LAB 1

TIC TAC TOE (user v/s computer)

Algorithm.

Step 1 → create a 3×3 matrix which will give 9 empty boxes.

Step 2 → Assign X to or O to the user depending on the symbol he chooses And then assign the other to the computer. Assign – to represent empty spaces.

Step 3 → If the user starts the next turn will be given to the computer. this will happen till one of them win or till all the null spaces are filled which results in a tie.

Step 4 → If Now if the user starts, the computer should place O in such a way that it should not let the user make a row, a column or a diagonal with X's. (basically should not let the user win).

Step 5 → If the user or the computer creats a row or a column or a

Date / /
Page 2

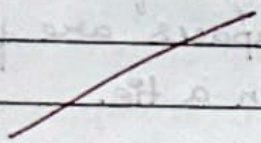
diagonal with three respective symbols, will be the winner of that particular round in the tic-tak-toe game.

OR

If all the moves are done and there are no empty space left and non of them have made a row or a column or a diagonal then it will be considered as a tie.


```
{ if a[i][0] == a[i][1] == a[i][2] == player
  if [0][i] == a[1][i] == a[2][i] == player
  if [2][0] == a[1][1] == a[2][2] == player
  if [0][2] == a[1][1] == a[2][0] == player
```

{ if a[i][i] == 'x' if a[i
}



code :-

```
import random
def print_board(board):
    for row in board:
        print("|".join(row))
        print("-" * 9)

def check_winner(board):
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] != " ":
            return board[i][0]
        if board[0][i] == board[1][i] == board[2][i] != " ":
            return board[0][i]
    if board[0][0] == board[1][1] == board[2][2] != " ":
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != " ":
        return board[0][2]
    return None

def is_board_full(board):
    return all(cell != " " for row in board for cell in row)

def ai_move(board):
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "O"
                if check_winner(board) == "O":
                    return
                board[i][j] = " "
```

```
for i in range(3):
    for j in range(3):
        if board[i][j] == " ":
            board[i][j] = "x"
            if check_winner(board) == "x":
                board[i][j] = "0"
                return
            board[i][j] = " "
if board[1][1] == " ":
    board[1][1] = "0"
    return

corners = [(0,0), (0,2), (2,0), (2,2)]
random.shuffle(corners)
for corner in corners:
    if board[corner[0]][corner[0]][corner[1]]
        == " ":
        board[corner[0]][corner[1]] = "0"
    return

side = [(0,1), (1,0), (1,2), (2,1)]
random.shuffle(sides)
for side in sides:
    if board[side[0]][side[1]] == " ":
        board[side[0]][side[1]] = "0"
        return

def play_game():
    board = [[" " for _ in range(3)] for _ in range(3)]
print("Welcome to tic tac toe!")
print_board(board)
```

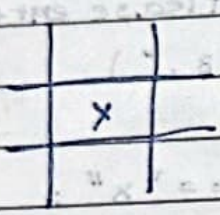
```
while True:
    while True:
        try:
            row = int(input("Enter row (1-3):"))-1
            col = int(input("Enter column (1-3):"))-1
            if board[row][col] == " ":
                board[row][col] = "x"
                break
            else:
                print("cell already taken, Choose
                       another.")
        except (ValueError, IndexError):
            print("Invalid input. Please enter numbers
                   between 1 and 3.")
    print_board(board)
    if check_winner(board) == "x":
        print("You win!")
        break
    if is_board_winfull(board):
        print("It's a draw!")
        break.

    print("AI's turn ...")
    ai_move(board)
    print_board(board)

    if check_winner(board) == "O":
        print("AI wins!")
        break
    if is_board_full(board):
        print("It's a draw!")
```

break

if __name__ == "__main__"
play_game()

Output

Enter row (1-3): 2
Enter row (1-3): 2



last matrix

O	X	O
O	X	
O	X	X

AI wins!

24/9/21

Date __/__/__
Page 7

LAB 2

Vacuum cleaner

Algorithm.

1) Create 2 rooms using classes

```
class room:
    def __init__(self, a):
        self.state = a

    def suck(self):
        self.state = 0 "clean"
```

2) Instansiate the class and take user input.

```
a = str(input("Room A state:"))
b = str(input("Room B state:"))

room_list = []
room_list.append(room(a))
room_list.append(room(b))
```

3 Perception sequence:

```
for i in room list:
    if (i.state == "dirty"):
        i.suck().
```

Proceed

code

```
class room :

    def __init__(self, a):
        self.state = a
    def suck(self):
        self.state = "clean"

n = 2
roomlist = []
for i in range(n):
    a = str(input(f"Enter room {i+1} state: "))
    roomlist.append(room(a))
start = int(input("Enter starting room number: "))

print("Before cleaning")
print "Room \t State "
    for i in range(len(roomList)):
        print(f"{i+1}\t{roomlist[i].state}")

count = 0
while count < len(roomlist):
    if(roomlist[start].state.lower() ==
                        "dirty"):
        roomlist[start].suck()
    start = (start + 1) % len(roomlist)
    count += 1

print("\n"
```

```
print ("After cleaning")
print ("Room\t State")

for i in range (len (roomList)):
    print (f"{i+1}\t{roomList[i].state}")
```

Output : (for 2)

Enter room 1 state : dirty
Enter room 2 state : dirty
Enter starting room number: 1

Before cleaning

Room	State
1	dirty
2	dirty

After cleaning

Room	state
1	clean
2	clean.

(for 4)

Enter room 1 state : dirty
Enter room 2 state : dirty
Enter room 3 state : dirty
Enter room 4 state : dirty
Enter starting room number: 1

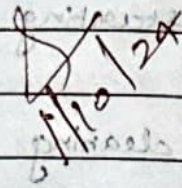
Date __/__/__
Page 10

Before cleaning

Room	state
1	dirty
2	dirty
3	dirty
4	dirty

After cleaning

Room	state
1	clean
2	clean
3	clean
4	clean.



Date __/__/__
Page 11

Lab 3

8 word puzzle using DFS and Manhattan distanc

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

In an 8 word puzzle who have 9 boxes with 8 with movable puzzle

DFS

```
start_state = [ ]
goal_state = = [ ]

stack = push (start_state)
visited_set = { }
moves = 0

f(i,j)
{
    visited_set.add (current_state)
    if (current_state == goal_state)
        return moves
    if (not in visited_set)
    {
        left = f(i, j-1)
        right = f(i, j+1)
        up = f(i-1, j)
        down = f(i+1, j)
    }
}
print (moves)
```

code:-

```
def manhattan (puzzle, goal):
    dist = 0
    for i in range (9):
        if puzzle [i] != 0
            goal_idx = goal.index ( puzzle [i])
            dist += abs (i//3 - goal_idx //3) +
                    abs (i%3 - goal_idx %3)
    return dist

def dfs_manhattan ( puzzle, goal, visited, path):
    if puzzle == goal:
        return path.
    visited.add (tuple (puzzle))
    idex = puzzle.index (0)
    moves = [(1, 3), (-1, 3), (3, 1), (-3, 1)]
    next_states = []
    for move, cond in moves:
        new_idx = idx + move
        if 0 <= new_idx < 9 and (new_idx //3 == idx //3 or new_idx %3 == idx %3):
            new_puzzle = puzzle [:]
            new_puzzle [idx], new_puzzle [new_idx] = new_puzzle [new_idx], new_puzzle [idx]
            if tuple (new_puzzle) not in visited:
                next_states.append ((new_puzzle, manhattan (new_puzzle, goal)))
    next_states.sort (key = lambda x: x[1])
    for state, _ in next_states:
        res = dfs_manhattan (state, goal, visited,
                             path + [state])
```

```
if res:
    return res
return None

def prettify(res):
    i = 0
    for j in range(3):
        for k in range(3):
            print(res[i], end = " ")
            i+1
        print("\n")
start = [1, 2, 3, 4, 0, 5, 6, 7, 8]
goal = [0, 1, 2, 3, 4, 5, 6, 7, 8]
result = dfs-manhattan (start, goal, set(),
                        [start])
for i in result:
    prettify(i)
    print("- - - -")
```

output

1

1	2	4
3	6	5
0	7	8

2

1	2	4
3	6	5
7	0	8

} 1st 2

3

3	1	2
0	4	5
6	7	8

4

0	1	2
3	4	5
6	7	8

} last 2

8/10/24

Date __/__/__
Page 14

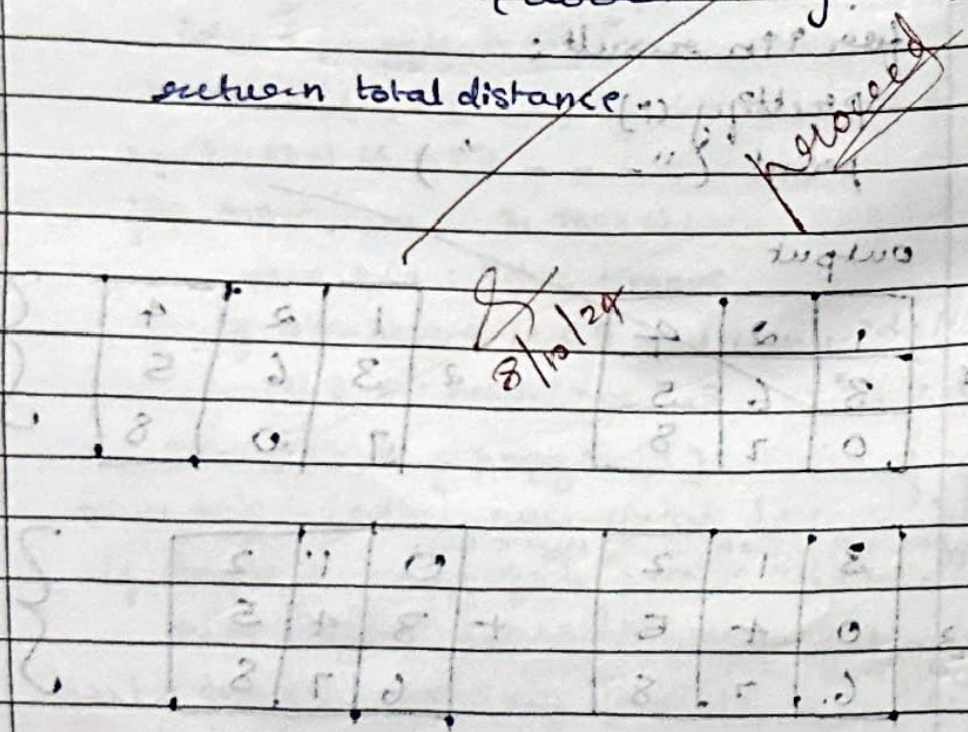
Manhattan distance

→ start at initial list of the given matrix (3×3)

→ compare each element in the index to the final state and see how far it is from the final state

manhatten (current-state, final_state)
    if the tile is not an blank-tile
    (current x, current y) = position of the current tile
        = (current x - goal x) + (current y - goal y)

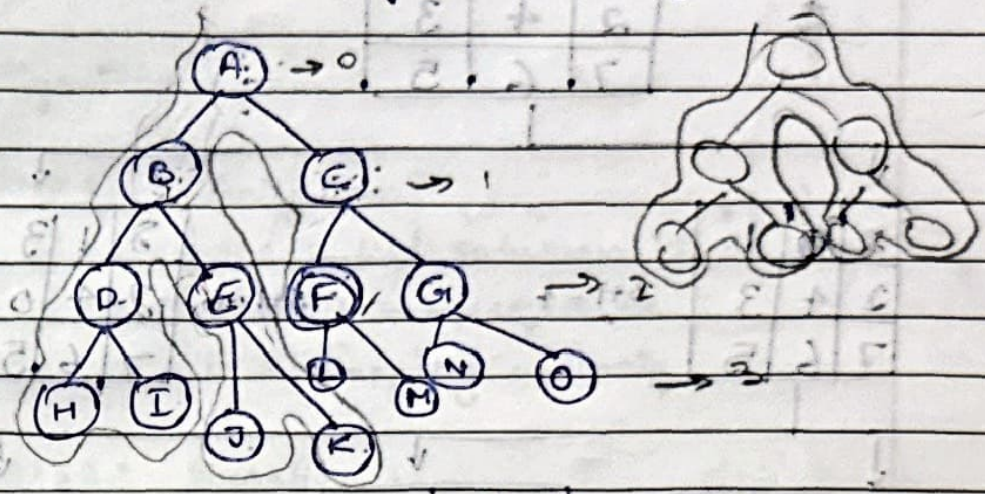
return total distance



Date 15/10/24
Page 15

LAB 4

1) Iterative deepening search algorithm.



Normal : A B A → B → D → H → I → E
J → K → C → F

Iteration 1 : (depth = 0) A

Iteration 2 : (depth = 1) A → B → C

Iteration 3 : (depth = 2) A → B → D → E
C → F

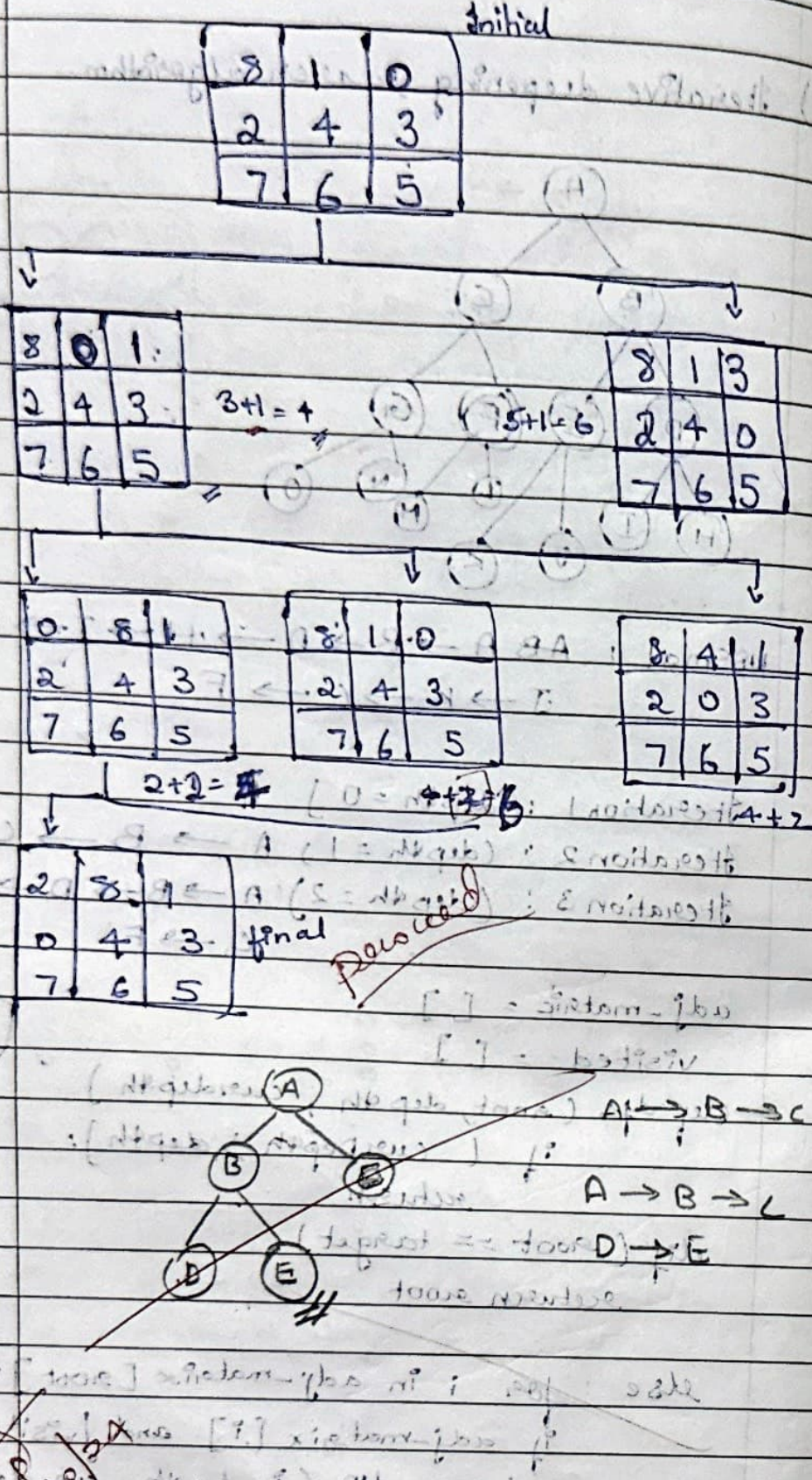
```
adj_matrix = [ ]
visited = [ ]
def dfs (root, depth, curdepth)
        if ( curDepth > depth):
            return
    elif ( root == target)
        return root
    else : for i in adj_matrix [root]:
        if adj_matrix [i]: and visited [i]:
            return dfs (i, depth, curdepth + 1)
```

visited [i] = 1

Paused

Date / /
Page 16

ii) A* (puzzle)



Date 22/10/24
Page 1

P ↑ T ↑
% difference is high P ↓

Lab 5

Annealing Algorithm.

Algorithm :

Step 1: Initialization
- Choose initial solution 'S'
- Set initial temperature T
- set a stopping criteria

Step 2: Iteration :
while
{
- Generate neighbours create neighbouring solution 'S1' for 'S'
- calculate energy difference : compute cost of E(S) & neighbouring E(S')

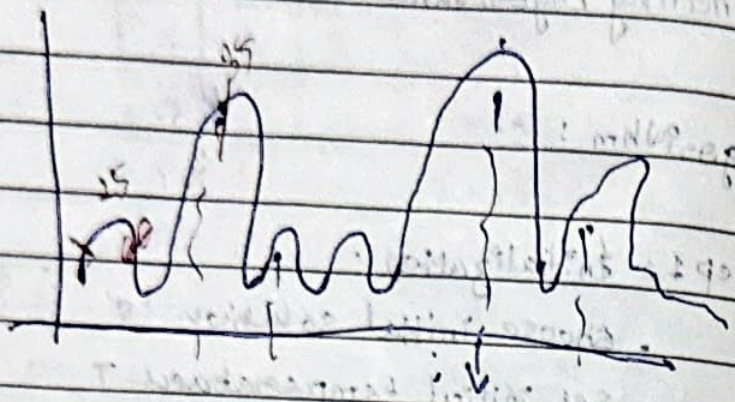
Step 3 : Acceptance criteria :
if (E(S1) < E(S)) → accept 'S' as new solution
else
{ accept 'S1' with probability

$$P = \exp\left(\frac{E(S) - E(S1)}{T}\right)$$

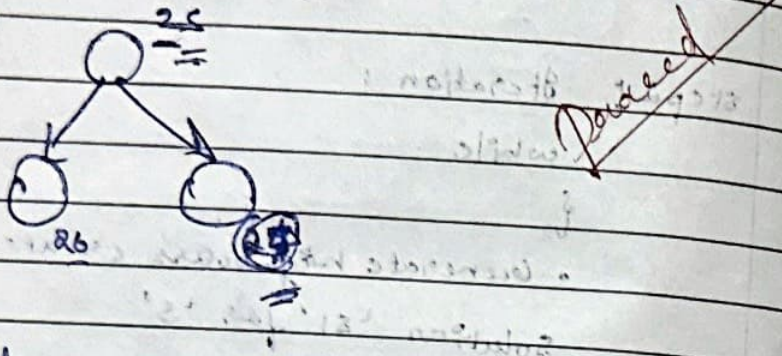
- update S to S' as new solution if accepted.

Date / /
Page 18

Step 4: Termination: Return the best solution (S or S') found



optimum solution.



Code

```
import math
import random

def objective_function(x):
    return 10*len(x) + sum([xi**2 - 10*math.cos(2*math.pi*xi) for xi in x])

def get_neighbor(x, step_size=0.1):
    neighbor = x[:]
    index = random.randint(0, len(x)-1)
    neighbor[index] += random.uniform(-step_size, step_size)
    return neighbor
```

```
def simulated_annealing(objective, bound, n_iterations,
                        step_size, temp):
best = [random.uniform(bound[0], bound[1]) for
        bound in bounds]
best_eval = objective(best).
current, current_eval = best, best_eval.
scores = [best_eval]:

for i in range(n_iterations):
    t = temp / float(i+1)
    if candidate_eval < best_eval or random
        candidate_eval / t)
        if candidate_eval < best_eval:
            scores.append(best_eval).

if i % 100 == 0:

return best, best_eval, scores

bounds = [(-5.0, 5.0) for _ in range(2)]
n_iteration = 1000
step_size = 0.1
temp = 10

best, score, scores = simulated_annealing(
objective_function, bound, n_iterations,
step_size, temp)

print('best')
print('score')
```

Date __/__/__
Page 20

Iteration 0	Temperature 1.0	Best evaluation	41.27
100	0.099		16.93
200	0.050		16.920
300	0.033		16.916
400	0.025		16.916
500	0.020		16.915
600	0.017		16.915
700	0.014		16.914
800	0.012		16.914
900	0.011		16.914

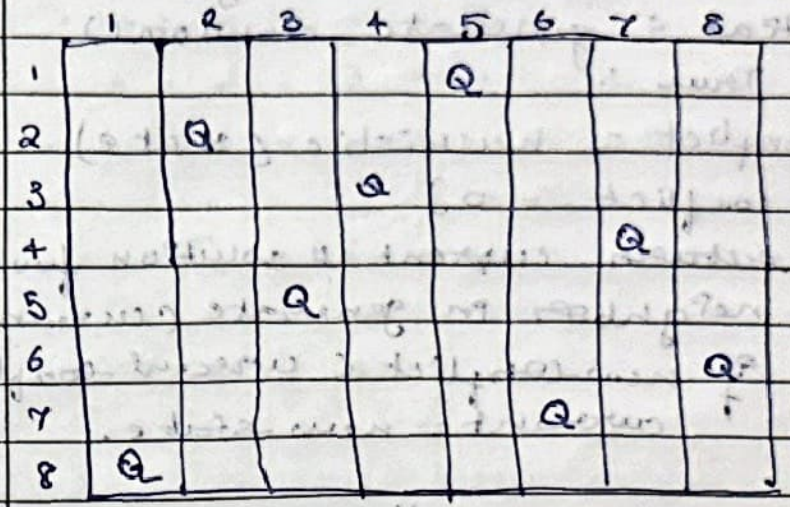
Best solution [-1.98904173998868, -1.975245 2551 4428]

Best score : 28.855376424762137 1

Date 29/10/24
Page 21

Lab 6

1) A* implementation for 8-queens.



eg. 8×8 board 8-queen

```
function 8_Queen_A star ():
    initial = random_position ()
    open_list = priority_Queue ()
    open_list.push (initial, heuristic ())

    while (open_list != 0):
    current = open_list.pop_min()
        if (heuristic (current) == 0)
            return current  # Solution found
        for neighbor in generate (current):
            open_list.push (current, heuristic)
    return "No solution".
function heuristic ()
    count = 0
    if (same row || same column || same
                                    diagonal)
        count = 1
    return count.
```

Date / /
Page 22

ii) 8- Queens - hill climbing

```
function 8-Queen_Hill-Algo():*
    initial = generate_random()
while True:
    conflict = heuristic(state)
    if (conflict == 0)
        return current # solution found
    for neighbor in generate(current):
        if new_conflict < current_conflict:
            current = new_state.

return "No solution"

function heuristic(state):
    count = 0
if (same row | same column | same diagonal):
        count ++
    return count.
# the might not find the solution as
this algorithm.
```

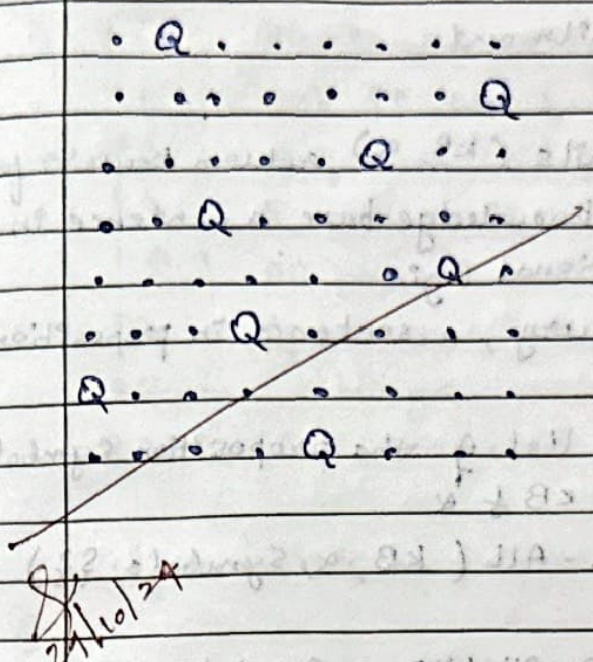
Revised

output for A*

Q . . . . . .
. . . . . Q .
. . . . Q . .
. . . . . Q
. Q Q . . . . .
. . . Q Q . . .
. . . . . Q . .
. . Q . . . . .

Date __/__/__
Page 23

Output for hill climbing.



Date 12/11/24
Page 24

Lab 7

Propositional entailment

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 symbol in KB & α
return TT-check-All (KB, α, symbols, {})

function TT-check-All (KB, α, symbols, model)
return 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}))

Date __/__/__
Page _____

Knowledge Base

1] Premise

R1 : Alice is the mother of Bob

R2 : Bob is the father of Charlie

R3 : A father is a parent.

R4 : A mother is a parent.

R5 : All parents have children

R6 : If someone is a parent, their children are siblings

R7 : Alice is married to Bob.

Hypothesis

2] "Charlie is a sibling of Bob"

3] Entailment Process

R2 → R3 ⇒ Bob is a parent

R5 → Bob has a child since he is a parent

R2 → Charlie is a child of Bob

R6 → If someone is a parent, their children are siblings

Therefore Charlie being a child of Bob cannot be sibling.

Date __/__/__
Page _____

Lab 8

Implementation of first Order logic

Translate natural language sentences into first Order logic

1] "John is a human"
→ Human (John)

2] "Every human is mortal
→ ∀x (Human (x) → mortal (x))

3] "John loves Mary"
→ Loves (John, Mary

4] "There is someone who loves mary".
∃ x (Loves (x, Mary))

5] "All dogs are animals"
→ ∀x (dogs (x) → animals (x)

6] "Some dogs are brown"
→ ∃ x (Dog (x) ∧ Brown (x)).

~~Translatio~~

Eats (x, Apple)

Eats (Riya, y)

Eats (Riya, Apple)

Date __/__/__
Page _____

Algorithm

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 ~~$\{(\Psi_1 / \Psi_2)\}$~~ Failure

Step 2 : If the initial Predicate symbol in Ψ_1 & 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 returns Failure,

c) If S ≠ NIL then do,

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

b. SUBST = Append (S, SUBST)

Step 6 : Return SUBST

Prabhleen

Date __/__/__
Page

Output-

Enter a sentence like
1. John is a human
2. Everyone human is mortal
3. John loves
4. There exists someone who loves many

Enter a sentence : John loves many
First order logic Translation : L(john, m

19/11/24

Date / /
Page

Lab 9

Forward Reasoning Algorithm

function FOL-FC-ASK (KB, α) return a substitution or false
input: 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
    (p1 ∧ ... ∧ pn ⇒ q) ← Standardize-Variables (rule)
    for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) =
      SUBST(θ, p'1 ∧ ... ∧ p'n)
      for some p'1 ... p'n in KB
    q' ← SUBST(θ, q)
    if q' does not unify with some sentence already in KB or new then
      add q' to new
      φ ← UNIFY(q', α)
      if φ is not fail, then return φ
  add new to KB
return false.

Date __/__/__
Page

→ It is a crime for an American to sell weapons to hostile nations

→ Let's say p, q & 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, introducing new constant T1:

Owns(A, T1)

missile(T1)

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

∀x Missile(x) ∧ owns(A, x) ⇒ sells(Robert, x, A)

→ Missile ~~(x) ⇒ weapons~~

Missile(x) ⇒ weapon(x)

→ Enemy of America is known as hostile

∀x enemy(x, America) ⇒ Hostile(x)

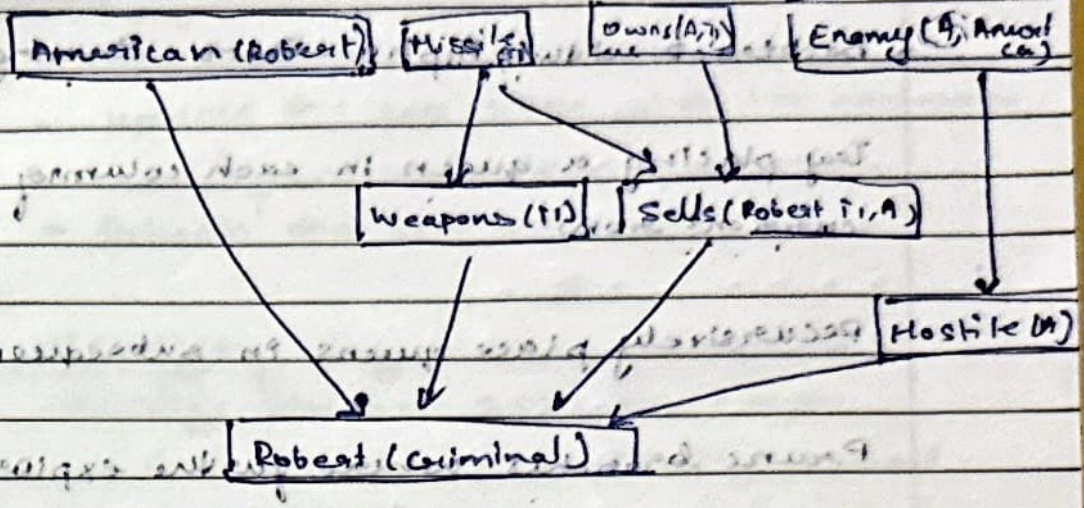
→ Robert is an American

American(Robert)

→ The country A, an enemy of America

Enemy(A, America)

Date __/__/__
Page _____



The Alpha - Beta Search algorithm for solving 8 Queens Problem

Algorithm.

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

→ Define is_safe function to check if placing a queen at (row, column) is safe. i.e, no 2 queens threaten each other (same column or row or diagonal)

→ Alpha - Beta Search :
- Alpha : Best score found for maximizing player (valid placement of queens)
- Beta : Best score found for minimizing player (invalid state)

Date __/__/__
Page

→ Backtrack with Alpha-Beta Pruning:

Try placing a queen in each column of the current row.

Recursively place queens in subsequent rows

Prune branches where the exploration is unnecessary (based on alpha & beta values)

→ Return solution: If solution is found, return the board configuration

Minimax Algorithm for Tic-Tac-Toe

→ check if the game is over

- If a player wins, return the score:
  - +1 for 'X' win
  - -1 for 'O' win
  - 0 for a draw

→ If the board is full & no one has won, return 0 (draw)

→ Maximizing player ('X')

- Initialize the best score to -infinity
- For each available move on the board
  - Make the move (place 'X')
  - Call the minimax function recursively, switching to the minimizing player ('O')

Date __/__/__
Page _____

- Undo the move
- Update the best score with the maximum value from the recursive call.
- Return the best score.

→ Minimizing Player (O):
- Initialize the best score to +∞.
- For each available move on the board:
  - Make the move (place 'O').
  - call minimax function recursively, switching to the maximizing player (X).
  - Undo the move.
  - Update the best score with the minimal value from the recursive call.
- Return the best score.

→ Find the best move for the maximizing player (X):
- For each available move, evaluate the move using the minimax function.
- Return the move with the highest score.

Passed

Output for 8 queens

. Q . . . . . .
. . . . Q . . .
. . . . . Q .
Q . . . . . .
. . Q . . . . .
. . . . . Q
. . . Q . . . .

Output B for TIC TAC TOE

Current Board:

X O X
O X O
_ _ _

The best move for AI is: (2,0)

3/12/24