INDEX

NAME : VATSAL MURAL

STD. : _____ DIV. : _____ ROLL NO. : _____

SUBJECT : _____

Sl.No.	Date	Tittle	Page No.	Remarks
01	4/10/24	LAB 01 [Tic Tac Toe]	10	
02	18/10/24	LAB 02 [Vacuum cleaner & Puzzle]	10	
03	25/10/24	LAB 03 [A* Algorithm]	10	
04	8/11/24	Hill climb search algorithm	10	✓ 8/11/24
05	15/11/24	Stimulated Annealing using N-Queen	10	✓ 15/11/24
06	22/11/24	Unification	10	✓ 22/11/24
07	29/11/24	Forward Reasoning	10	✓ 29/11/24
08	19/11/24	Alpha Beta Pruning	10	✓ 29/12/24
09	19/11/24	Converting FOL to CNF	10	✓ 20/12/24
10	19/11/24	Creating KB using Propositional Logic to Prove using Resolution.	10	✓ 20/12/24
11	19/11/24	Knowledge Base using Propositional logic	10	✓ 20/12/24

4/10/23

AI

Define Tic Tac Toe

Pseudocode

Function Minimax(Nodes, depth, min, max)

if node is a terminal state:

return evaluate (node) -> check if the game has end.

if isMaximizing Player:

bestValue = -infinity

for each child in node:

Value = minimax(child, depth+1, false)

bestValue = max(bestvalue, value

return bestvalue

else:

bestValue = +infinity

for each child in node:

Value = minimax(child, depth+1, true)

bestValue = min(bestValue, value)

return bestValue

8 4/10/23

node : current game state

depth : current depth

isMaxplayer : boolean indicating where it is maximizing player or not.

LAB-02

12/10/2024

Vaccum Cleaner:

Pseudo Code

```
Function Vaccum_World():
    SET goal_state = {'A':'0','B':'0'}
    SET cost = 0

    PROMPT "Enter Location of Vaccum (A or B):"
    READ location_input
    PROMPT "Enter Status of {location_input}
                (0 for clean, 1 for Dirty):"
    READ status_input
    PROMPT "Enter status of other room:"
    READ status_input_complement

IF location_input == "A":
    IF status_input == "1"
        CLEAN A
        IF status_input_complement == "A":
            Move to B
            CLEAN 'B
        ELSE IF status_input_complement == "
            MOVE to B
            CLEAN B

ELSE IF location_input == 'B':
    IF status_input == '1';
        CLEAN B
```

```
IF status_input == '1';
    MOVE to A
    CLEAN
ELSE IF status_input == '1';
    MOVE to A
    CLEAN A

PRINT "GOAL STATE :" ;goal_state
PRINT "Performence Measurement :" cost

CALL Vacuum_World()
```

18/10/24

8 Puzzle

BFS Algorithm

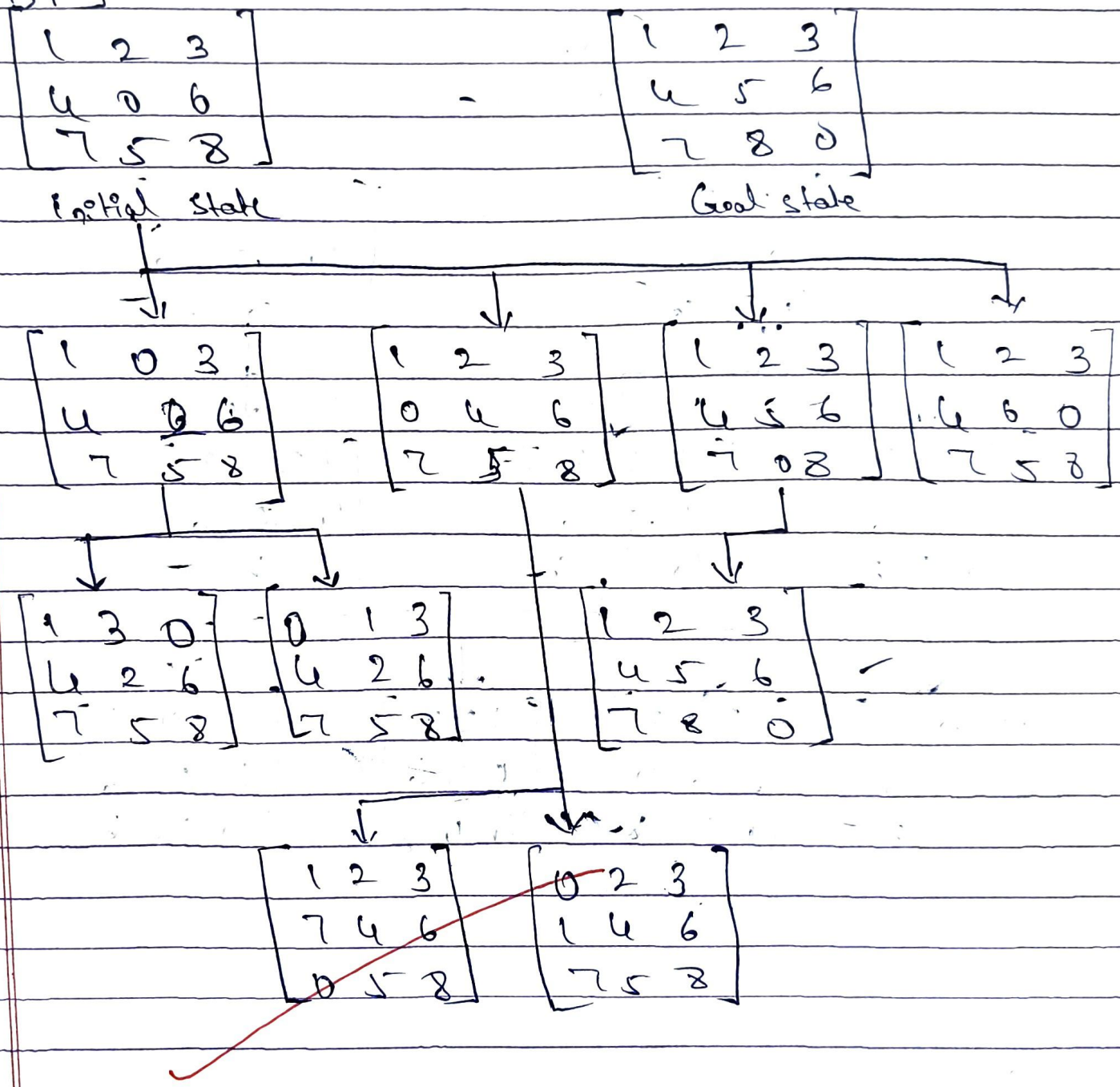
LOOP
if fringe is empty return failure
Node ← remove-first (fringe)
if Node is a goal
then return the path from initial state to final state
else, generate all successors of Node,
and add generated node to the back of fringe
End LOOP

DFS Algorithm

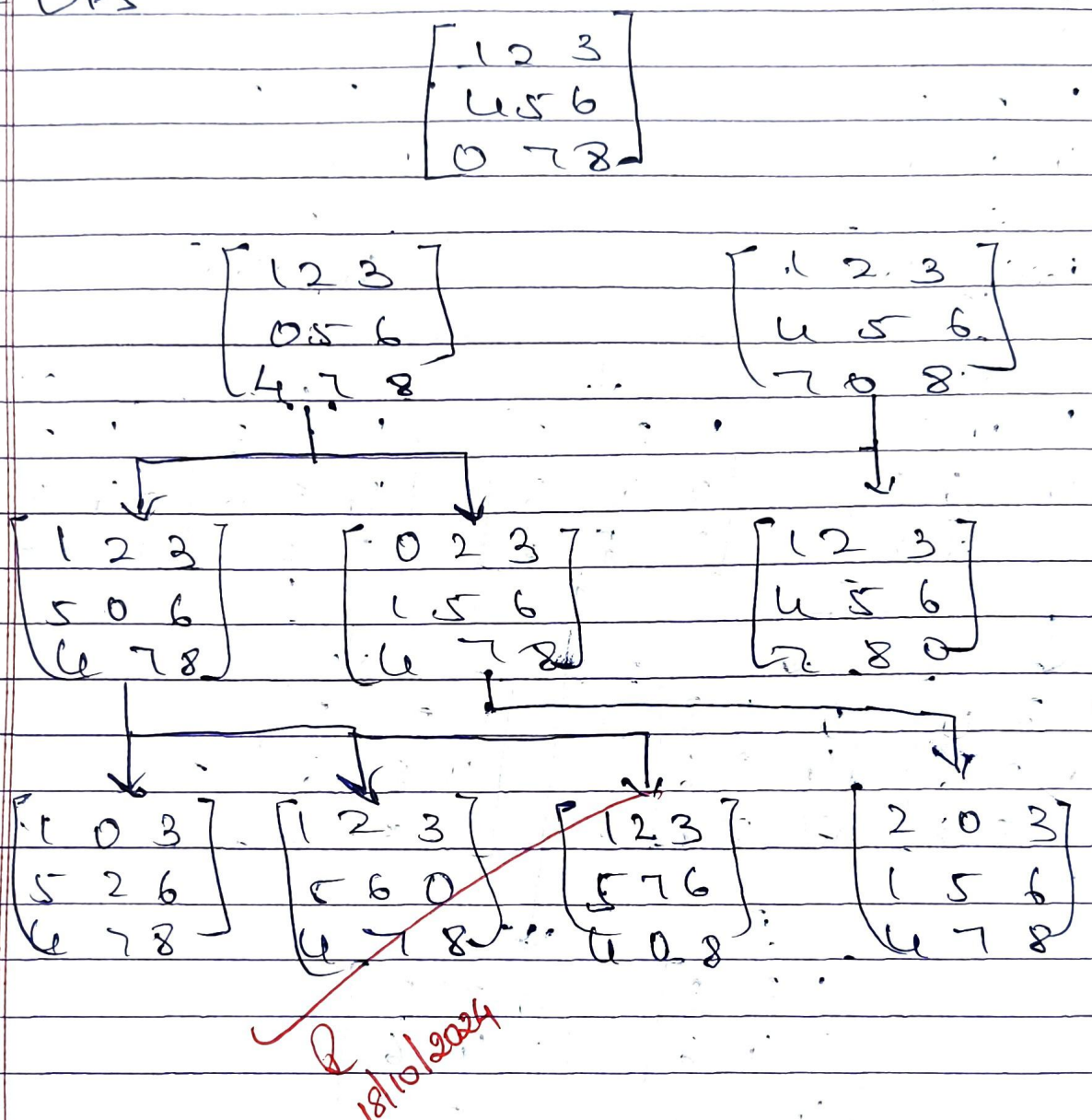
LOOP
if fringe is empty return failure
Node ← remove-first (fringe)
if Node is a goal
the return the path from initial state to final state
else generate all successor of Node
And add generated node to the front of fringe
End LOOP

State Space Tree

1) BFS



2 DFS



18/10/2024

LAB-03

A* algorithm

function A* Search (problem) return a solution or failure

node ← a node n with n-state-problem initial state, n.g-0

frontier ← a Priority queue ordered by ascending g^th only element n.

loop do

if empty? (frontier) then return failure

n ← pop (frontier)

if problem. ~~global~~ goal Test (n:,state) then return Solution(n) for each action in problem action (n.state) do

n' ← childNode (problem, n, a)

insert (n', g(n') + h(n'), frontier)

Using no. of misplaced tiles as heuristic function

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

$g(n)$ = depth of node

$h(n)$ = no. of misplaced tiles

Initial state

1	2	3
4	5	6
0	7	8

Goal state

1	2	3
4	5	6
7	8	0

$f(0) = 0 + 2 = 2$

1	2	3
0	5	6
4	7	8

$f(1) = 1 + 3 = 4$

1	2	3
4	5	6
7	0	8

$f(1) = 1 + 1 = 2$

1	2	3
4	0	6
7	5	8

$f(2) = 2 + 2 = 4$

1	2	3
4	5	6
7	8	0

$f(2) = 2 + 0 = 2$

Goal state

Using no. of Manhattan distance

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

$g(n)$ = depth of node

$h(n)$ = no Manhattan distance

Initial state

1	2	3
4	5	6
0	7	8

Goal state

1	2	3
4	5	6
7	8	0

$f(0) = 0 + 2 = 2$

$h(0) = 0+0+0+0+0+0+1+1 = 2$

1	2	3
0	5	6
4	7	8

$f(1) = 1 + 3 = 4$

$h(1) = 0+0+0+1+0+0+1+1 = 3$

1	2	3
4	5	6
7	0	8

$f(1) = 1 + 1 = 2$

$h(1) = 0+0+0+0+0+0+0+1 = 1$

1	2	3
4	0	6
7	5	8

$f(2) = 2 + 2 = 4$

$h(2) = 0+0+0+0+1+0+0+1 = 2$

1	2	3
4	5	6
7	8	0

$f(2) = 2 + 0 = 2$

$h(2) = 0+0+0+0+0+0+0+0 = 0$

8/11/2024

LAB-04

Hill-climbing Search Algorithm

function Hill-Climbing (problem) return a state that is local maximum

current ← Make-Node (problem.Initial-State)

loop do

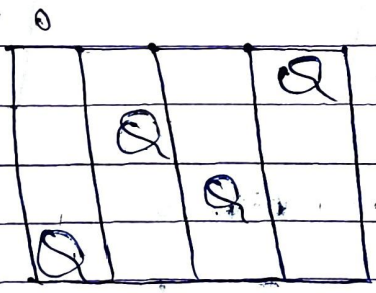
neighbor ← a highest valued successor of current

if neighbour.VALUES ≤ current.Value then return State current

current ← neighbor

Execute in

N-Queen problem



Initial state $x_0=3$, $x_1=1$, $x_2=2$, $x_3=0$ & cost = 2

Neighbor

1] $x_0=1$, $x_1=3$, $x_2=2$, $x_3=0$ Cost = 1

2] $x_0=2$, $x_1=1$, $x_2=3$, $x_3=0$ Cost = 1

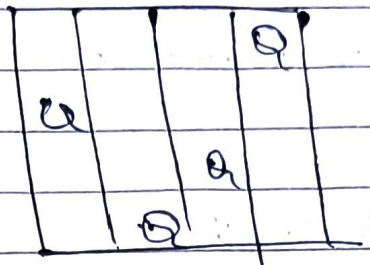
3] $x_0=0$, $x_1=1$, $x_2=2$, $x_3=3$ Cost = 6

4] $x_0=3$, $x_1=2$, $x_2=1$, $x_3=0$ cost = 6

5] $x_0=3$ $x_1=1$ $x_2=0$ $x_3=2$ Cost = 1

6] $x=3$ $x_1=0$ $x_2=2$ $x_3=1$ Cost = 1

Next chromosome

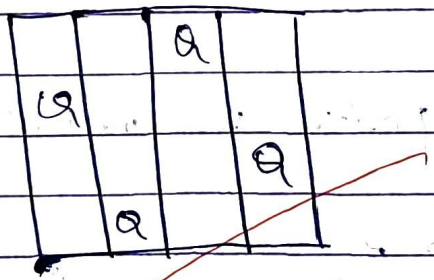


Initial state $x_0 = 1 \quad x_1 = 3 \quad x_2 = 2 \quad x_3 = 0$

Neighbour state

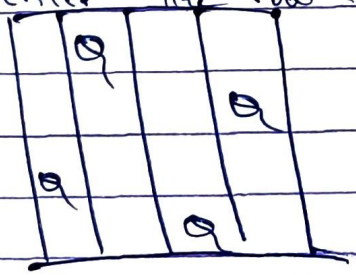
x_0	x_1	x_2	x_3	Cost
3	1	2	0	2
2	3	1	0	2
0	3	2	1	4
1	2	3	0	4
1	0	2	3	2
1	3	0	2	0

Find state



Output

Enter the row position for the queens : 3 1 2 0



15/11/24

LAB-5

N-queen implementation using Stimulated Annealing

Algorithm

```
Current ← Initial state.
  while T>0 do
  next ← a random neighbor of current
  ΔE ← current.cost - next.cost
     if ΔE > 0 then
          Current ← next
     else
        Current ← next with probability P = e^(ΔE/T)
     end if
        decrease T
  end while
  return current
```

Output :-

```
Enter the column position for queen : 2 5 7 6 3 4 1 0
Solution found !
. . . . . Q .
. . . . Q . . .
. . Q . . . . .
Q . . . . . .
. . . . . Q . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
```

LAB-06

Unification :-

Algorithm Unify (Ψ_1, Ψ_2)

Step 1 :- If Ψ_1 or Ψ_2 are variable or constant then

a) if Ψ_1 or Ψ_2 are identical, then return NIL.

b) else if Ψ_1 is a variable,

i] then if Ψ_1 occurs in Ψ_2 then return Failure

ii] else return $\{(\Psi_1 / \Psi_2)\}$

c) else if Ψ_2 is a variable,

i] If Ψ_2 occur in Ψ_1, then return Failure ii] else return $\{(\Psi_2 / \Psi_1)\}$

d] else return failure

Step 2 :- If the initial ~~predict~~ predicate symbol in Ψ_1 & Ψ_2 are not same, the return failure

Step 3 :- If Ψ_1 and Ψ_2 have different number of argument 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 i^{th} element of Ψ_1 & i^{th} element of Ψ_2 & put the result into S

b] If S = failure then return failure

d] If S ≠ NIL then do

a] Apply S to the remainder of both L_1 & L_2

b] SUBST = APPEND (S, SUBST)

Step 6 - Return SUBST

Output

Enters two term to unify (eg:- f(x,y) ; f(a,b)):

Enter first term : f(x, apple)

Enter first term : f(eats, y)

Unifying terms : ('f', 'x', 'apple') & ('f', 'eats', 'y')

Unification Successful !

Substitution {'x': 'eats', 'y': 'apple'}

Unified Expression :

Term 1 after substitution : ('f', 'eats', 'apple')

Term 2 after substitution : ('f', 'eats', 'apple')

22/11/24

LAB-07

Forward Reasoning Algorithm

Algorithm

function FOL-FC-ASK (KB, α) returns a substitution or false
  input: KB, the knowledge base, a set of first-order definite
    α, the query, an atomic sentence
  local variables: new, the new sentence inferred on each iteration

repeat.
repeat until new is empty
  new $\leftarrow$ { }
  for each rule in KB do
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES (rule)
    for each θ such that SUBST ($\theta, p_1 \wedge \ldots \wedge p_n$)
        = SUBST ($\theta, p_1' \wedge \ldots \wedge p_n'$)
      $q' \leftarrow$ 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

2 d 11/24

Input
Initial
It is raining
The ground is wet

Final fact deduced:
It is raining
The ground is wet
People might slip

a] Emily is either a surgeon or lawyer:

Occupation (Emily, Surgeon) ∨ Occupation (Emily, lawyer)

b] Joe is an actor, but also holds another job:

Occupation (Joe, Actor) ∧ (∃ ≠ Actor Occupation (Joe,))

c] All surgeon are doctors:

∀p (Occupation (p, surgeon) → Occupation (p, Doctor))

d] Joe doesn't have a lawyer

∀p (Customer (Joe, p) → ¬ Occupation (p, lawyer))

e] Emily has boss who is lawyer

$\exists b\ (Boss(b, Emily) \wedge Occupation(b, lawyer)$

f] There exist a lawyer all of whose customer are doctors

$\exists p (Occupations(p, lawyer) \wedge \forall c\ (Customer(c, p) \Rightarrow Occupation(c, Doctor))$

g] Every surgeon has a lawyer

$\forall p\ (Occupation(p, surgeon) \rightarrow \exists c\ (Customer(p, c) \wedge Occupation(c, lawyer))$

24/11/24

LAB-08

Alpha - Beta pruning

```
Function alpha-beta pruning (node, depth, alpha, Beta)
    maximizing (player):
  If depth == 0 or node is terminal:
    Reborn Evaluate (node)
  If maximizing_player:
      max_eval = -Infinity
      for each child of node:
          eval = alpha_beta_Pruning (child, depth-1,
                                     alpha, Beta, false)
      max_eval = max (max_eval, eval)
      alpha = max (alpha, eval)
      If beta <= alpha:
          Break
    return max_eval
Else:
    min_eval = Infinity
    For each child node:
        eval = alpha-beta. pruning (child, depth-1, alpha, Beta, true
        min_eval = min (min_eval, eval)
        beta = min (beta, eval)
        If beta <= alpha:
            Break
    return min_eval
```

Output:

For tree = [[3,5,6],[9,1,2],[0,7,4]]

Optimal value : 6

(AB -)14

Converting FOL into CNF

Input first order logic statement
Eliminate implication : Replace (A → B) using (¬A ∨ B)
Move ¬ (negation) inwards using De morgan's laws
~~Standardize variables to the front (P form)~~
Standardize variable : Ensure each quantifiers has unique variable
Move quantifiers to the front (Prenex form)
Skolemize : Eliminate existential quantifiers by introducing Skolem function
Drop universal quantifiers.
Distribute ∨ over ∧ to obtain CNF form
Output CNF

Output :-
Original statement :- (A ∧ B) → C
CNF form :- ¬A ∨ ¬B ∨ C

20/12/24

LAB-10

Creating a knowledge Base using propositional logic and Proving query using resolution

Initalize knowledge-base with propositional logic statement
Input query

Convert knowledge-base and query into CNF
Add query to CNF-clause

While True:
    select two clauses from CNF-clause
    Resolve clauses to produce a new clause
    If new clause is empty:
        Print "Query is proven using resolution"
        Break
    If new clause is not already in CNF-clause
        Add new clause to CNF-clause
    If no new clause can be generated:
        Print "query cannot be proven using resolution"
        Break

Output:
For knowledge-base ("A", "B", "A∧B=>C", "C=>D")
query="D"

Query is Proven using resolution

LAB - 11

Knowledge Base using propositional logic

Initalize knowledge base with propositional logic statement
Input query
If ~~forduo~~ forword-chaining (knowledge_base, query):
    Print "Query is entailed by the knowledge - Base"
Else:
    Print "Query is not entailed by knowledge Base"

Function forword-chaining (knowledge-base, query):
    Initalize agenda with known facts from knowledge base
    while agenda is not empty:
        Pop a fact from agenda
        If fact matches query
            Return True
        For each rule in knowledge_base
            If fact satisfy a rule's premise:
                Add the rule's conclusion to agenda
    Return False

Output:
20/12/24
For the knowledge_Base = ["A", "B", "A&B => C", "C => D"]
Query = "D"

Query is entailed by the knowledge Base