INDEX

Name : YASHRAJ SINHA Class : .....

Section : ..... Roll No. : 1BM22CS335 Subject : Artificial Intelligence Lab

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
1.	4/10	Implementing tic-tac-toe algorithm using python.	1	10
2.	18/10	Implementing ~~tic-tac~~ Vacuum cleaner using python.	2-3	10
3.	13/10	Implementing 8-puzzle problem using python.	4	10
4.	25/10	A* Algorithm	5-67	10
5.	8/11	Implementing Hill Climbing Algorithm to solve N-Queens problem.	9-11	10
6.	15/11	Implement Simulated Annealing to solve N-Queens problem	12-	10
7.	22/11	Implement Unification in First Order logic.	13-14	10 22/11/24
8.	29/11	Create a knowledge base consisting of first order logic statements and prove a given query	15-17	10
9.	19/12	Create a KB using propositional logic and prove the query using resolution, Convert	18-23	10

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
		FOL to CNF, Alpha-beta pruning, Knowledge base using propositional logic / entails	10	2012

4/10/24

LAB-1

Date
Page 1

① Implementing tic-tac-toe algorithm using python:

→ if isMaximizingPlayer:
    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

4/10/24

18/10/24

LAB - 2

2) Implementing vacuum cleaner using python

```
FUNCTION vacuum_world()
    INITIALISE goal_state as ['A':'0','B':'0']
        // 0: Clean, 1: Dirty
    SET cost to 0
    PROMPT for vacuum location (A/B) as location_input
    PROMPT for status of location_input as status_input
    SET other_location to 'B' if 'A' else 'A'
    PROMPT for status of other_location as status_input_
    compliment

    IF location_input is 'A':
        IF status_input is '1':
            SET goal_state['A'] to '0'
            Increment cost by 2
            IF status_input_compliment is '1':
                SET goal_state['B'] to '0'
        ELSE IF status_input_compliment is '1'
            Increment cost by 1
            SET goal_state['B'] to '0'
    ELSE:
        IF status_input is '1':
            SET goal_state['B'] to '0'
            Increment cost by 2
            IF status_input_compliment is '1':
                SET goal_state['A'] to '0'
        ELSE IF status_input_compliment is '1':
            Increment cost by 1
            SET goal_state['A'] to '0'
    Print goal_state and cost
END FUNCTION
CALL vacuum_world().
```

Algo

1) Initialize the agent's starting (x,y)

2) Loop until all cells are clear:
   a. Perceive the current cell
   b. If the ~~cell~~ cell is dirty
      i. Clean the current cell
   c. else
      i. Check surrounding ~~cell~~ cells (up, down, left, right), to see if any are dirty.
      ii. Move to the next dirty cells applying strategy such as BFS, DFS or other algorithm.
   d. If no dirty cell are perceived, stop.

3) End.

18/10/2024

3) Implementing 8-puzzle problem using python

BFS

Let fringe be a list containing the initial state.
LOOP
if fringe is empty return failure
Node ← remove-first (fringe)
if Node is goal
then return the path from initial state to node and add generated nodes to the head of fringe
else generate all successors of Node and add generated node to the back of fringe.
END LOOP.

DFS

Let fringe be a list containing the initial state.
LOOP
if fringe is empty return failure
Node ← remove-first (fringe)
if Node is goal
then return the path from initial state to Node
else generate all successors of Node and add generated node to the front fringe.
END LOOP.

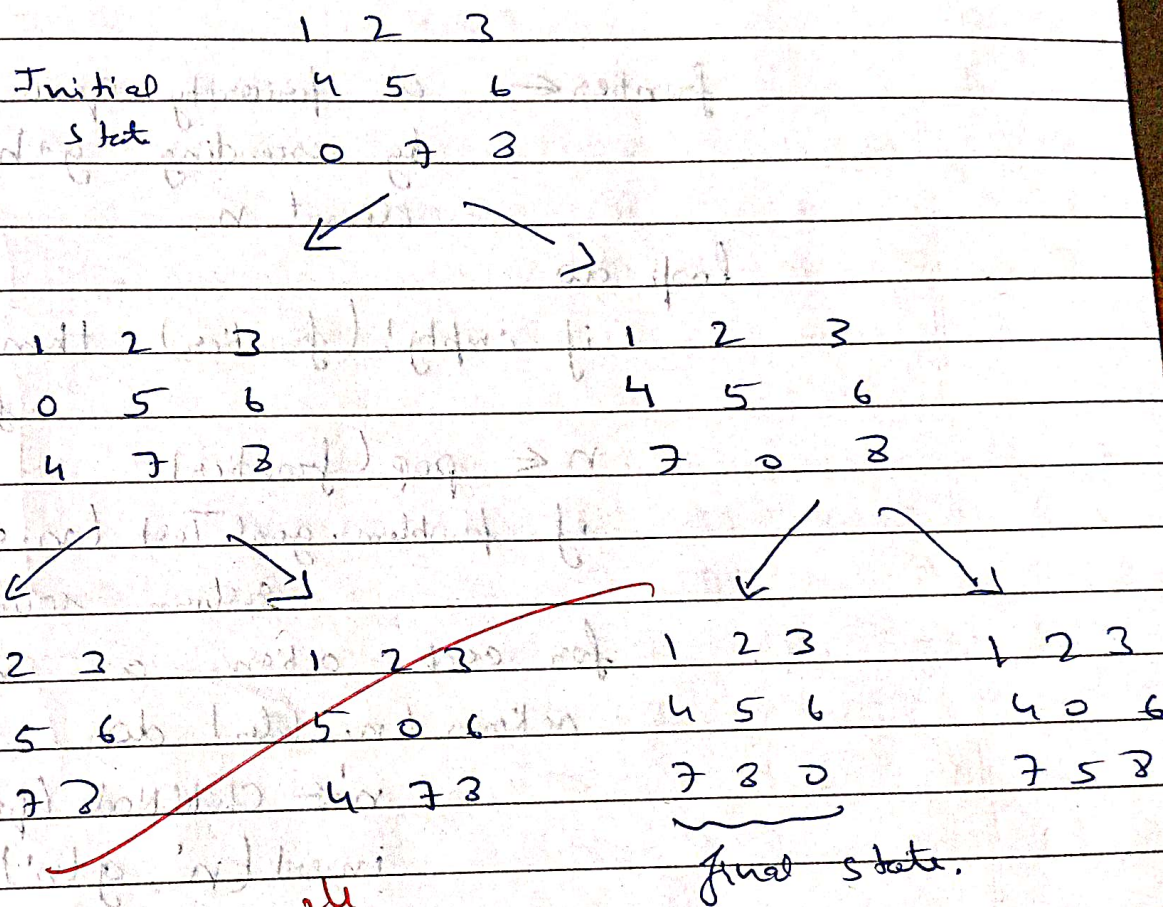
State Space tree : {using BFS}

Initial

1 2 3
4 5 6
0 7 8

Final

1 2 3
4 5 6
7 8 0



8/10/2024

25/10/24 LAB-3

4) Implementing A* algorithm using python.

→ A*

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

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

frontier ← a priority queue ordered by ascending g+h, only element n.

loop do

if empty? (frontier) then return failure

n ← pop (frontier)

if problem.goalTest (n.state) then return solution(n)

for each action a in problem. actions (n.states) do

n' ← ChildNode (problem, n, a)

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

(i) Using no. of misplaced tiles as heuristic function

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

1	2	3
4	5	6
0	7	8

Initial State

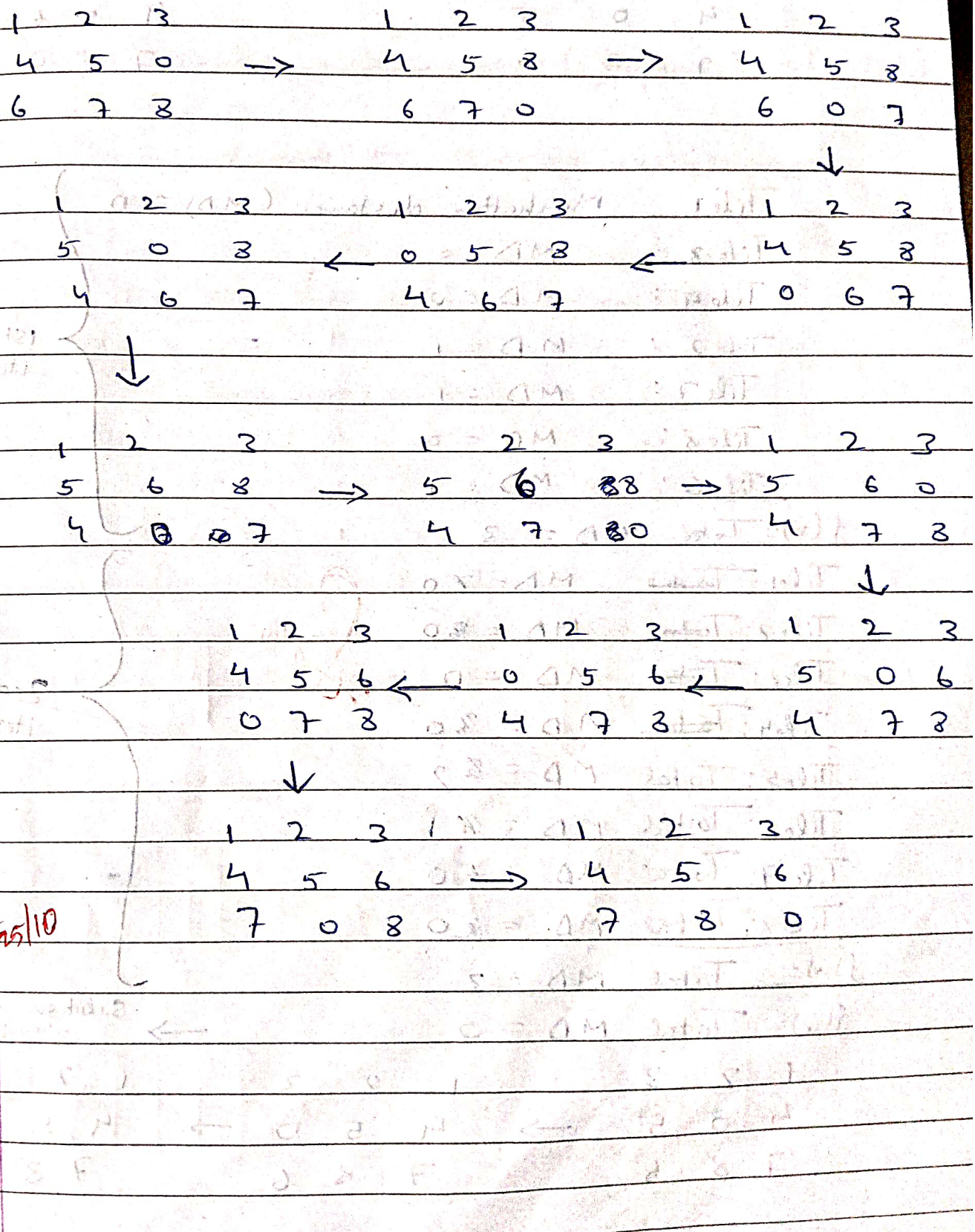
1	2	3
4	5	6
7	8	0

Goal state

Output

1	2	3
4	0	5
6	7	8

Initial state



25/10

ii) Using Manhattan distance

→ Initial state

1	2	3
4	0	5
7	8	6

Goal state

1	2	3
4	5	6
7	8	0

Tile 1 : Manhattan distance (MD) = 0
Tile 8 : MD = 0
Tile 4 : MD = 0
Tile 2 : MD = 1
Tile 7 : MD = 1
Tile 8 : MD = 0
Tile 6 : MD = 1
f(n) = Total MD = 3

1st iteration

Tile 1 : ~~Total~~ MD = ~~3~~ 0
Tile 2 : Total MD = ~~3~~ 0
Tile 3 : ~~Total~~ MD = 0
Tile 4 : ~~Total~~ MD = ~~3~~ 0
Tile 5 : Total MD = ~~3~~ 2
Tile 6 : Total MD = ~~3~~ 1
Tile 7 : Total MD = ~~3~~ 0
Tile 8 : Total MD = ~~3~~ 0
f(n) = Total MD = 3
f(n) = Total MD = 0

2nd iteration

→ Solution in 3rd iteration

1	2	3		1	2	3		1	2	3
4	0	5	→	4	5	0	→	4	5	6
7	8	6		7	8	6		7	8	0

3/11/24

LAB - 04

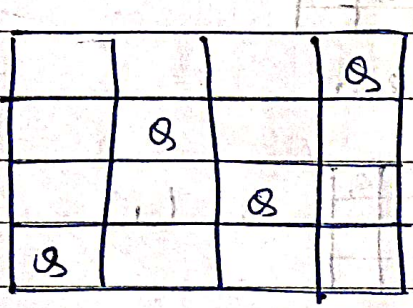
5) Implementing Hill Climbing search algorithm to solve N-Queens problem.

→ function Hill_Climbing (problem) returns a state that is local maximum.

```
current ← Make-Node (problem.Initial-State)
loop do
    neighbour ← a highest value successor of current
    if neighbour.value ≤ current.Value
        then return current.State
    current ← neighbour.
```

Execute

6) Show how the cost calculation of current state and neighbour nodes. And continue until you reach goal configuration of 4-Queens board.



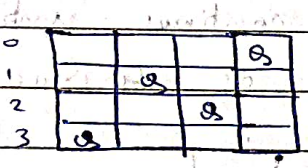
State	h(score)
3,1,2,0	2
~~1,2,3,0~~ 1,3,2,0	1
2,1,3,0	1
0,1,2,3	6
~~1,2,1,0,3~~	
~~2,0,1,3~~	

$4C_2 = 6$

6 swaps

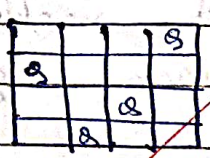
State	h(score)	
3,1,2,0	2	
1,3,2,0	1	← this
2,1,3,0	1	
0,1,2,3	6	
3,2,1,0	6	
3,0,2,1	1	
3,1,0,2	1	

State space :



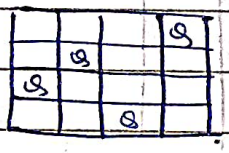
h = (2)

3,1,2,0



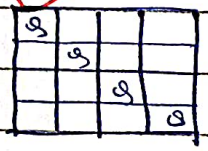
h = (1)

1,3,2,0



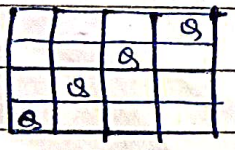
h = (1)

2,1,3,0



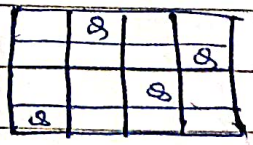
h = (6)

0,1,2,3



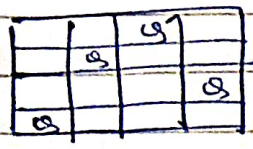
h = (6)

3,2,1,0



h = (1)

3,0,2,1

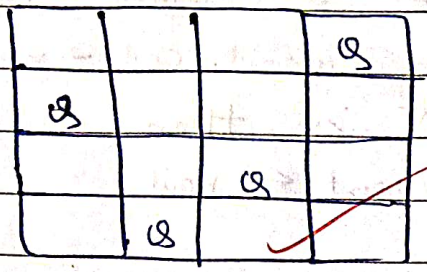


h = (1)

3, 1, 0, 2

- Next step

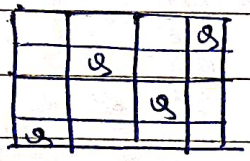
1, 3, 2, 0



h = (1)

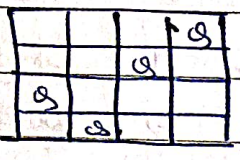
state	h (score)
1, 3, 2, 0	1
3, 1, 2, 0	2
2, 3, 1, 0	2
0, 3, 2, 1	3
1, 2, 3, 0	3
1, 3, 0, 2	0 ← return solution.

state space :



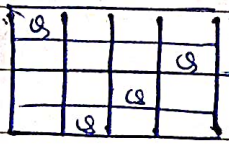
h = (2)

3, 1, 2, 0



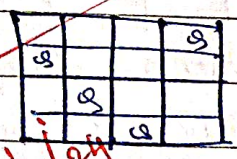
h = (2)

2, 3, 1, 0



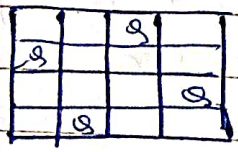
h = (3)

0, 3, 2, 1



h = (3)

1, 2, 3, 0



h = (0)

1, 3, 0, 2

15/11/24

LAB - 5

⑥ Implement Simulated Annealing to solve N-Queens problem.

⇒

```
current ← initial state
T ← a large positive value
while T > 0 do
    next ← a random neighbour 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

```
Iteration 1 : Conflicts = 24
Iteration 2 : Conflicts = 26
      :
      :
Iteration 4999 : Conflicts = 0
Iteration 5000 : Conflicts = 0

Final solution : [3, 6, 2, 7, 1, 4, 0, 5]
Number of conflicts in final solution : 0
```

15/11/24

22/11/24 LAB-6

(7) Implement Unification in First Order Logic

→ i) If Ψ_1 and Ψ_2 is a variable or constant then :

(a) If Ψ_1 and Ψ_2 are identical, then return NIL.

(b) Else if Ψ_1 is a Variable,
- then if $\Psi_{2,1}$ occurs in Ψ_2 then return failure
- Else return $\{(\Psi_2/\Psi_1)\}$.

(c) Else if Ψ_2 is a variable,
- If Ψ_2 occurs in Ψ_1 then return failure
- Else return $\{(\Psi_1/\Psi_2)\}$.

(d) Else return failure.

ii) If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return failure

iii) If Ψ_1 and Ψ_2 have a different number of ~~eq~~ arguments, then return failure.

iv) Set substitution set (SUBST) to NIL.

v) 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 and

put the result into S.

(b) If S = failure then return Failure.

(c) If S ≠ NIL then do.
- Apply S to the remainder of both L_1 and L_2.
- SUBST = APPEND(S, SUBST).

vi) Return SUBST.

Execute

Output

Expression 1 : ("Eats", "x", "Mango", "Pizza")
Expression 2 : ("Eats", "Sumit", "y", "z")

Substitutions : {'x' : 'Sumit', 'y' : 'Mango', 'z' : 'Pizza'}

Ans.

29/11/24 LAB-7

8) Create a knowledge base consisting of first order logic statements and 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 iteration.

repeat until new is empty
new ← { }
for each rule ~~is empty~~ in KB do
(p1 ^ ... ^ pn => q) ← STANDARDIZE-VARIABLES(rule)
for each θ such that SUBST(θ, p1 ^ ... ^ pn) = SUBST(θ, p1' ^ ... ^ pn')
for some p1', ..., pn' in KB
q' ← SUBST(θ, q)
if q' does not unify with some sentence already in KB or new then
add q' to new
θ ← unify(q', α)
if φ is not fail then return
add new to KB
return false

Output

Criminal (Robert) is proven!
Inferred facts:
Missile (T_1)
Weapon (T_1)
Hostile (A)
Owns (A, T_1)
Criminal (Robert)
Enemy (A, America)
American (Robert)
Sells (Robert, T_1, A)

29/11/24

~~8) Consider a vocabulary with the following symbols~~

#

(a) Occupation (Emily, Surgeon) ∨ Occupation (Emily, Lo

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

(c) ∀p (Occupation (p, Surgeon) → Customer (Joe, p))

(d) ∃p (Occupation (p, Lawyer) ∧ Customer (Joe, p))

(e) $\exists p\ (Boss(p, Emily) \wedge Occupation(p, Lawyer))$

(f) $\exists p\ (Occupation(p, Lawyer) \wedge \forall c\ (Customer(c, p) \rightarrow Occupation(c, Doctor)))$

(g) $\forall p\ (Occupation(p, Surgeon) \rightarrow \exists l\ (Occupation(l, Lawyer) \wedge Customer(p, l)))$

29/11/24

19/12/24

9) Create a knowledge base using Propositional logic and prove the given query using resolution.

Function: resolution (KB, query): return query if true or false

input: KB, the Knowledge base, set of propositions

query: a ~~s~~ state to be proven

clauses = convert to CNF (KB, negated query = negate (query))

new_clause = set()

apply the resolution rule:

- select the clauses contain complimentary clauses
- resolve the two to form a new clause
- add the new_clause is empty {} contradiction is found.

If new_clause == {} print "query if true" else print "~~if~~ to false"

Output:

KB: P ∨ Q

To prove: P is true

¬Q ∨ R ✓

¬P ∨ Q ✓

R ∨ ¬P

Resolution ¬R ✓

(¬Q ∨ R), (¬R) → ¬Q

(¬P ∨ Q), (¬Q) → ¬P

$(R \vee P), (\neg P) \rightarrow R$

contradiction / ~~is~~ query is true.

10) Convert a given FOL to CNF

→ Function FOL-to-CNF :

Replace $(P \leftrightarrow Q)$ with $(P \Rightarrow Q) \wedge (Q \rightarrow P)$

Replace $(P \rightarrow Q)$ with $(\neg P \vee Q)$

Replace $\neg(A \vee Q)$ with $\neg P \wedge \neg Q$

Replace $\neg(P \wedge Q)$ with $\neg P \vee \neg Q$

Replace $\neg\neg P$ with P

Rename all variable to make them unique.

Replace $\exists x[p(x)]$ with $P(c)$ if x is ~~an~~ independent

Replace $\exists x[p(x)]$ with ~~$P(y)$~~ $P(f(y))$ if x depends on a universally quantified variable.

~~Remo~~ Distribute $\vee$ over $\wedge$:

Ensure all formulas is a conjuction of disjunction

Output

$\forall x (\exists y (f(x,y) \rightarrow Q(y)) \wedge \neg R(x)$

$y \neq f(x)$ $y = f(x)$

$\forall x ((\neg p(x, f(x)) \vee Q(y))) \wedge \neg R(x)$

$\Rightarrow (\neg p(x, f(x)) \vee (f(x)) \wedge \neg R(x)$

Q.1) Implement alpha-beta pruning

-> alpha : the best option for the maximizer
Beta : the best option for the minimizer
pruning : stop exploring sub-tree when it is clear that they won't affect the out-come.

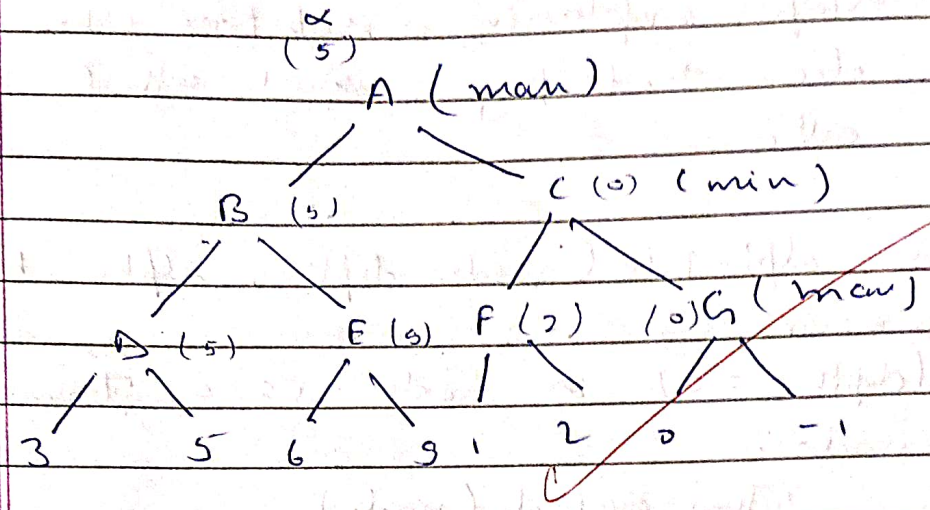
```
Function alphabeta ( node, depth, alpha, beta,
    isMaximizing
  if (depth == 0) or node is a terminal
    node :
      return evaluate (node)
  if is_maximizing () :
      maxeval = -∞
    for each child in children (node) :
      eval = alphabeta ( child, depth-1, alpha,
          beta, false)
      maxeval = max ( maxeval, eval)
      alpha = max ( alpha, eval)
        if alpha >= beta :
            break (prune)
      return maxval.

  else :
      minval = +∞
      for each child in children (maybe beta) :
      eval = alphabeta ( child, depth-1, alpha,
                beta, eval)
      beta = min (beta, eval)
      if alpha >= beta :
```

break (prune)

return min val

Output



12) Knowledge - base using proposition logic | Entails

→ Initialize KB with proposition logic statement

If forward - chaining (KB, query) :
    print "Query entailed by Knowledge base"
else
    print "Query if not entailed by KB"

Initialize Forward chaining (KB, query)
    Initialize agenda with known facts from KB.
    While agenda is not empty
        POP fact from agenda.
        If fact matches query :
            return true
        For each rule in KB
            If fact satisfies rule premise
                add rules to agenda.
    return false.

Output

20/12/24

For the KB = [A, B, A ∧ B => C, C => D]
query D.

Query established by Knowledge Base.