

Step 6: Algorithm for the Computer's Move:

Conditions of draw.

X	O	O
X	O	X
O	X	O

Similarly.

Step 7: Main Game Loop:
While the game is going on.
{ Step (2); Step (3); Step (4), Step (5), Step (6), Step (5) }

Source code (in python):

```
import random
board = [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

def print_board():
    for row in board:
        print(" | ".join(cell if cell != ' ' else ' ' for cell in row))
        print("-" * 9)


def start():
    print("Welcome to tic-tac-toe!")
    random_num = random.randint(0, 1)
    if random_num == 0:
        print("player plays first.")
    else:
        print("Computer plays first.")
        computer_plays()

    while True:
        if random_num == 0:
            player_plays()
            if check_win('X'):
                print_board()
                print("Player won!")
                return
            random_num = 1
        else:
            computer_plays()
            if check_win('O'):
                print_board()
                print("Computer won!")
                return
            random_num = 0
```

```
if board_full():
    print_board()
    print("Draw!")
    return

def check_win(player):
    win = [
        [(0,0), (0,1), (0,2)],
        [(1,0), (1,1), (1,2)],
        [(2,0), (2,1), (2,2)],
        [(0,0), (1,0), (2,0)],
        [(0,1), (1,1), (2,1)],
        [(0,2), (1,2), (2,2)],
        [(0,0), (1,1), (2,2)],
        [(0,2), (1,1), (2,0)],
    ]

    for win in wins:
        if all(board[x][y] == player for x,y in win):
            return True
    return False

def player_plays():
    print_board()
    while True:
        try:
            a,b = map(int, input("Enter the co-ordinates (row and
                column 0-2, eg. '0 1'): ").split())
            if board[a][b] == ' ':
                board[a][b] = 'X'
                break
            else:
                print("Cell already taken. Try again.")
        except (ValueError, IndexError):
```

print("Invalid Input. Please enter co-ordinates in the format row-column!")

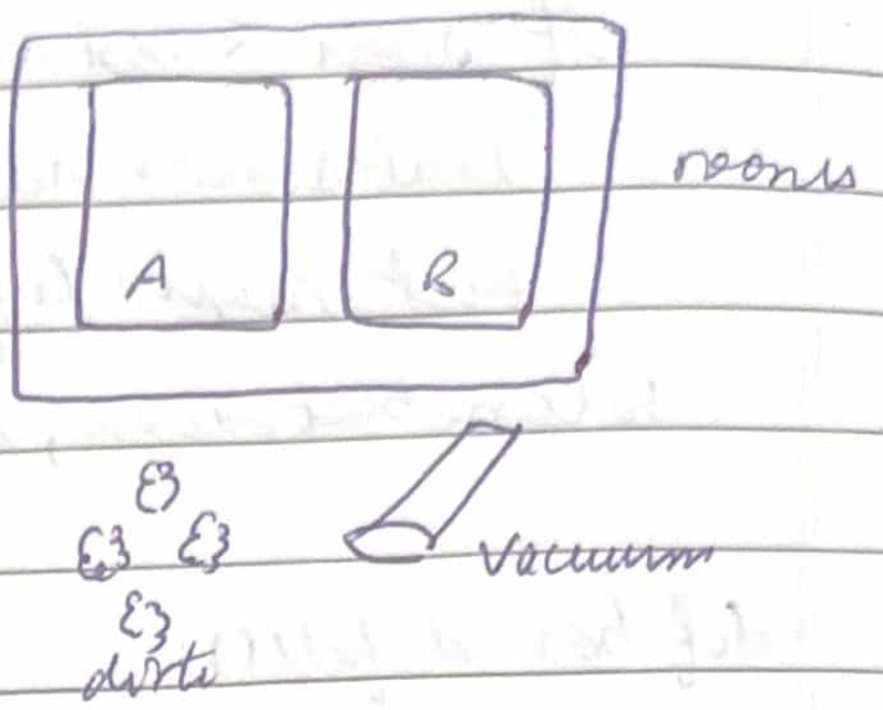
```
def computer_plays():
    _, move = algo(board, '0')
    board[move[0]][move[1]] = '0'


def algo(board, current_player):
    if check_win('X'):
        return -1, None
    elif check_win('0'):
        return 1, None
    elif board_full():
        return 0, None
    if current_player == '0':
        best_score = -float('inf')
        best_move = None
        for i in range(3):
            for j in raye(3):
                if board[i][j] == '':
                    board[i][j] = '0'
                    score, _ = algo(board, 'X')
                    board[i][j] = ''
                    if score > best_score:
                        best_score = score
                        best_move = (i, j)
        return best_score, best_move
    else:
        best_score = float('inf')
        best_move = None
        for i in range(3):
            for j in range(3):
                if board[i][j] == '':
                    board[i][j] = 'X'
```

```
score, _ = algo(board, 'O')
board[i][j] = ''
if score < best_score:
    best_score = score
    best_move = (i, j)
return best_score, best_move

def board_full():
    return all(all(i != '' for row in board for cell in row)

start()
```

Date
Page

1st October, 2024 Tuesday Laboratory - 2

2) Implement the working of vaccuum cleaner agent for two rooms.



Algorithm for implementing the vacuum cleaning for two rooms A and B

Step 1: Input the data from sensors whether the room is dirty or clean

Step 2:
```
while ( room != clean)
  {
    if (current room = dirty)
      clean();
    else if ( current room = clean)
      {
        if (agentpresent in = room A)
          move right;
        else if ( agentpresent = roomB)
          move left;
      }
  }
```

Step 3: Output the status of room cleanliness

Percept Sequence	
[A, clean]	right
[A, dirty]	suck
[B, clean]	left
[B, dirty]	suck

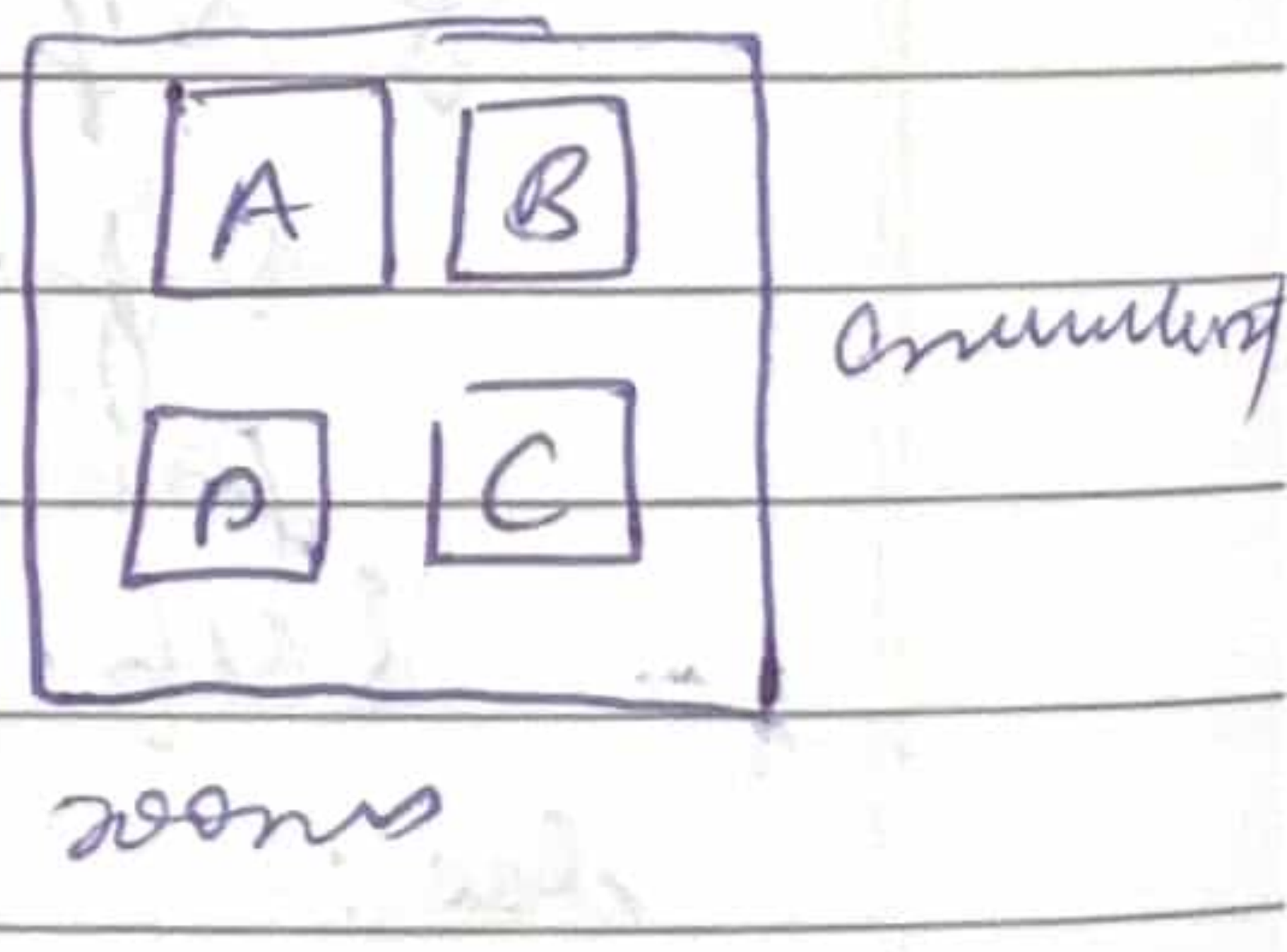
[A, clean] [A, clean]	left
[A, clean] [A, dirty]	suck
[B, clean] [B, clean]	right
[B, clean] [B, dirty]	suck
[A, clean] [B, clean]	exit

17/10/24

Source code (in python)

```
class VacuumCleaner:
  def __init__(self):
    self.rooms = {'A': 'dirty', 'B': dirty, 'C': 'dirty', 'D': dirty'}
        # All rooms start dirty
    self.position = 'A' # start from A
    self.room_order = ["A', 'B', 'C', 'D'] # order of the rooms
    self.current_index = 0  # start with first room in list order

  def status(self):
    print(f"In Current Position : {self.position}")
    for room, state in self.rooms.items():
      print(f"Room {room}: {state}")

  def suck(self):
    if self.rooms[self.position] == 'dirty':
      self.rooms[self.position] = 'clean'
      print(f"Sucking in room {self.position}. Room
      {self.position} is now clean.")
    else:
      print(f"Room {self.position} is already clean.")
```

```
def.move_clockwise(self) :
    self.current_index = (self.current index + 1) % len(self roomorder
    self.position = self.room_order [self.current_index]
    print(f"moved clockwise to room {self.position}.")

def move_conticlockwise (self)
    self.current_index = (self.current index - 1) % len (self.room_or
    self.position = self.room_order[self.curnt-index]
    print (f"Moved conticlockwise to room {self.positio}.")

def run(self)
    while 'dirty' in self.rooms_values():
        self status()
        if self.rooms. [self.position]== 'dirty'
        self suck()
        else
            if self.position == 'D'
                self.move_conticlockwise()       #
            else
                self.move_clockwise()          #
    otherwise
        print ("All rooms are clean!")

# You can move the vacuum cleaner based on your preference
  of choice.

if _name_ == '_main_'
    vacuum = VacuumCleaner()
    vacuum.run()
```



S B
1/10/24

8th October, 2024 Tuesday Laboratory - 3

3) Solve the 8 puzzle problem using
(i) Depth First Search
(ii) Manhattan Distance

Algorithm for the 8 puzzle problem using Depth First Search & Manhattan Distance

Step 1: Initialize the start and goal states:
Define the initial puzzle configuration and the goal configuration.

Step 2: Find the blank / empty tile:
Locate the position of the blank / empty tile.

Rough work:

8	2	6
3		7
1	4	5

Blank tile is in — Number of moves port
middle :- 4
edge : 3
corner : 2

Step 3: Check if the current state is the goal state:
Comparision of the current puzzle state with the goal configuration. If they match, the puzzle is solved.

Step 4: Explore possible moves as shown in the rough work
(Right, down, left, up)

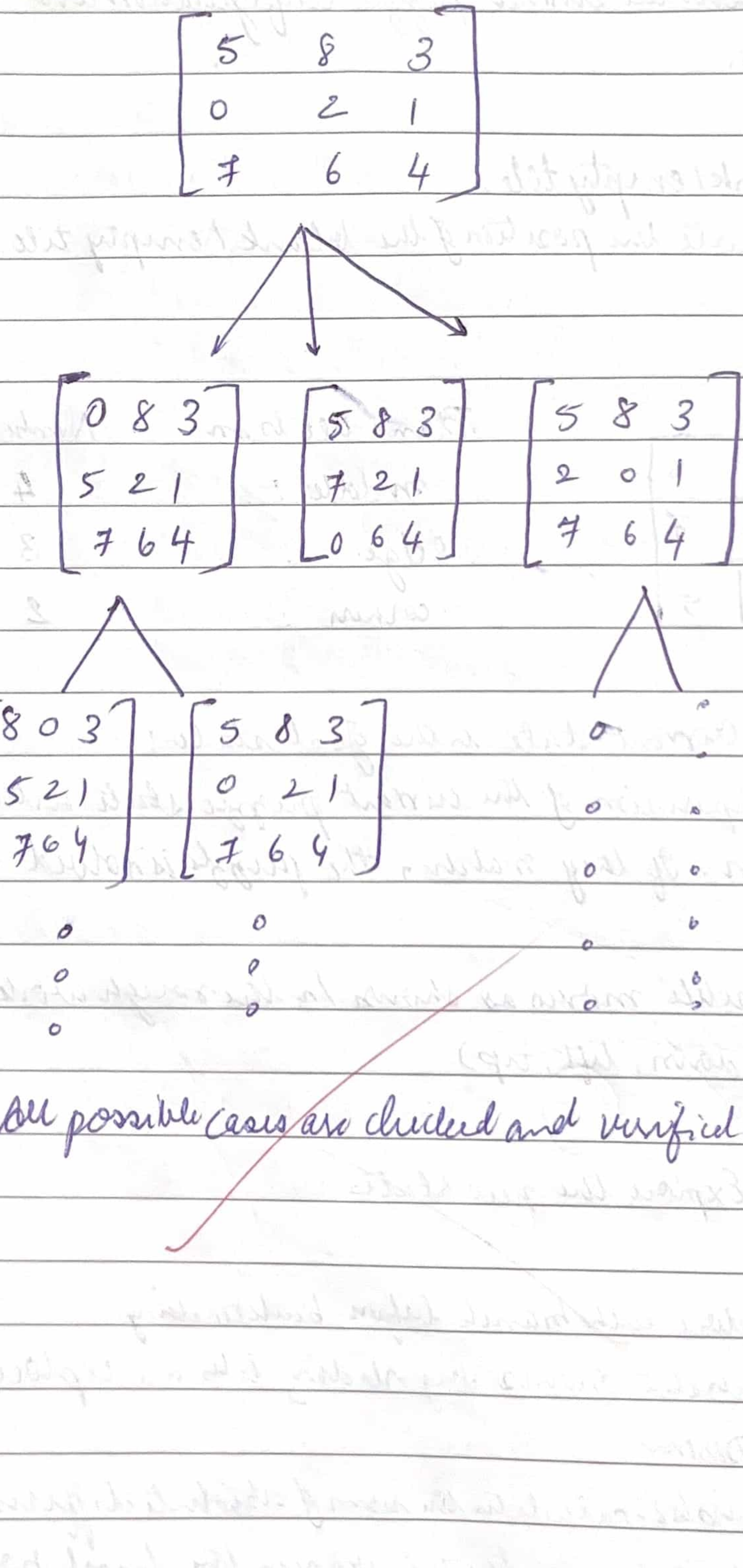
Step 5: Recursively Explore the new states:
DFS:
- Explore each branch before backtracking
- Generate moves by sliding tiles and explores possible mov
- Manhattan Distance
- Heuristic: calculate the sum of absolute differences between the current positions of the tile verses the final positions tile

Step 6: Backtracking if and only if no solution is found

Step 7: End when the goal state is reached or all possibilities are exhausted.

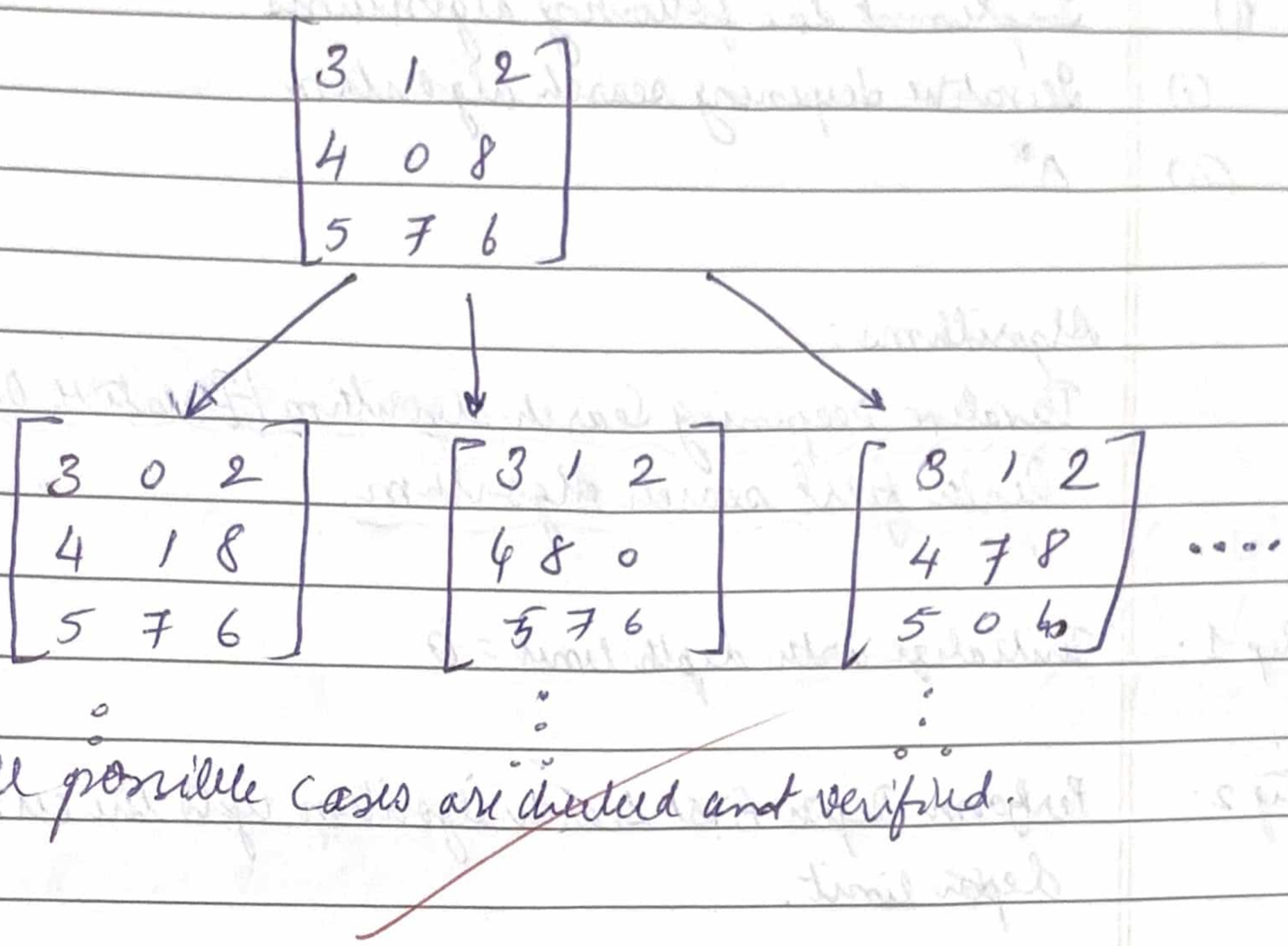
Example :

Depth First Search



All possible cases are checked and verified.

Manhattan Distance



All possible cases are checked and verified.

Date
Page

15th October, 2024. Tuesday Laboratory - 4

4) Implement the following algorithms:

(i) Iterative deepening search algorithm

(ii) A*

Algorithms:

Iterative Deepening Search algorithm / Iterative Deepening Depth first search Algorithm:

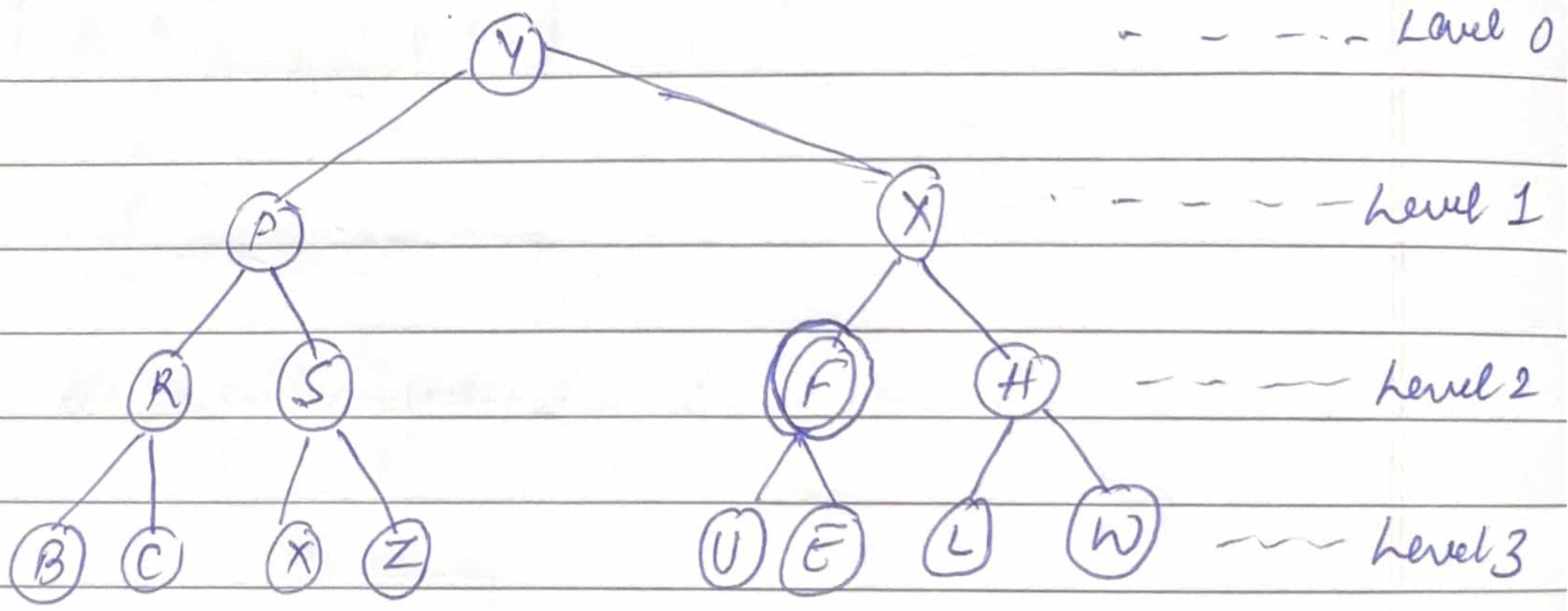
Step 1: Initialize with depth limit = 0

Step 2: Perform Depth First Search algorithm upto the current level / depth limit.

Step 3: If a solution is found, return it.

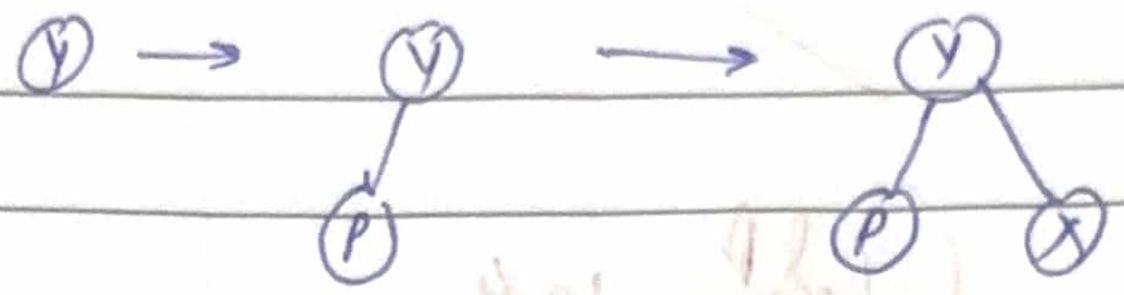
Step 4: If no solution is found, increment the depth limit and repeat the depth first search algorithm from the start

Step 5: Continue until a solution is formed or all the levels are explored.

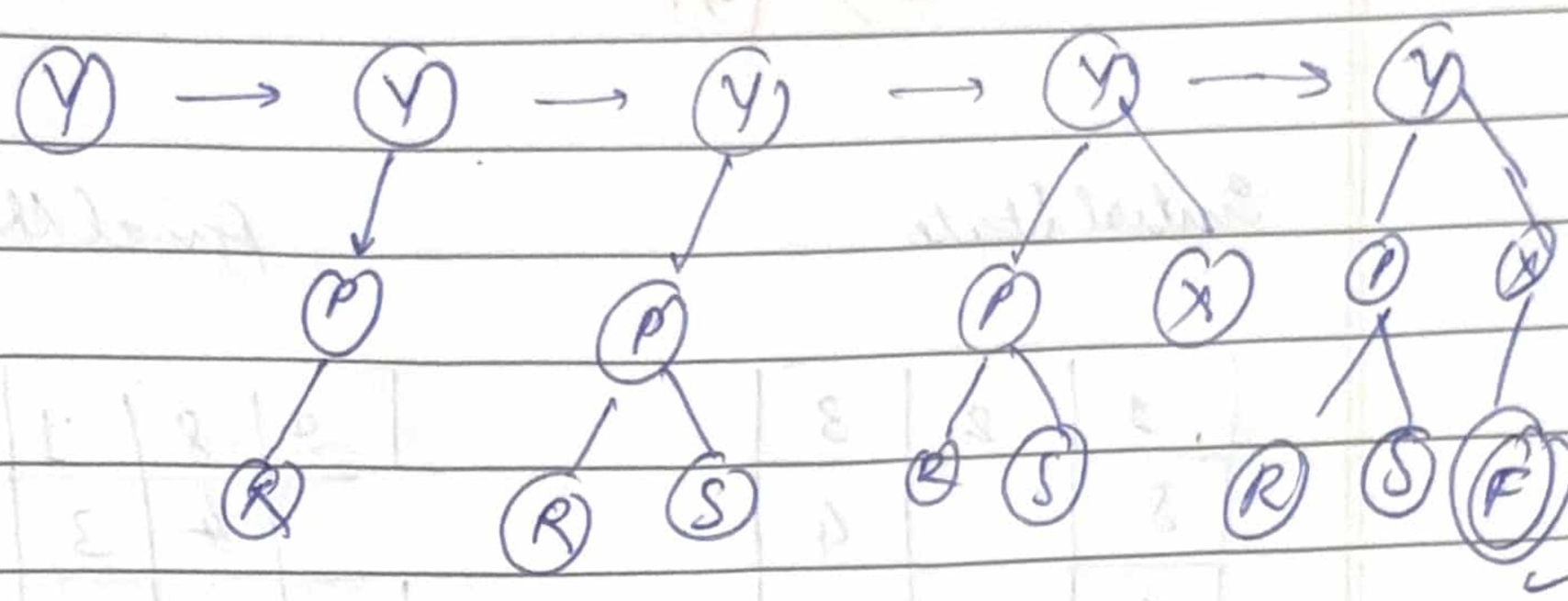


Level 0:

Level 1:



Level 2:



Found final state (F).

A* Algorithm

Step 1: Initialize the open list: the set of all nodes to be evaluated with the start node and closed list: set of already evaluated nodes

Step 2 while (open list != empty)

{

- select the node with the lowest f(n) value from the open list
- If the selected node is goal, reconstruct and the path is returned
- else, move it to the closed list
- for every neighbour of the current node:
  - → If the neighbour is in the closed list, ignore it.
  - → If the neighbour is not in the open list, add it and compute its f(n) score
  - → If the neighbour is in the open list but a better g(n) value is found, update its score as part

22nd October, 2024 Tuesday Laboratory - 5

Write a program to implement Simulated Annealing Algorithm. Write the complete program and define the objective function. Also, execute a program considering the temperature and include valid comment lines to indicate the flow of the program.

Simulated Annealing Algorithm :- Optimization Test

Step 1: Initialize Parameters:
- Set an initial solution S
- Define an initial temperature T
- Set Cooling rate α $(0 < \alpha < 1)$ and minimum temperature T_{min}
- Define a maximum number of iterations per temperature

Step 2: Evaluate Initial Solution:
Calculate the cost / energy $E_1(S)$ of the initial solution

Step 3: while $(T > T_{min})$
For each iteration i in the maximum iterations:
- Generate a new solution S' by making a small random change to S.
- Calculate the E(S') of the new solution.
- If $E(S') < E(S)$:
Accept the new solution S' as the current solution
(set $S = S'$)
- Else, calculate the probability of accepting S':
$P = e^{\left(\frac{E(S) - E(S')}{T}\right)}$
If a random number 'r' (from uniform distribution [0, 1]) is less than P.
- Cool the temperature:
$T = \alpha T$

Step 4: Return the Best Solution found.

22/10/24

Output :

Iteration 0, Temperature 10.000, Best Evaluation 23.63658
Iteration 100, Temperature 0.099, Best Evaluation 9.01327
Iteration 200, Temperature 0.050, Best Evaluation 8.95493
Iteration 300, Temperature 0.033, Best Evaluation 8.95493
Iteration 400, Temperature 0.025, Best Evaluation 8.95493
Iteration 500, Temperature 0.020, Best Evaluation 8.95493
Iteration 600, Temperature 0.017, Best Evaluation 8.95493
Iteration 700, Temperature 0.014, Best Evaluation 8.95493
Iteration 800, Temperature 0.012, Best Evaluation 8.95493
Iteration 900, Temperature 0.011, Best Evaluation 8.95493
Best Solution: [-0.0011933906599737107 7, 2.9843731809163994]
Best Score: 8.954922976488485 4.

Sneha S B
22/10/24

29th October, 2024 Tuesday Laboratory-6

For the (N=8) Queens Puzzle, implement using the following algorithms:

(i) A*

(ii) Hill Climbing Algorithm

A* Algorithm to solve the (N=8) queens Problem.

Step 1: Create a 8×8 chessboard in the initial state
Setup an open set to explore configurations
Setup a visited state to display different sets visited

Step 2: Calculate the number of attacking pairs that Queens should not be in the same row or same column or same diagonal

```
if state[j] == state[j] or abs(state[i])
    state[j] == j-1  // diagonal
    attacks += 1
```

Then we increment the variables attacks to determine attacking pairs.

```
open_set : []
```

Step 3: Assign initial state to open state for the first iteration. If the node is not visited, then first put it iteration. If open set and will push the node to heap q' (priority q)

```
heap q = heap_push(open set, Node(new state, g, h))
```

Step 4: Total estimated cost f = g + h,
g → cost to reach the current state
h → number of attacks to reach goal state

Step 5: Main loop { remove the node with lower cost }

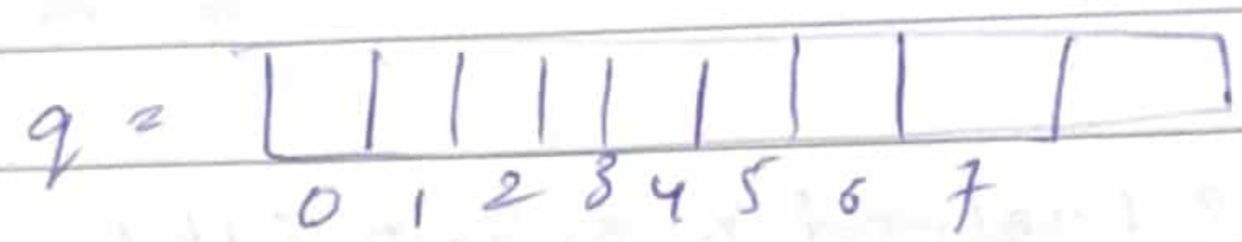
Step 6: Determination of the next row to place the queen

Step 7: Update g and calculate h

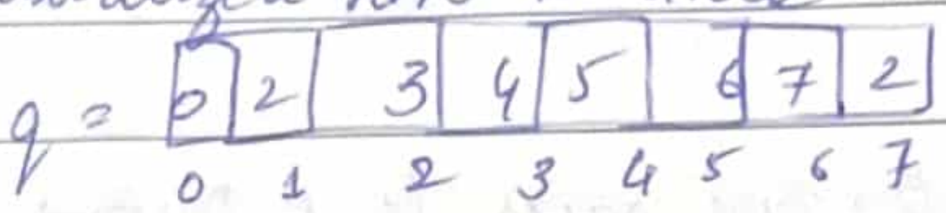
Hill climbing algorithm

Step 1: Create an array where each index represents a column and the value represents the row position of the queen in that row

```
int[] q = new int[8]
```



Step 2: Initialize a random state



Step 3: Store a heuristic value h(n) where 'h' represents the number of conflicting pairs in each state.

Step 4: Check for conflicting pairs:

```
conflicts()
{
conflicts = 0
for i ← 0 to 7 do
  for j ← i+1 to 7 do
    if (board[i] = board[j])
      conflicts += 1
return conflicts
```

Step 5: current = conflicts(board)

Step 6:

```
for i ← 0 to 7 do
  for j ← 0 to 7 do
    board[i] = j
    C = conflicts(board)
```

```
if (c < current)
  { temp = current
    current = c,
  }

if (c == temp)
      return No Solution

if (c == 0)
      return current state
```

Pseudocode

Output:

Best Solution found (with 0 attacking pairs)

```
. . . . Q . . .
. . . . . . . Q .
. Q . . . . . . .
. . . . . Q . . .
. . Q . . . . . .
Q . . . . . . . .
. . . Q . . . . .
. . . . . . . Q .
```

Screen

24/10/24

12th November, 2024. Tuesday Laboratory - 7

Entailment is a deduction or implication which follows or is in accordance and accuracy with logic

To derive these implications, rules of inference and logical equivalences are used (to derive relationships) together with rules of truth functional values.

① Modus Ponens (MP)

$P \rightarrow Q$ (If P then Q)

P is true, then Q is true

Example: P ⇒ It is raining

Q = Playground wet

If it is raining then playground will be wet.

② Modus Tollens (MT)

$P \rightarrow Q$, $\neg Q$ Conclusion $\neg P$

(If P then Q), Q is false)

If it is raining, the ground will be wet, ground is not wet

It is not raining.

③ Hypothetical Syllogism (HS)

$P \rightarrow Q$, $Q \rightarrow R \therefore P \rightarrow R$

R = Grass will grow

If it is raining, the ground will be wet, the grass will grow

④ Disjunctive Syllogism (DS)

$P \vee Q_1$, $\neg P$ Conclusion: Q_1 is true

Q_1 = It is raining

Either it is raining or snowing
It is raining ✗
Then, it is snowing

⑤ Bi-conditional:

If $P \rightarrow q$ is true, $P \rightarrow q$ and $q \rightarrow P$
are true

Conclusion:

$P \rightarrow q$

$q \rightarrow P$

⑥ Contradiction

P leads to a contradiction, $\neg P$ must be true

Raining & ground not wet.

conclusion: $\leftrightarrow (P \wedge \neg q))$

Solution to the question given:

① Premises from the knowledge base:

P1: M(A, B) Alice is the mommy of Bob

P2: F(B, C) Bob is the daddy of Charlie

P3: $\forall x (F(x,y) \rightarrow P(x))$ if x is dad of y, x is parent of y

P4: $\forall x (M(x,y) \rightarrow P(x))$ " → "mom-", → " → ",

P5: All parents have children

$\forall x (P(x) \rightarrow \exists y (C(x,y)))$

$\forall x \forall y (P(x) \wedge C(x,y) \rightarrow S(y,z))$ (If x is a parent
P6: x has children y & z, then y & z are siblings

P7: Alice is named to dad

Mark(A, D)

Date
Page

(2) Hypothesis:
$S(B, C)$

(3) Entailment Proves
↳ $M(A, B)$ P1
↳ $F(B, C)$ P2
$F(B, C) \rightarrow P(B)$ deduction P3
$M(A, B) \rightarrow P(A)$ P4

↳ $P(A) \rightarrow \exists y\ (C(A, y))$
Alice, a parent has children.

↳ If someone is a parent, their children are siblings.
A and B are parents. Bob and Charlie are children of Bob and Alice.
Hence, Bob and Charlie are siblings

(4) Conclusion
$S(B, C)$ is entailed by knowledge base

Sahasra
1/12/24

19th November, 2024 Tuesday Laboratory - 8

Explanation of the Algorithm -

1) John is a human
   Human (John)

2) Every Human is mortal.
   $\forall x$ (Human (x) $\rightarrow$ Mortal (x))

3) Joseph loves Mary
   Loves (Joseph, Mary)

4) There is someone who loves Mary
   $\exists x$ (loves (x, Mary))

5) All dogs are animals
   $\forall x$ (dogs (x) $\rightarrow$ animal (x))

6) Some dogs are black
   $\exists x$ (Dog (x) $\wedge$ Black (x))

Unification: Eats (x, Apple)
Eats (Riya, y)

= Eats (Riya, Apple)

Proceed

Sample:

Enter a sentence like

1) John is a human
2) Every Human is mortal
3) John loves Mary
4) There exists someone who loves Mary

Enter a sentence: Mary is the mother of Jesus
False

3rd December, 2024. Tuesday. Laboratory - 9

Q: Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

Forward chaining is one of the two methodologies using an inference engine which starts with a base state and uses the inference rules and available knowledge in the forward direction till it reaches the goal state.

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

Prove that "Robert is criminal"

Solution: / Proof /

→ It is a crime for an American to sell weapons to hostile nation
Lets say p, q, and r are variables.

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

→ Country A has some missiles

∃ x owns (A, x) ∧ missile(x)
Existential Instantiation thus introducing a new constant T
Owns (T, P)
missile (T)

→ All of the missiles were sold to country A by Robert
∀x Missile(x) ∧ Owns(A, x) ⇒ sells (Robert, x, A)

→ Missiles are weapons

Missile(x) ⇒ weapon(x)

→ Enemy of America is known as hostile

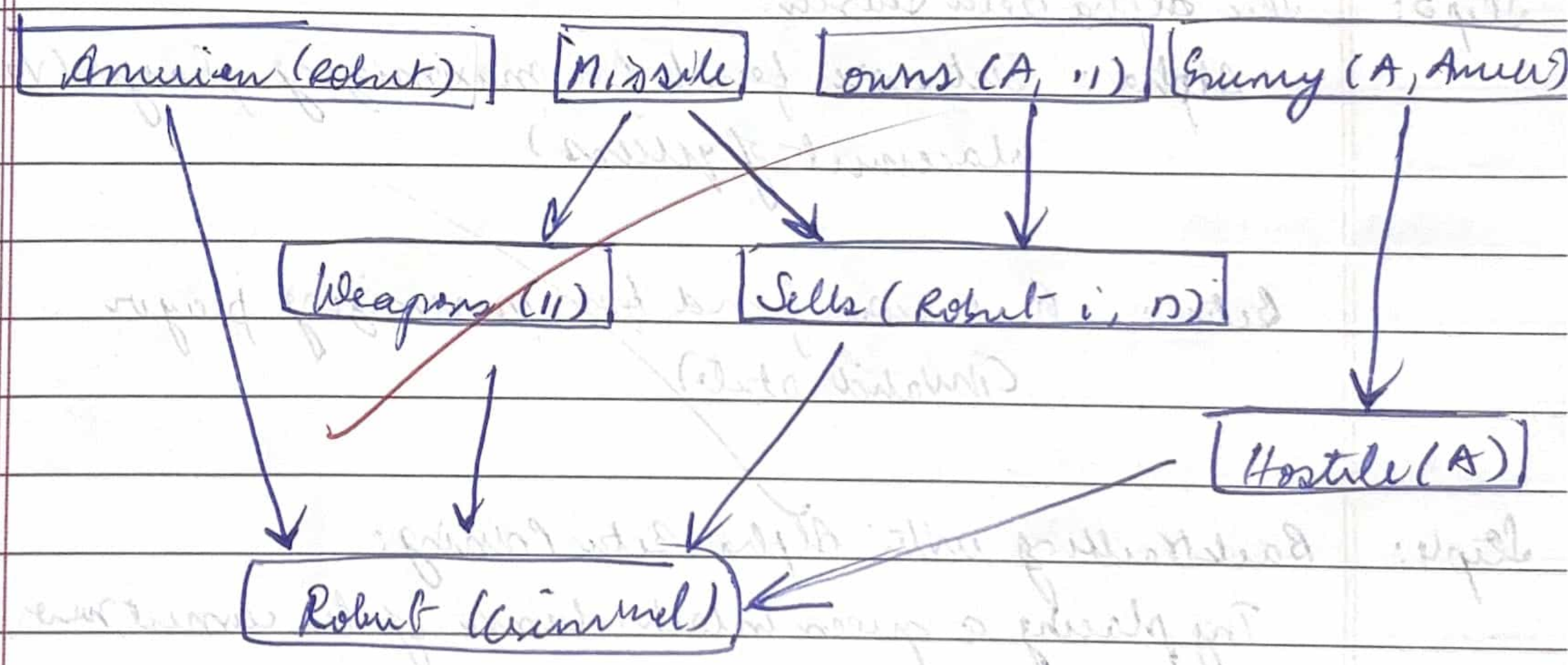
∀x Enemy(x, America) ⇒ Hostile(x)

→ Robert is an American

American(Robert)

→ The country A, an enemy of America

Enemy(A, America)



Date
Page

8: 8 Queens Problem: Alpha Beta Pruning.

Algorithm:

Step 1: Initialize the board: Create an 8x8 board where each row will hold the column number of the queens placed in that row. Start with no queens placed: (board = [-1] * 8)

Step 2: Define 'is_safe' function: check if the placing of a queen at (row, column) is safe; i.e the two queens threaten each other.

Step 3: The Alpha Beta Search:
- Alpha: Best score found for maximizing player (valid placement of queens)

Beta: Best score found for minimizing player (invalid state)

Step 4: Backtracking with Alpha Beta Pruning:
Try placing a queen in each column of the current row.
Recursively place queens in subsequent rows.
Prune branches where further exploration is unnecessary (based on alpha and beta values)

Step 5: Return Solution: If solution is found, return the solution

Date
Page

Q: Minmax Algorithm for Tic Tac Toe

Step 1: Check if the game is over:
- If a player wins, return the score
  * +1 for 'x' win
  * -1 for '0' win
  * 0 for a draw

If the board is full and no one has won, return (0' draw)

Step 2: Maximizing player (x)
- Initialize the best score to -∞
- For each available move on the board
  → Make the move (place 'x')
  → Call the minmax function recursively, switching to the minimizing player (0)
- Undo the move
- Update the best score with the maximum value from the recursive call.
- Return the best score

Step 3: Minimizing Player (0)
- Initialize the best score to ∞
- For each available move on the board.
  - Make the move (place '0')
  - Call the minmax function recursively.
- Similar to step 2

Sneha SB
3/12/24

Step 4: Find the best move for the maximizing player (x).
- For each available move, evaluate move using minmax
- Return the move with highest score

17th December, 2024. Tuesday Lab no 10 - 80

1) Consider the following English statements

1. If someone suffers from allergies, then he/she sneezes
2. If someone lives with a cat and is allergic to cat, then he/she will suffer from allergies.
3. Tom is a cat
4. Mary is allergic to cats

Represent the above sentences in FOL and prove by FOL resolution "Mary sneezes"

On representation of the above premises in First order logic

1. allergic(x) → sneeze(x)
2. cat(y) ∧ allergic to cats(x) → allergic(x)
3. cat(Tom)
4. allergic to cats(Mary)

The goal state is sneeze(Mary)

allergic(x) → sneeze(x)

¬ allergic(x) ∨ sneeze(x)

cat(Y) & allergic to cats(X) → allergic(X)

¬(cat(Y) ∧ allergic to cats(X)) → ∨ allergic(X)

¬ cat(Y) ∨ allergic to cats(b) → ∨ allergic(X)

State Diagram to reach goals:

¬allergies(w) ∨ sneeze(w)

¬cat(y) ∨ ¬allergic to cat(z) ∨ allergic(z)

w/z

¬cat(y) ∨ sneeze(z) ∨ allergic to cats(z) (1) cat(Tom)

y/Tom

allergic to cats(May) sneeze(z) ∨ ¬allergic to cats(z)

z/May

Sneeze(May) ¬sneeze(May)

{}

∵ $x + \bar{x} = 1$

$x \cdot \bar{x} = 0$

2) Consider the first order Logic (FOL) representations for the following sentences

(i) Every real number has its corresponding negative

(ii) Everybody loves somebody

(iii) There is somebody whom no one loves

(iv) Sue ate everything that Ronaldo brought

(v) Parot is green while rabbit is not.

3) Find a most general unifier for the set

(a) $W = \{ P(a, x, f, (g(y))), P(z, f(z)) f(w)) \}$

(b) $W = \{ g(f(a), g(x)), g(y, y) \}$

8

a) $\forall x \; real(x) \rightarrow negative(x)$

(or)

$\forall x \; real(x) \; \exists (negative(x))$

(i) $\forall x \exists y \; loves(x, y)$

(ii) $\forall x \; \exists y \; r \; loves(x, y)$

(iii) $\forall x \; brayn(Rusion, x) \rightarrow bright(Ronald, x)$

(iv) $green(parrot) \wedge \neg green(rabbit)$

(or)

$\forall x \forall y ((Parrot(x) \rightarrow Green(x) \wedge Rabbit(y) \rightarrow Green(y))$

9

(i) $P(a, x, f(g(y)) \quad P(z, f(z), f(w))$

Substitute z for a

$P(a, x, f(g(y)) \quad P(a, f(a), f(a))$

On substituting a by $g(y)$ (1)

$P(a, x, f(g(y)) \quad P(a, f(a), f(g(y))$

On substituting x by $f(a)$

$P(a, f(a), f(g(y)) \quad P(a, f(a), f(g(y))$

$x / f(a)$

The general unifier $g/a, w/g(y)$

$x / f(a)$

(ii) $g(f(a), g(x)) \; g(y, y)$ (UX)

Term is not unifiable as y cannot accept two values

3) Convert the following sentences into FOL statements and prove the following statement using both forward & backward chaining.

John likes all kinds of foods
$\forall x \; food(x) \rightarrow likes(John, x)$

Apples are food
$food(Apples)$

Chicken is food
$food(Chicken)$

Anything anyone eats and isn't killed by is food
$\forall x \forall y \; eats(x, y) \wedge \neg killedby(x) \Rightarrow food(y)$

Bill eats peanuts and is still alive
$eats(Bill, peanuts) \rightarrow alive(Bill)$

Sue eats everything Bill eats
$\forall x \; (eats(Bill, x) \rightarrow eats(Sue, x))$

* An additional required fact: $\neg killed(x) \rightarrow alive(x)$

The forward chaining

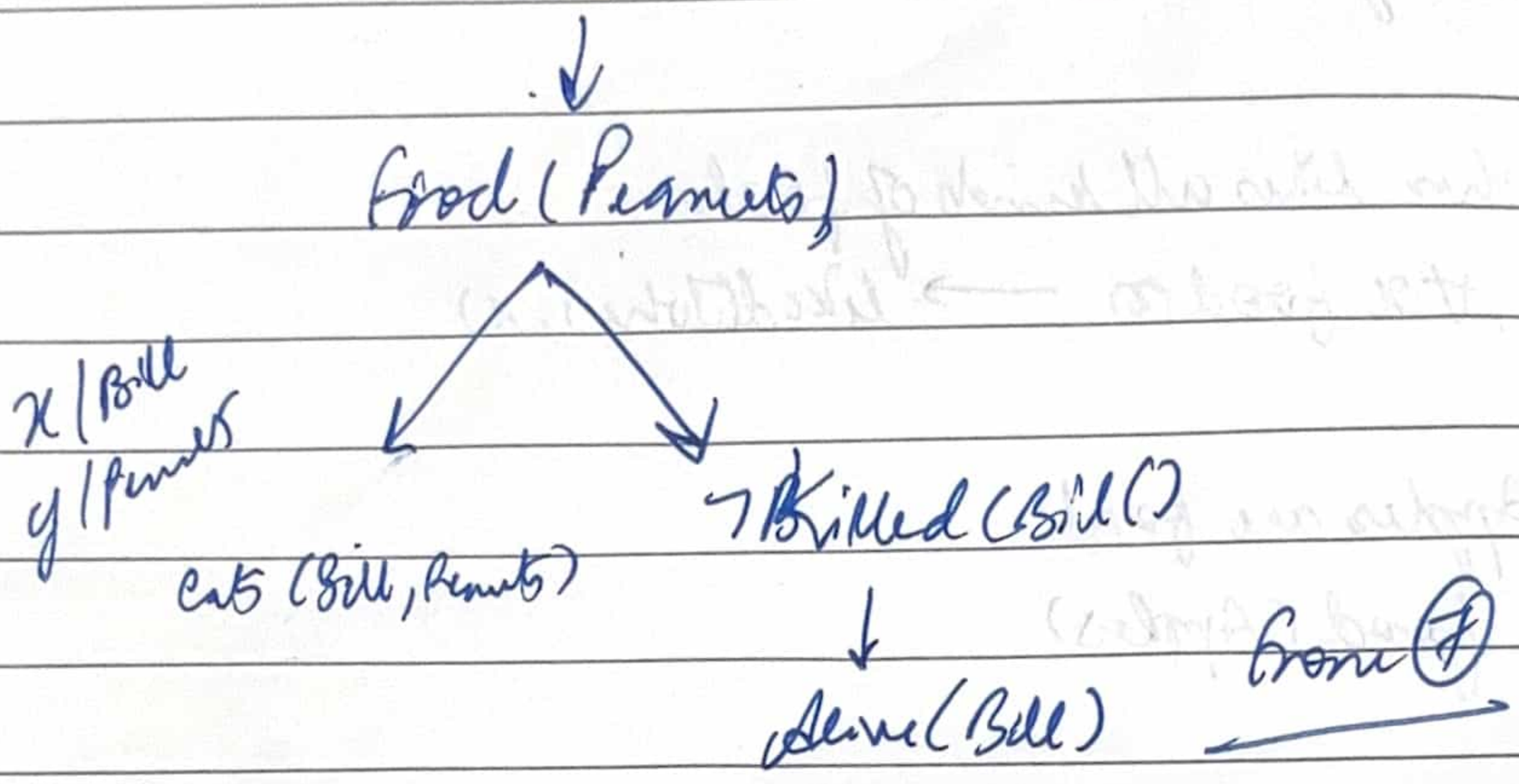
$x/Bill, y/Peanuts$ — $alive(Bill)$
$eats(Bill, Peanuts)$ — $\neg killed(Bill)$

$food(Peanuts)$

$x/Peanuts$ — from (1)

$likes(John, Peanuts)$

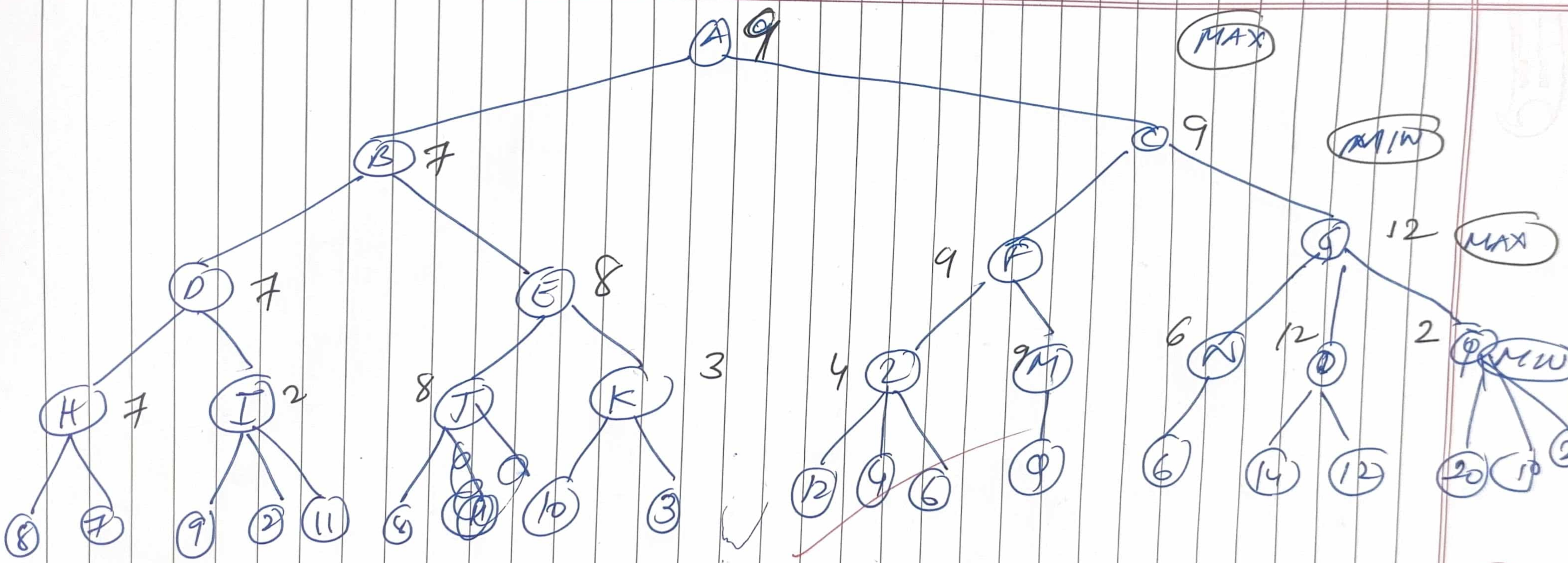
Backward chaining:



4) 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)
$v \leftarrow \infty$
for each a in ACTIONS (state) do
$v \leftarrow$ MIN (v, MAX-VALUE (RESULT (s, a))
return v



17/2/24

classmate
Date
Page