Sanketh. M. Hanasi
1BM22CS242
Artificial Intelligence

SI.NO	Date	Title	Sign
1	24/09/24	Tic-Tac-Toe	
2	1/10/24	Vacuum Cleaner	
3	8/10/24	8 puzzle - BFS, DFS	
4	15/10/24	8 puzzle - A* implementation	
5	22/10/24	8 puzzle - IDS N-Queens - Hill climbing	
6	29/10/24	Simulated Annealing	
7	12/11/24	Proposition Logic	
8	19/11/24	Unification for FOL	
9	26/11/24	Forward Chaining/Reasoning	
10)	26/11/24	FOL to Resolution	
11)	3/12/24	Alpha Beta Pruning	

1) Tic-Tac-Toe:

Algorithm:

1) Initialize the game board:
- create 3x3 matrix
- Each position in the board is initialized to '-'

2) Start the loop:
- The game runs for a maximum of 9 turns
- The game alternates between two players, "X" and "O"

3) Display the board:
- Print the current status of the board

4) Player Input:
- The current player selects the position
- The selected position is checked

5) Update the board:
- Place the current player mark in the selected positi

6) Check for win:
- After every move, check if the current player has won
- Any row or Any column has same all the same mar
  or either of the diagonal has all the same marl

7) Check for draw:
- If all 9 positions are filled, then it is a draw.

8) If a player wins or a draw, then end the game.

24-9-24

1/10/24

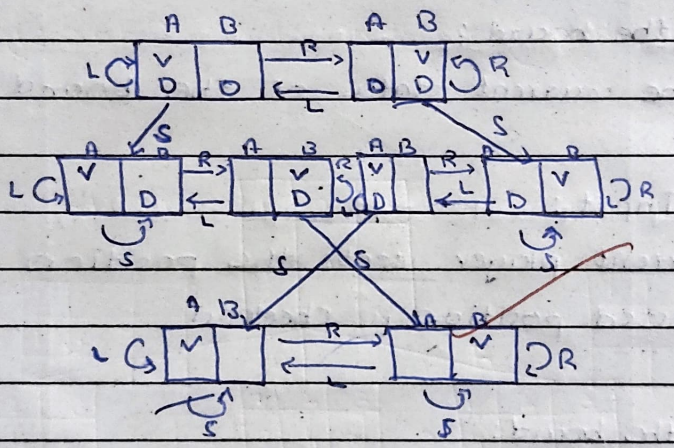
2) Implement vaccum world problem

function REFLEX-VACCUM-AGENT([LOCATION, status]) returns an action

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

State space diagram of vaccum world.

function simple

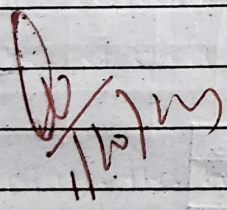


Problem formulation steps :

* States
* Initial state
* Actions
* Transition model
* Goal test
* Path cost

Algorithm :

1) Initialize room and cost
2) Checks the room are clean or not
   If room is dirty clean the room and move to the next location.
3) Next checks whether previously cleaned is room is dirty if dirty clean.
4) If both the rooms are clean then return the total cost and off the vacuum cleaner.



3) Solve 8 puzzle problem using DFS & BFS

4) Implement depth limited search algorithm & Uniform cost search algorithm

5) Implement iterative deepening search

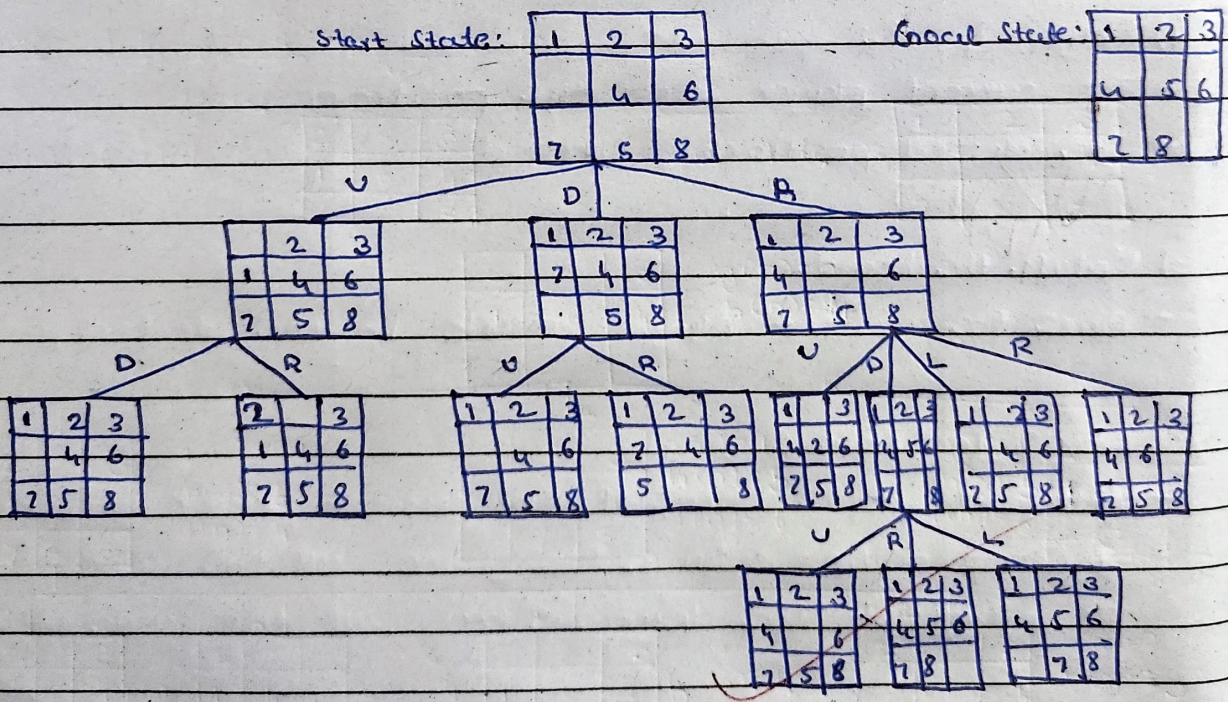
Breadth-First-Search:

Complete ? Yes

Optimal ? Yes, if path cost is nondecreasing function of depth

Time complexity : $O(b^d)$

Space Complexity : $O(b^d)$, node that every node in the fringe is kept in the queue.



3) Solve 8 puzzle problem using DFS & BFS.

BFS:

- The initial state & goal state will be given.
- We need to achieve goal state of 8 puzzle using BFS algorithm.
- Initially the parent node will be the initial state, the possible movements of blank space will be recorded i.e, up, down, right, left.
- These possible movements will be the child nodes of initial state.
- Define helper function Find_blank_title(state) to locate the position of blank tile
- Define another function get_possible_moves(state) that generates all valid moves for the current state:
  - Determine the position of the blank tile
  - For each possible direction (up, down, left, right), record direction & move the tile.
- Run the BFS loop until we reach the goal state.

Q) For 8 puzzle Problem using A* implementation, to calculate f(n) using a) misplaced tiles b) Manhattan Distance a) g(n) - depth of node b) h(n). 15/10/24

A*- Misplaced Tiles:

1) Define the initial & goal state.
2) Define list & $f(n) = g(n) + h(n)$ where $g(n)=0$, & $h(n) \rightarrow$ misplaced tiles where $g(n)$ is cost to reach each node & $h(n)$ is heuristic value.
3) While list is not empty, extract node with lowest frequency.
- If current state is goal state return path & add the current state to closed set.
4) Generate new state by moving up, down, left, right, if new state is not in closed set, calculate $g(n)$ for new state & $h(n)$ & $f(n)$. Continue the process until goal is reached.

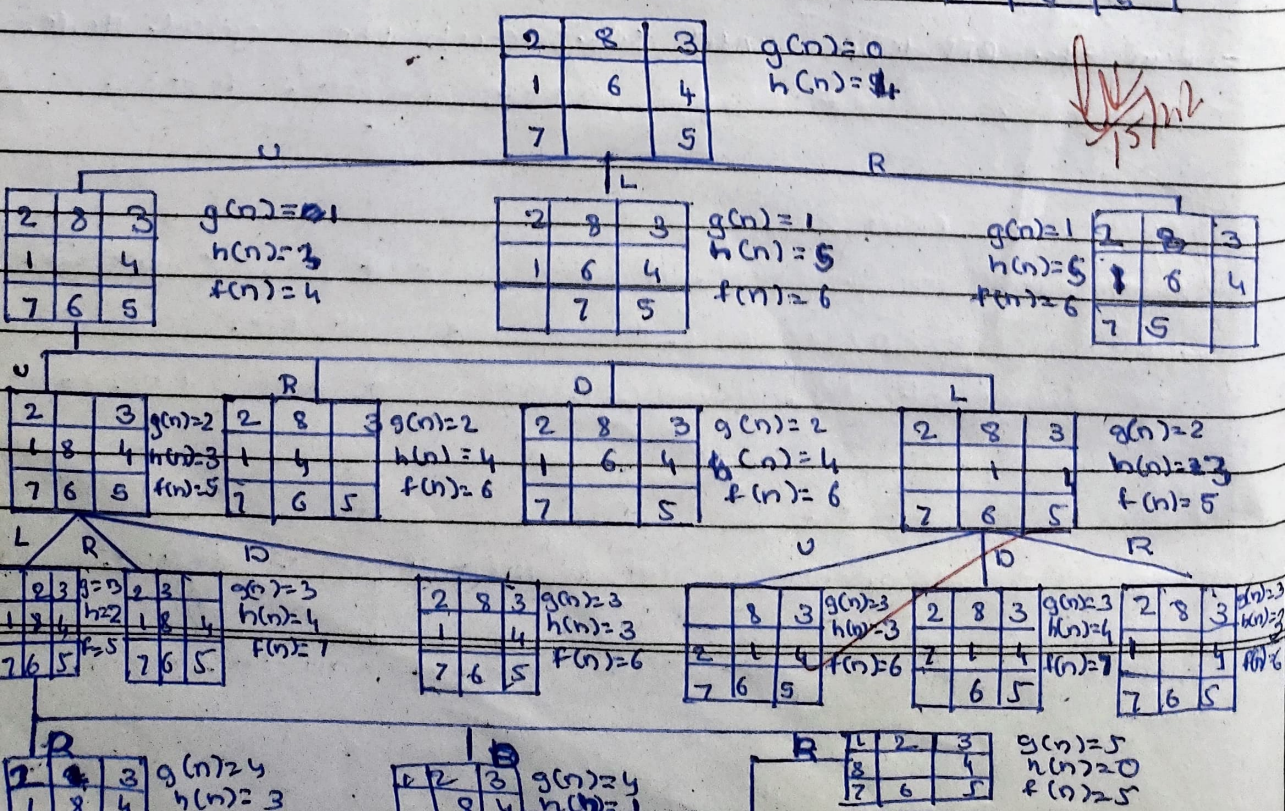
Give state space diagram for A* Misplaced Tiles:

Initial state:

2	8	3
1	6	4
7		5

Goal state:

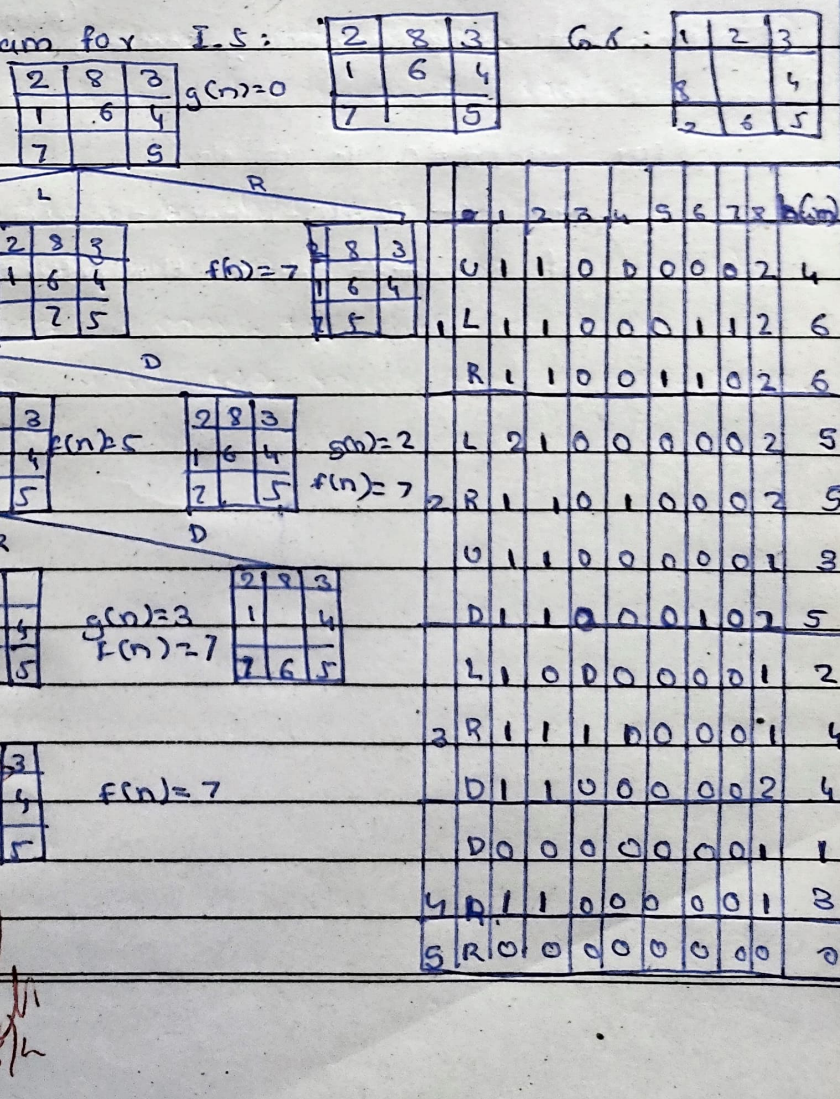
1	2	3
8		4
7	6	5



Manhattan Distance:

1) Define goal state & also create a dictionary.
2) Create a list containing $f(n)$, $g(n)$ & path with $g(n)=0$.
   Define a closed set to track visited states.
3) Calculate manhattan distance,
   - for each tile from 1 to 8 in current state
   - find this current position & find target by goal state & calculated distance.
   - sum these distances to get $h(n)$.
4) Expand nodes, while list is not empty, extract node with low frequency & return path.

*Give State space diagram for I.S:



		0	1	2	3	4	5	6	7	8	h(n)
	U	1	1	0	0	0	0	0	2	4
1	L	1	1	0	0	0	1	1	2	6
	R	1	1	0	0	1	1	0	2	6
	L	2	1	0	0	0	0	0	2	5
2	R	1	1	0	1	0	0	0	2	5
	U	1	1	0	0	0	0	0	1	3
	D	1	1	0	0	0	1	0	2	5
	L	1	0	0	0	0	0	0	1	2
3	R	1	1	1	0	0	0	0	1	4
	D	1	1	0	0	0	0	0	2	4
	D	0	0	0	0	0	0	0	1	1
4	R	1	1	0	0	0	0	0	1	3
5	R	0	0	0	0	0	0	0	0	0

Lab 5: Implement Iterative Deepening Search Algorithm.

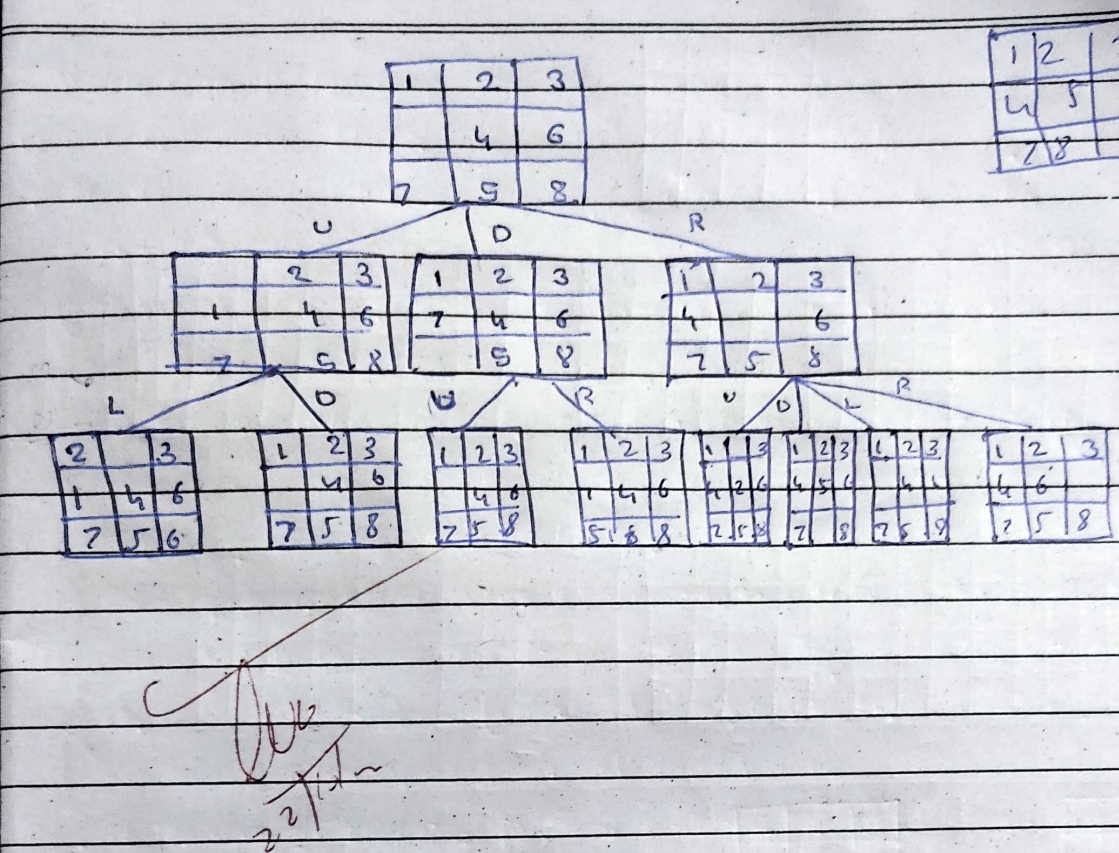
1) For each child of the current node
2) If it is the target node return
3) If the curr max depth is reached return
4) Set the current node to this node & go back to 1
5) After having gone through all children go to the next child of the parent (the next sibling)
6) After having gone through all children of the 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.

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


U
D
R
L
D
U
R
U
D
L
R

Q) Hill climbing search Algorithm to solve N Queens problem

Hill climbing search Algorithm :

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

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

loop do

neighbour ← a highest-valued successor of current

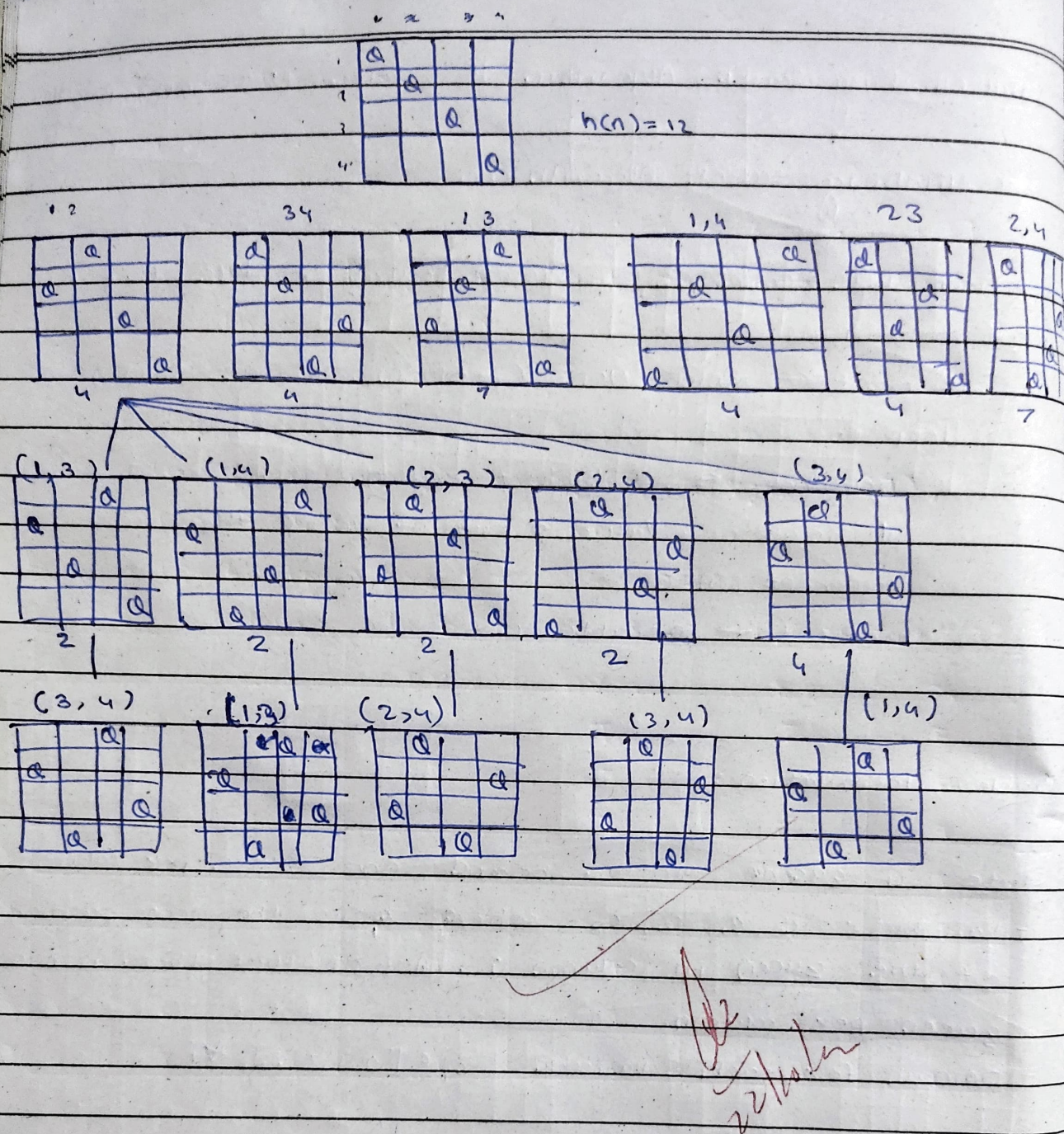
if neighbour.Value ≤ current.Value then return current.STATE

current ← neighbour

Problem Formulation :

- State : 4 queens on the board one queen per column
  - 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 position of two queens
- Cost function : The no of pairs of queens attacking each other, directly or indirectly.


h(n) = 12
12
34
13
1,4
23
2,4
4
4
7
4
4
7
(1,3)
(1,4)
(2,3)
(2,4)
(3,4)
2
2
2
2
4
(3,4)
(1,3)
(2,4)
(3,4)
(1,4)

6) 29/10/24

a) 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 $e^{\Delta E/T}$

SA - Steps:
1. Start at a random point x
2. Choose a new point x_j, on a neighborhood 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)

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

4. Reduce T

8-Queens Problem

Output:

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

The number of queens that are not attacking each other is : 8.0

TSP :

Result structure : (array([1, 0, 3, 5, 4, 2]), 21.029348853206, None)

Best route found : [1 0 3 5 4 2]

Total distance of best route : 21.029348853206

* Wumpus World using Proposition logic

KB + α → query
(, Knowledge base

$P_{x,y}$ - pit in [x,y]
$W_{x,y}$ - wumpus in [x,y]
$B_{x,y}$ - Breeze in [x,y]
$S_{x,y}$ - stench in [x,y]

$R_1 : \neg P_{1,1}$
$R_2 : B_{1,1} \iff P_{1,2} \vee P_{2,1}$
$R_3 : B_{2,1} \iff (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

function TT-ENTAILS? (KB, α) returns true or false
inputs : KB, the knowledge base, a sentence in PL
α, the query, a sentence in PL

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

function TT-CHECK-ALL (KB, α, symbols, model) returns true or false
if EMPTY? (symbols) then
if PL-TRUE? (KB, model) then return PL-TRUE? (α, model)
else return true // when KB is false, always return true
else do

P ← FIRST (symbols)

rest ← REST (symbols)

return (TT-CHECK-ALL (KB, α, rest, model ∪ {P=true}))

and

TT-CHECK-ALL (KB, α, rest, model ∪ {P=false}))

Propositional Inference : Enumeration Method

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

checking that $KB \models \alpha$

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

O/P seen

SSD

12/11/24

LAB-7

8) 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 & ψ_2 have a diff no of args, then return FAILURE

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

Step 5 : For i=1 to the no of ele in ψ_1.
a) Call Unify function with the ith ele of ψ_1 & put the result into S
b) If S= failure then return Failure
c) If S ≠ NIL then do,
a). Apply S to the remainder of both L1 & L2.
b) SUBST = APPEND(S, SUBST)

Step 6 : RETURN SUBST.

$P(x, F(y))$ → ①

$P(a, F(g(x)))$ → ②

① & ② are identical if x is replaced with a in ①

$P(a, F(y))$ → ①

if y is replaced with $g(x)$

$P(a, F(g(x)))$ → ①

$Q(a, g(x,a), f(y))$ → ①

$Q(a, g(f(b),a), x)$ → ②

$f(b)$ is replaced with x in ②

$Q(a, g(x,a), x)$

x is replaced with $f(y)$ in ②

$Q(a, g(x,a), f(y))$ → ③

① $\Psi_1 = P(f(a), g(y))$ $\Psi_2 = P(x, x)$

→ SUBST $(f(a)/x)$

$\Psi_1 = P(f(a), g(y))$ $\Psi_2 = P(f(a), f(a))$

→ SUBST $(f(a)/g(y)$

Failed

② $\Psi_1 = P(b, x, f(g(z)))$ $\Psi_2 = P(z, f(y), f(y))$

→ SUBST (b, z)

$\Psi_1 = P(b, x, f(g(b)))$ $\Psi_2 = P(b, f(y), f(y))$

→ SUBST $(f(y), x)$

$\Psi_1 = P(b, f(y), f(g(b)))$ $\Psi_2 = P(b, f(y), f(y))$

LAB-8

a) Create a knowledge base consisting of FOL statement & prove the given query using forward reasoning

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 iterations

repeat until new is empty
new ← { }
for each rule in KB do
$(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES (rule)
for each rule θ such that SUBST(θ, $p_1' \wedge \dots \wedge p_n'$) = SUBST (θ, $p_1 \wedge \dots \wedge p_n$)
for some $p_1', \dots, p_n'$ in KB
$q' \leftarrow$ SUBST (θ, q)
if q' does not unify with some sentence already in KB or new then
add q' to new
$\phi \leftarrow$ UNIFY (q', α)
if ϕ is not fail then return ϕ
add new to KB
return false

Representation in FOL:

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 a new constant T1 : Owns(A, T1), missile(T1)

→ All the missiles were sold to country A by Robert

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

→ Missiles are weapons : Missiles(x) ⇒ weapon(x)

→ Robert is an american : American(Robert)

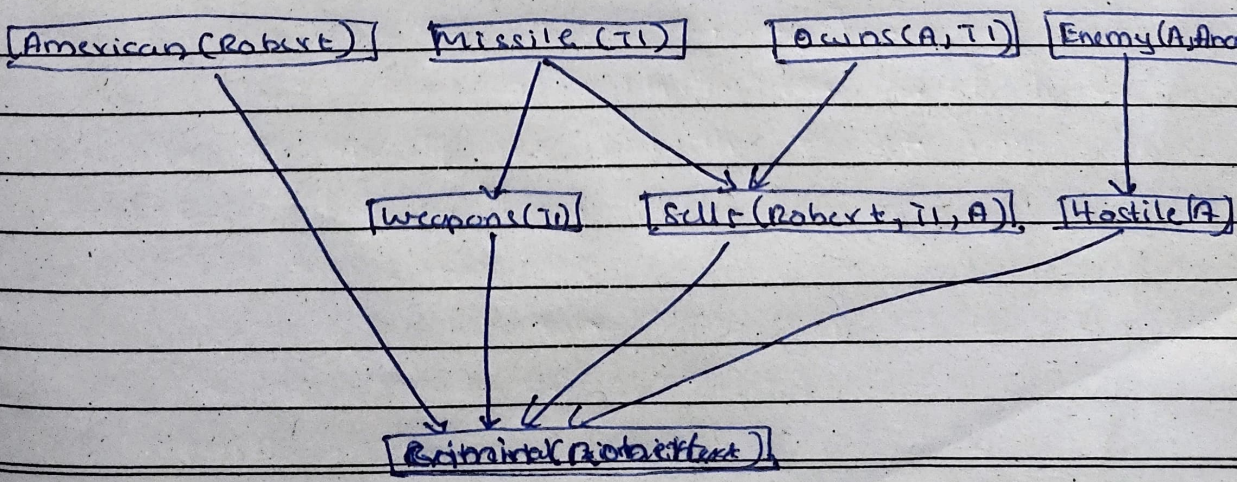
→ Enemy of America is known as hostile

∀ x Enemy(x, America) ⇒ Hostile(x)

→ The country A, an enemy of America

Enemy(A, America)

To Prove: Robert is a criminal : Criminal(Robert)



American(Robert) ∧ ... upons(T1) ∧ Sells(Robert, T1, A) ∧ Hostile(

⇒ Criminal(Robert)

26/11/24 LAB 9

8) Convert a given first order logic statement into Resolution:

Basic steps for proving a conclusion S given premises Premise, ... Premise n

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 resolvent is the empty clause, a contradiction has been found
   d. If not, add resolvent to the premises

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

Representation in FOL:

Given KB or premises:

a. John likes all kind of food — a. $\forall x : food(x) \rightarrow likes(John, x)$

b. Apple & vegetables are food — b. $food(Apple) \wedge food(vegetables)$

c. Anything anyone eats & not killed is food — c. $\forall x \forall y : eats(x,y) \wedge \neg killed(x) \rightarrow food(y)$

d. Anil eats peanuts & still alive — d. $eats(Anil, peanuts) \wedge alive(Anil)$

e. Harry eats everything that Anil eats — e. $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$

f. Anyone who is alive implies not killed — f. $\forall x : \neg killed(x) \rightarrow alive(x)$

g. Anyone who is not killed implies alive — g. $\forall x : alive(x) \rightarrow \neg killed(x)$

h. Prove by resolution that: John likes peanuts — h. $likes(John, peanuts)$

1) Eliminate Implication:

a) $\forall x \neg food(x) \vee likes(John, x)$

b) $food(Apple) \wedge food(vegetables)$

c) $\forall x \forall y \neg [eats(x,y) \wedge \neg killed(x)] \vee food(y)$ → c. $\forall x \forall y \neg eats(x,y) \vee killed(x) \vee food(y)$

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

e) $\forall e \neg eats(Anil, a) \vee eats(Harry, x)$

f) $\forall x \neg [\neg killed(x)] \vee alive(x)$ → f. $\forall x [killed(x)] \vee alive(x)$

g) $\forall x \neg alive(x) \vee \neg killed(x)$

h) $likes(John, Peanuts)$

2) Move Negation ($\neg$) inwards & write

3) 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. $likes(John, Peanuts)$

4) Drop Universal Quantification

a. $\neg food(x) \vee likes(John, x)$

b. $food(Apples)$

c. $food(vegetables)$

d. $\neg eats(y,z) \vee killed(y) \vee food(z)$

e. $eats(Anil, Peanuts)$

f. $alive(Anil)$

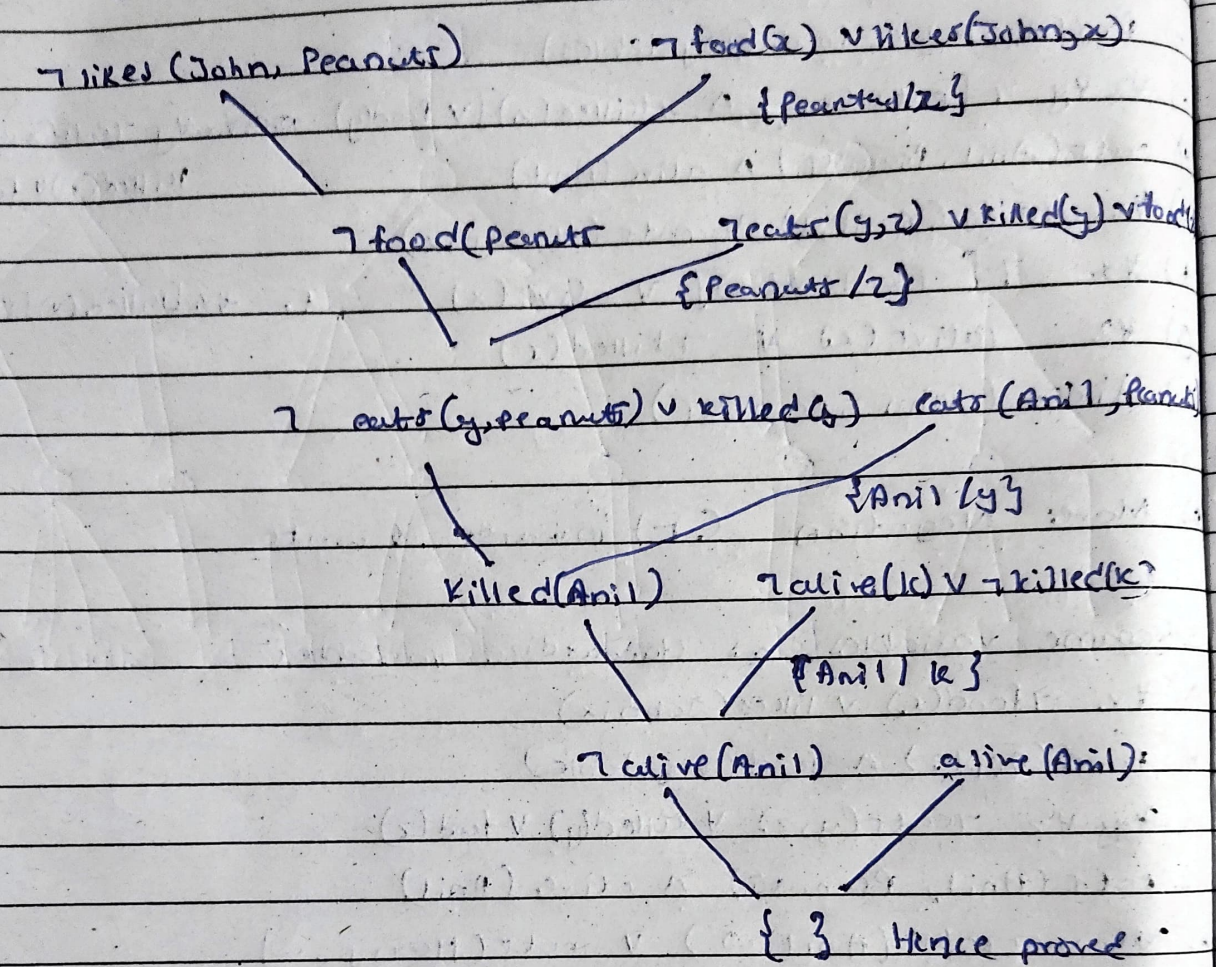
g. $\neg eats(Anil, w) \vee eats(Harry, w)$

h. $killed(g) \vee alive(g)$

i. $\neg alive(k) \vee \neg killed(k)$

j. $likes(John, Peanuts)$

Proof by Resolution:



LAB 10

26/11/24

Implement Apha Beta Pruning Algorithm

Alpha(α) - Beta(β) process to find the optimal path without looking at every node in the game tree.

max contains Alpha(α) and min contains Beta(β) bound during the calculation

In both MIN and MAX node, we return α, β which compares with its parent node only

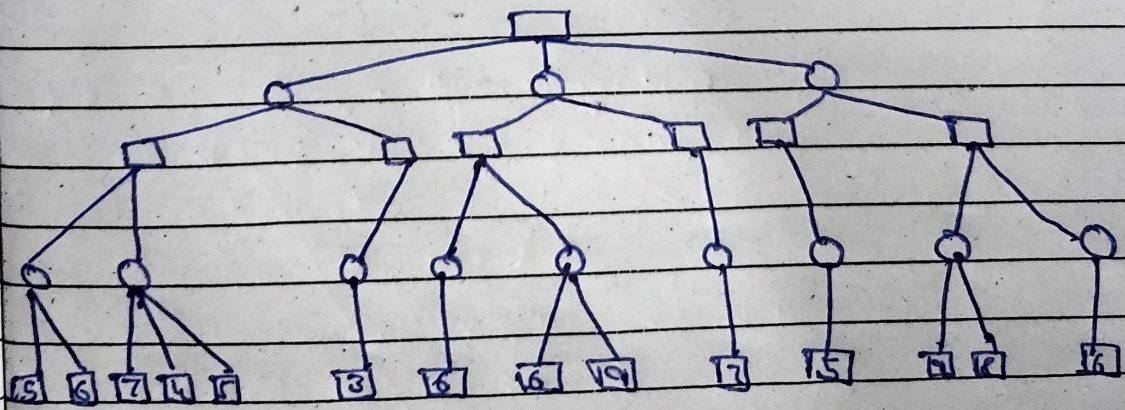
Both minimax & Alpha(α) - Beta(β) cut-off give same path.

Alpha(α) - Beta(β) gives optimal solution as it takes less time to get the value for the root node

MAX

MIN

Input:



Output : □ Max ○ Min