Page No.

Date

1. Write a LISP program to solve the water-jug problem using heuristic function.

```
CL-USER 1 > (start)

WATER JUG PROBLEM
FILL JUGNAME : (fill JUGNAME)
EMPTY JUGNAME : (empty JUGNAME)
PUT A TO B : (put A to B)

GIVE THE STARTING STATE OF EACH JUG IN
FROM (A B C) E.G (0 0 0)
MAX A=7, MAX B=4 AND MAX C=2 : (0 0 0)
NIL

CL-USER 2 > (fill B)

NEW STATE : (0 4 0)
NIL

CL-USER 3 > (put B to A)

NEW STATE : (4 0 0)
NIL

CL-USER 4 > (fill B)
```

Page No. .

Date . .

```
NEW STATE : (4 4 0)
NIL
CL USER 5> (put B to A)
NEW STATE
CL-USER : (7 1 0)
YOU WIN B=1
NIL

CL-USER 6>
```

Delight

Page No.

Date

2. Create a compound object using Turbo Prolog.

```
domains
    things = car (brand); flat; villa; networth (integer)
    name, brand = symbol
predicates
    owns (name, things)
clauses
        owns (jeff, car (bmw)).
        owns (jeff, villa).
        owns ('jeff, networth (43)).
        owns ('bill, car (audi)).
        owns (bill, flat).
        owns (bill, networth (40)).
```

Page No.

Date

3. Write a Prolog Program to show the advantage and disadvantage of green & red cuts.

```
Domains:
disease, indication = symbol
name-string

Predicates:
hypothesis (name, disease)
symptom (name, indication)
response (char)
go
goonce
Clauses:
go:-
goonce
write ("will you like to try again (y/n)?"),
response (Reply),
Reply = 'n'.
go.
goonce:-
write ("what is the patient's name"), nl,
readln (Patient),
hypothesis (Patient, Disease), !,
write (Patient, "probably has", Disease), !,
goonce:-
```

Page No.

Date

```
write ("sorry, i am not in a position to diagnose"),
write ("the disease").
symptom (Patient, fever):-
write ("does", Patient, "has a fever (y/n)?"), nl,
response (Reply),
Reply = 'y', nl.
symptom ( Patient, rash):-

write ("does", Patient, "has a rash (y/n) ?"),
nl, response (Reply),
Reply = 'y'
symptom (Patient_body, ache):-
write ("does", Patient, "has a body ache (y/n)?"),
nl, response (Reply).
Reply = 'y', nl.
symptom (Patient, runny-nose):-
write("does", Patient, " has a runny_nose (y/n)?"),
response (Reply),
Reply = 'y'
hypothesis (Patient, flu):-
symptom ( Patient, fever),
symptom ( Patient, body-ache),
hypothesis ( Patient, common-cold):-
symptom (Patient, body-ache),
Symptom (Patient, runny_nose).
response (Reply):-
readchar (Reply),
write (Reply).
```

Delight

Page No.

Date

4. Write a prolog program to use BEST-FIRST SEARCH applied to the eight puzzle problem.

```
test(Plan):-
write('Initial state:'), nl,
Init = [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4),
 at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8),
 at(tile7,9)],
write_so(Init),
Goal = [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4),
 at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8),
 at(tile8,9)],
nl, write('Goal state:') nl,
write(Goal), nl, nl,
solve(Init, Goal, Plan).

solve(State, Goal, Plan):-
    solve(State, Goal, [], Plan).

is_movable(X1, Y1):- (1 is X1-Y1); (-1 is X1-Y1);
 (3 is X1-Y1); (-3 is X1-Y1).

solve(State, Goal, Plan, Plan):-
    is_subset(Goal, State), nl,
    write_sol(Plan).
```

Page No.

Date

```
solve(State, Goal, Sofar, Plan):-
    act(Action, Preconditions, Delete, Add),
    is_subset(Preconditions, State),
    \+ member(Action, Sofar),
    delete_list(Delete, State, Remainder),
    append(Add, Remainder, NewState),
    solve(NewState, Goal, [Action|Sofar], Plan).

act(move X,Y,Z),
    [at(X,Y), at(empty,Z), is_movable(Y,Z)],
    [at(X,Y), at(empty,Z)],
    [at(X,Y), at(empty,Y)]).

is_subset([H|T], Set):-
    member(H,Set),
    is_subset(T,Set).
is_subset([],_).

delete_list([H|T], Curstate, Newstate):-
    remove(H, Curstate, Remainder),
    delete_list(T, Remainder, Newstate).
delete_list([], Curstate, Curstate).

remove(X, [X|T], T).
remove(X, [H|T], [H|R]):-
    remove(X, T, R).
```

Page No.
Date

```
write_sol([]).
write_sol([H|T]):-
    write_sol(T),
    write(H), nl.

append([H|T], L1, [H|L2]):-
    append(T, L1, L2).
append([], L, L).

member(X, [X|_]).
member(X, [_|T]):-
    member(X, T).
```

Page No.

Date

5. Implementation of the problem solving strategies:- Forward chaining, Backward Chaining, Problem Reduction.

(i) Forward chaining:-

```
:- op(1100, xfx, if).
:- op(1000, xfy, and).
:- op(900, xfy, or).
:- dynamic rule/1.

forward(Facts):-
    fixed_point (nil, [true], Facts).

fixed_point(Base, Base, Base):-!.
fixed_point(_, Base, Facts):-
    setof(Fact, derived(Fact, Base), NewFacts),
    ord_union(NewFacts, Base, NewBase),
    fixed_point(Base, NewBase, Facts).

derived(Fact, Base):-
    rule(Fact if Condition),
    satisfy(Base, Condition).

satisfy(Base, G and Gs):-
    !,
    member(G, Base),
    satisfy(Base, Gs).
```

Page No.

Date

```
satisfy (Base, G or Gs) :-
    !,
    ( member (G, Base)
    ; satisfy (Base, Gs)).

satisfy (Base, Condition) :-
    member (Condition, Base).
```

(ii) Backward chaining :-

```
is_true (P, P) :-
    fact P.
is_true (C, C ⇐ ProofTreeA) :-
    if A then C, is_true (A, ProofTreeA)
is_true (P1 and P2, ProofTree1 and ProofTree2) :-
    is_true (P1, ProofTree1), is_true (P2, ProofTree2).
is_true (P1 or _, ProofTree1) :- is_true (P1, ProofTree1).
is_true (_ or P2, ProofTree2) :- is_true (P2, ProofTree2).
```

Page No.

Date

6. Write a LISP Program to implement STEEPEST-ASCENT HILL CLIMBING.

```
(setf (get 's 'neighbours) '(a d)
      (get 'a 'neighbours) '(s b d)
      (get 'b 'neighbours) '(a c e)
      (get 'c 'neighbours) '(b)
      (get 'd 'neighbours) '(s a e)
      (get 'e 'neighbours) '(b d f)
      (get 'f 'neighbours) '(e))

(setf (get 's 'neighbours) '(0 3)
      (get 'a 'neighbours) '(4 6)
      (get 'b 'neighbours) '(7 6)
      (get 'c 'neighbours) '(11 6)
      (get 'd 'neighbours) '(3 0)
      (get 'e 'neighbours) '(6 0)
      (get 'f 'neighbours) '(11 3))

(defun straight-line-distance (node1 node2)
  "Compute the straight line distance between two nodes"
  (let ((coord1 (get node1 'coordinates))
        coord2 (get node2 'coordinates)))
    (sqrt (+ (expt (- (first coord1) (first coord2)) 2)
             (expt (- (second coord1) (second coord2)) 2
                   ))))
```

Delight

Page No.

Date

```
(defun extend-path (path)
  "Extend the path by consing on new neighbor
   nodes that dont result in loops"
  (mapcar #'(lambda (new-node) (cons new-node path))
          (remove-if
           #'(lambda (neighbor) (member neighbor path))
           (get (first path) 'neighbor))))

(defun closerp (node1 node2 target)
  "Returns true if node1 is closer to target than
   node2, else false"
  (< (straight-line-distance node1 target)
     (straight-line-distance node2 target)))

(defun hill-climb (start finish &optional (queue
                                           (list (list start))))
  "Perform hill climbing search for path from
   start to finish"
  (cond ((endp queue) nil)
        ((eq finish (first (first queue)))
         (reverse (first queue)))
        (t
         (print queue)
         (hill-climb
          start finish
          (append (sort (extend-path (first queue))
                        #'(lambda (p1 p2).
```

Delight

```
(closeup (first p1) (first p2) finish)))
(rest queue)))))
```

Page No.
Date

7. Write a prolog program to implement COUNTER PROPAGTION NETWORK.

About counter propagation network:-

CPN is a multilayer network based on the combinations of I/p, O/p and clustering layers. The application of CPN are data compression, function approximation & pattern association.
This model is 3 layer Neural Network that performs input-output data mapping, producing an output vector(y) in response to i/p vector (x) on the basis of competitive learning.

Full counterpropagation network:-
Full CPN represents a large number of vector pair x:y by adaptively constructing a look-up table. The full CPN works best if the inverse function exists & is continuous.

Traning Algorithm for Full CPN:-

Step 0:- Set the initial wts. & initial learning rate.

Step 1:- Perform Steps 2-7 if stopping condition is false for phase-I traning.

Delight

Page No. .

Date . .

Step 2:- For each of the training input vector pair x:y presented, perform Step 3-5.

Step 3:- Make the X-input layer activations to vector X. Make the Y-input layer activations to vector Y.

Step 4:- Find the winning cluster unit, If dot product method is used, find the cluster unit Z_j with target net input: for $j = 1$ to p

$$Z_{inj} = \sum_{i=1}^{n} x_i v_{ij} + \sum_{k=1}^{m} y_k w_{kj}$$

If euclidean distance method is used, find the cluster unit Z_j whose squared distance from i/p vector is the smallest

$$D(j) = \sum_{i=1}^{n} (x_i - v_{ij})^2 + \sum_{k=1}^{m} (y_k - w_{kj})^2$$

Step 5:- Update the weights over the calculated winner unit Z_j

$$v_{ij}(new) = v_{ij}(old) + \alpha [x_i - v_{ij}(old)] \quad i = 1 \text{ to } n$$
$$w_{kj}(new) = w_{kj}(old) + \beta [y_k - w_{kj}(old)] \quad k = 1 \text{ to } m$$

Step 6:- Reduce the learning rates α & β

$$\alpha(t+1) = 0.5\alpha t$$
$$\beta(t+1) = 0.5\beta t.$$

Page No.
Date

Step 7:- Test stopping condition for phase-I training

Step 8:- Perform step 9-15 when stopping condition is false for phase-II training.

Step 9: Perform 10-13 for each training input pair $x:y$. Here α & β are small constant values.

Step 10: Make the X-input layer activations to vector x. Make the Y-input layer activations to vector y.

Step 11: find the winning cluster unit (use formulas for step 4). Take the winner unit index as J

Step 12: Update the weights entering into unit Z_j

$$v_{ij}(new) = v_{ij}(old) + \alpha[x_i - v_{ij}(old)] \quad i = 1-n$$

$$v_{kj}(new) = w_{kj}(old) + \beta[y_k - w_{kj}(old)] \quad k = 1-m$$

Step 13:- Update the weights from unit Z_j to o/p layers.

$$t_{ji}(new) = t_{ji}(old) + b[x_i - t_{ji}(old)] \quad i = 1 \text{ to } n$$

$$u_{jk}(new) = u_{jk}(old) + a[y_k - u_{jk}(old)] \quad k = 1 \text{ to } m.$$

Step 14:- Reduce the learning rates a & b.

$$a(t+1) = 0.5\,at$$

$$b(t+1) = 0.5\,bt$$

Step 15:- Test stopping condition for phase-II training.

Page No.

Date

8. Development of a small Expert System using PROLOG.

Expert System on Diagnosis of Mental Disorders.

```
diagnose:-
  write('This is an expert system for dignosis of mental
  disorders.'), nl,
  write('There are several questions you need to answer
  for diagnosis of mental disorders.'), nl, nl, disorder(X),
  write('Condition was diagnosed as'),
  write(X),
  write('.').
diagnose:-
  write('The diagnose was not found.').

question(Attribute, Value):-
  retract(yes, Attribute, Value), !.
question(Attribute, Value):-
  retract(_, Attribute, Value), !, fail.
question(Attribute, Value):-
  write('Is the'),
  write(Attribute),
  write('-'),
  write(Value),
  write('?'),
  read(Y),
  asserta(retract(Y, Attribute, Value)).
```

Page No.
Date

```
Y == yes.

questionWithPossibilities(Attribute, Value, Possibilities):-
    write('What is the patient's'), write(Attribute),
    write('?'), nl, write(Possibilitities), nl,
    read(X),
     check_val(X, Attribute, Value, Possibilities),
     asserta(retract(yes, Attribute, X)),
     X == Value.


check_val(X, _, _, Possibilities):- member(X, Possibilities), !.
check_val(X, Attribute, Value, Possibilities):-
    write(X), write('is not a legal value, try again.'), nl,
    questionWithPossibilities(Attribute, Value, Possibilities).

:-dynamic(retract/3).

food_amount(X):- question(food_amount, X).
symptom(X):- question(symptom, X).
mentality(X):- question(mentality, X).
cause(X):- question(cause, X).
indication(X):- question(indication, X).
social_skill(X):- question(social_skill, X).
condition(X):- question(condition, X).
consequence(X):- question(consequence, X).
speciality(X):- question(speciality, X).
face_features(X):- question(face_features, X).
```

Delight

Page No.
Date

```
ears_features(X) :- question(ears_features, X).
brain_function(X) :- question(brain_function, X).
perceptions(X) :- question(perceptions, X).
behaviour(X) :- questionWithPossibilities(behaviour, X,
    [repetitive_and_restricted, narcissistic, aggresive]).

disorder(anorexia_nervosa) :- type(eating_disorder),
                              consequence(low_weight),
                              food_amount(food_restriction).

disorder(bulimia_nervosa) :- type(eating_disorder),
                             consequence(purging),
                             food_amount(binge_eating).

disorder(asperger_syndrome) :- type(neurodevelopmental_disorder),
                               specialty(psychiatry),
                               social_skill(low),
                               behavior(repetitive_and_restricted).

disorder(dyslexia) :- type(neurodevelopmental_disorder),
                      social_skill(normal),
                      perceptions(low),
                      symptom(trouble_reading).

disorder(autism) :- type(neurodevelopmental_disorder),
                    social_skill(low),
                    symptom(impaired_communication).
```

Delight

Page No.

Date

```
disorder(tourettes_syndrome) :- type(neurodevelopmental_disorder),
                                social_skill(normal),
                                specialty(neurology)
                                symptom(motor_tics).

disorder(bipolar_disorder) :- type(psychotic_disorder),
                              indication(elevated_moods).

disorder(schizophrenia) :- type(psychotic_disorder),
                           indication(hallucinations).

disorder(down_syndrome) :- type(genetic_disorder),
                           symptom(delayed_physical_growth),
                           face_features(long_and_narrow),
                           ears_features(large),
                           brain_function(intellectual_disability).

disorder(fragile_X_syndrome) :- type(genetic_disorder),
                                face_features(small_chin_and_slanted_eyes),
                                brain_function(intellectual_disability).

type(eating_disorder) :- symptom(abnormal_eating_habits),
                         mentality(strong_desire_to_be_thin).

type(neurodevelopmental_disorder) :- condition(affected_nervous_system),
                                     brain_function(abnormal),
                                     cause(genetic_and_environmental).
```

Delight

Page No.

Date

type(psychotic_disorder):- symptom(false_beliefs),
mentality(manic_depressive),
cause(genetic_and enviromental).

type(genetic_disorder):- cause(abnormalities_in_genome).

Page No.

Date

9. Development a prolog program for monkey - banana problem.

```
do (state (middle, onbox, middle, hasnot)    % grab banana
    grab,
    state ( middle, onbox, middle, has)).

do (state (L, onfloor, L, Banana)            % climb box
    climb,
    state (L, onbox, L, Banana)).

do (state (L1, onfloor, L1, Banana),    % push box from L1→L2
    push ( L1, L2),
    state (L2, onfloor, L2, Banana)).

do ( state (L1, onfloor, Box, Banana),  % walk from L1 to L2
    walk (L1, L2),
    state (L2, onfloor, Box, Banana)).

canget (state (_, _, _, has)).

canget (State1):-
    do (State1, Action, State2),
    cagnet (State2).
```

Page No.

Date

```
canget (state(_,_,_,has), []).

canget (State1, Plan):-
    do (State1, Action, State2),
    canget (State2, PartialPlan),
    add (Action, PartialPlan, Plan).

add (X, L, [X|L]).
```

10. Write a program for parsing a given sentence:-

parse a sentance into article, noun, verb, preposition & adjective.

```
np([X|T], np(det(X), NP2), Rem):-
    det(X),
    np2(T, NP2, Rem).
np(Sentance, Parse, Rem):- np2(Sentence, Parse, Rem).
np(Sentance, np(NP, PP), Rem):-
    np(Sentance, NP, Rem1),
    pp(Rem1, PP, Rem).

np2([H|T], np2(noun(H)), T):- noun(H).
np2([H|T], np2(adj(H), Rest), Rem):-adj(H), np2(T, Rest, Re

pp([H|T], pp(prep(H), Parse), Rem):-
    prep(H),
    np(T, Parse, Rem).

vp([H|[]], vp(verb(H))):-
    verb(H).

vp([H|T], vp(verb(H), Rem)):-
    vp(H, Rem),
    pp(T, Rem, _).
```

Page No.

Date

```
vp([H|T], vp(verb(H), Rem)):-
    verb(H),
    np(T, Rem, _).
```