Date __/__/__
Page _____

Lab: vaccum world cleaner

function REFLIX-VACCUM-AGENT ([location, status]) return an action
if status = Dirty then return suck
else if location = A then return right
else if location = B then return left

Algorithm for two quadrants,

step 1 : Initialization.

- Input current room (either A or B)
- Input the status of the room clean or dirty.
- Initialize a cost to 0.

Step 2: Display initial room status

Step 3: Cleaning loop

- While either of the room is dirty.
  1) If current room is A and is dirty then clean and increase cost by one
  2) If current room is B and is dirty then clean and increase cost by one.

step 4: Display cost
step 5: stop.

Algorithm for four quadrants.

Step 1: start.

- Accept current room input from user (either A, B, C or D).
- Step 2: Display the initial status.
- Step 3: Clean rooms.

While all rooms are clean.

1) If first room is dirty clean and go to next room
2) If second room is dirty clean and go to next room
3) If third room is dirty clean and go to next room
4) If fourth room is dirty clean and go to next.
5) Increase cost when cleany & decrease count.

Step 4: Show the last status.
Step 5: End.

Date __/__/__
Page

Lab: 8 Puzzle Problem - BFS & DFS

Algorithm for BFS:

1) Initialization:

- Create a node data structure to represent the current state of the puzzle
- Initialize the node with the initial state of the puzzle
- Create a queue to store node to be explored.
- Enqueue the initial node.

2) Loop untill goal state is reached or queue is empty.

3) Return solution path.

Algorithm for dfs:

1) Initialization:

- Ititicalize the node with the initialize of puzzle
- Push the initial node onto the stack

2) Loop until goal state is reached or stack is empty.

Date __/__/__
Page

3) Backtrakk: if the stack is empty and the goal state is not reached, return failure.

Output.

Enter the initial state (space-separated):

1 2 3 0 4 6 7 5 8

Move 0:	Move 1:	Move 2:	Move 4:
1 | 2 | 3	1 | 2 | 3	1 | 2 | 3	1 | 2 | 3
0 | 4 | 6	4 | 0 | 6	4 | 5 | 6	4 | 5 | 6
7 | 5 | 8	7 | 5 | 8	7 | 0 | 8	7 | 8 | 0

No. of moves = 3.

Date __/__/__
Page _____

Lab: For 8 puzzle problem using A* implementation to calculate f(n) using
(a) g(n) = depth of node
h(n) = heuristic value → no. of misplaced tiles
f(n) = g(n) + h(n)
(b) g(n) ⇒ depth of node
h(n) ⇒ heuristic value ⇒ manhatten distance
f(n) = g(n) + h(n)

Algorithm for No. of misplaced Tiles:

Step 1: Initialize the initial state and goal state

Step 2: Based on the position of '-' tile calculated possible movements and calculate g(n) depth of puzzle and h(n):

Step 3: Calculate f(n) = h(n) + g(n) for all possible tiles with minimum f(n).

Step 4: When f(n) = 0, the goal is achieved

Algorithm for manhatten distance →

Date __/__/__
Page

Step 1: Initialize the initial stage and goal state.

Step 2: loop until not become a i.e. goal state is achieved.

Step 3: Perform $f(n) = g[n] + h(n)$ and move forward by selecting minimum value.

Step 4: Once the goal is achieved print $g(n)$ and No. of movements.

Date __/__/__
Page _____

{Iterative DFS-graph}

Lab 3: Iterative deepening search Algo

Algorithm:

1) For each child of the current node
2) If it is the target node, return
3) If current maximum depth is reached, return
4) Set the current node to this node and 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 and go back to 1.
7) If we have reached all leaf (bottom) nodes, the goal node doesn't exist.

function ITERATIVE-DEEPENING-SEARCH(problem) return a solution, or failure

Date __/__/__
Page _____

for depth = 0 to ∞ do

result ← Depth-limited-search
(problem, depth,
if result ≠ cutoff then
return result)

Page

Lab: Hill climbing Algo for N-Queens

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 neighbor.VALUE ≤ current VALUE then return current.STATE
    current ← neighbor

- State: 4 queens on the board.
  - variables: x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume there is one queen per column
  - domain for each variable x: $\in \{0,1,2,3\}, \forall i$.
- Initial state: a random state
- Goal state: 4 queens on the board. No pair of queens attacking each other.

Date __/__/__
Page _____

- Neighbour relation : Swap the row position of two queens.
- Cost function : The no. of pairs of queens attacks on each other, directly and indirectly.

Date __/__/__
Page

Lab: Annealing Algo {n-queens}

function simulated-Annealing (problem, schedule) return a solution state
inputs: problem, a problem
schedule a mapping from time to temperature
current ← Make-Node (problem, Initiate-state).
from t ← 1 to ∞ do
T ← schedule (t)
if T=0 then return current
if ΔE > 0 then current ← next
else current ← next only with probability $e^{\Delta E/T}$

OR

1. Start a random point x
2. Choose a new point x_j on a neighbourhood N(n)
3. Decide whether or not to move to the new point x_j The decision will be made based on probability function P(n, x_j, T).

$$P(x, x_j, T) = \begin{cases} 1 \text{ if } f(x_j) \geq f(n) \\ e^{\frac{f(x_j) - f(n)}{T}} \text{ si } f(x_j) < f(n) \end{cases}$$

4. Reduce T

Output :

1) for 8 queen problem -
The best position found is
[6 3 1 7 5 0 2 4].

Date __/__/__
Page _____

Lab: Propositional Entailment

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

symbols ← a list of propositional symbols in KB and 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 do,

P ← FIRST (symbols)
rest ← REST (symbols)

Date __/__/__
Page _____

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

and

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

Date __/__/__
Page

Lab: Unification in FOL

Algorithm: Unify (Ψ_1, Ψ_2)

Step 1: If Ψ_1 or Ψ_2 is variable or constant, then:
- (a) If Ψ_1 or Ψ_2 are identical, then return nil.
- (b) Else if Ψ_1 is variable then if Ψ_1 occurs in Ψ_2 then return failure
- (c) Else if Ψ_2 is variable If Ψ_2 occurs in Ψ_1 then return failure, else return (Ψ_1/Ψ_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: Set substitution set (SUBST) to NIL

Step 4: For i = 1 to the number of elements in Ψ_1,
- (a) Call unify function with i^{th} element of Ψ_1 and i^{th} element of Ψ_2, and put result into S.
- (b) If S ≠ NIL then do, Apply S to the remainder

of both.

(b) SUBST = APPEND (S, SUBST)

Step 6: Return SUBST

eg: $P(x, F(y)) \quad -①$

$P(a, F(g(x))) \quad -②$

in ① & ② predicate are identical and no. of argument are equal.

in ① replace x with a

$P(a, F(y))^{a/x} \quad -①$

in ① ~~and ② are same~~ replace y with g(x).

$P(a, F(g(x))^{g(x)/y} \quad -①$

Now ① and ② are same.

Date __/__/__
Page _____

Lab: Create KB consisting of FOL statement & prove the query

→ {FOL - Forward + Reagean}

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 automic sentense.
local variables: new, the new sentence infored 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 subset (θ, p1 ^ .. ^ pn) = SUBSET(θ, p'1 ^ .. ^ p'n) for some p'1 ... p'n in KB

q' ← SUBST(θ, q)

if q' does not unify with some sentences 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

Date __/__/____
Page _____

Lab: FOL statement to CNF: (Resolution)

Basic steps for proving a conclusion S given premises Premise 1, ..., Premise n.

(all expressed in FOL):

1) Convert all sentences to CNF
2) Negate conclusion S and convert result in CNF.
3) Add negated conclusion S to the premise clauses.
4) Repeat until contradiction or no progress is made.
   (a) Select two clauses
   (b) Resolve them together, performing all required unifications.
   (c) If resolvent is the empty clause, a contradiction has been found (ie, S follows from the premise.
   (d) If not, add resolvent to the premises.

Date __/__/__
Page _____

Lab: Alpha Beta Pruning

```
function ALPHA-BETA-SEARCH (state)
        return an action
    v ← MAX-VALUE (state, -∞, +∞)
return the action in ACTIONS
    (state) with value v

function MAX-VALUE (state, α, β)
        returns a utility value
if TERMINAL-TEST (state) then return
    UTILITY (state)
v ← -∞
for each a in ACTIONS (state) do
    v ← MAX(v, MIN-VALUE (RESULT (s,a), α, β))
    if v ≥ β then return v
        α ← MAX (α, v)
    return v

function MIN-VALUE (state, α, β)
return a utility value if
TERMINAL-TEST (state) then
return UTILITY (state)
v ← ∞
for each a in ACTIONS (state) do
    v ← MIN(v, MAX-VALUE (RESULT (s,a), α, β))
    if v ≤ α then return v
        β ← MIN (β, v)
    return v
```

Date __/__/__
Page _____

Max α 6

Min β 3 6 5

Max α 5 3 6 7 5 8

Min β 5 4 3 6 6 7 5 8 6

5 6 7 4 5 3 6 6 9 7 5 9 8 6

Output:

Enter the leaf node value separated by space :-
-1 8 -3 -1 2 1 -3 4
Optimal value calculated using Minimax : 2