KKR & KSR INSTITUTE OF TECHNOLOGY AND SCIENCES

Approved by AICTE, New Delhi, Affiliated to JNTUK Kakinada)
(Accredited by NBA , Accredited by NAAC – 'A' Grade)

Department of Computer Science and Engineering

LAB MANUAL
R20(AUTONOMOUS)

Artificial Intelligence Using Python Lab
[III B.TECH, II-SEM]

KKR & KSR INSTITUTE OF TECHNOLOGY AND SCIENCES

Vinjanampadu, Guntur District- 522017 (A. P.)

Document NO: KITS/CSE/LAB MANUAL/AI	Date of issue:	Compiled by	Authorized by Prof. R.RAMESH HoD DEPT. OF CSE
	Date of revision	Verified by	

KKR & KSR INSTITUTE OF TECHNOLOGY AND SCIENCES
Approved by AICTE, New Delhi, Affiliated to JNTUK Kakinada)
(Accredited by NBA , Accredited by NAAC – 'A' Grade)

Department of Computer Science Engineering
LAB MANUAL
(R20)

Artificial Intelligence Using Python Lab

[III B.TECH, SEM-II]

INDEX

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE INSTITUTE

To produce eminent and ethical Engineers and Managers in society by imparting quality professional education with emphasis on human values and holistic excellence.

MISSION OF THE INSTITUTE

- To incorporate benchmarked teaching and learning pedagogies in curriculum.
- To ensure all round development of students through judicious blend of curricular, co-curricular and extracurricular activities.
- To support cross-cultural exchange of knowledge between industry and academy.
- To provide higher/continued education and research opportunities to the employees of the institution.

DEPARTMENT VISION

- To become a reputed center in computer Science and systems engineering for quality, competency and social responsibility

DEPARTMENT MISSION

- Strengthen the Core Competence with Vibrant Technological Education in a congenial Environment
- Promote innovative research and development for the Economic, Social and Environment.
- Inculcate professional behavior, strong ethical values to meet the challenges in collaboration and lifelong learning

.

PROGRAM EDUCATIONAL OBJECTIVES OF CSE DEPARTMENT

PEO 1: Domain Knowledge: Have a strong foundation in areas like mathematics, science and engineering fundamentals so as to enable them to solve and analyze engineering problems and prepare them to careers, R&D and studies of higher level.
PEO 2: Professional Employment: Have an ability to analyze and understand the requirements of software, technical specifications required and provide novel engineering solutions to the problems associated with hardware and software.
PEO 3: Higher Degrees: Have exposure to cutting edge technologies thereby making them to achieve excellence in the areas of their studies.
PEO 4: Engineering Citizenship: Work in teams on multi-disciplinary projects with effective communication skills and leadership qualities.
PEO 5: Lifelong Learning: Have a successful career wherein they strike a balance between ethical values and commercial values.

PROGRAM OUTCOMES (PO'S)

1. Engineering knowledge:

Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis:

Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions:

Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems:

Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage:

Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society:

Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability:

Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics:

Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work:

Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication:

Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance:

Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning:
Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOME (PSO'S)
PSO1: Application Development Able to develop the business solutions through Latest Software Techniques and tools for real time Applications.
PSO2: Professional and Leadership Able to practice the profession with ethical leadership as an entrepreneur through participation in various events like Ideathon, Hackathon, project expos and workshops.
PSO3: Computing Paradigms Ability to identify the evolutionary changes in computing using Data Sciences, Apps, Cloud computing and IoT.

Course Code	Course Name	L	T	P	C
20CS6L03	**ARTIFICIAL INTELLIGENCE LAB USING PYTHON**	**0**	**0**	**3**	**1.5**

Course Objectives:

- Study the concepts of Artificial Intelligence
- Learn the methods of solving problems using Artificial Intelligence
- Introduce the concept of machine learning.

List of Experiments

1) Study of Prolog.
2) Write simple fact for the statements using PROLOG.
3) Write predicates One converts centigrade temperatures to Fahrenheit, theother checks if a temperature is below freezing
4) Write a program to solve the Monkey Banana problem.
5) Write a program in turbo prolog for medical diagnosis and show the advantageand disadvantage of green and red cuts
6) Write a program to implement factorial, Fibonacci of a given number
7) Write a program to solve 4-Queen and 8-puzzle problem.
8) Write a program to solve traveling salesman problem.
9) Write a program to solve water jug problem using LISP
10) Implementation of A* Algorithm using LISP /PROLOG
11) Implementation of Hill Climbing Algorithm using LISP /PROLOG
12) Implementation of DFS and BFS for water jug problem using LISP /PROLOG
13) Implementation of Towers of Hanoi Problem using LISP /PROLOG.

E-Resources:

1) https://www.studocu.com/in/document/dr-apj-abdul-kalam-technical-university/artificial-intelligence-lab/artificial-intelligence-lab-files/16507070
2) https://www.tutorialspoint.com/prolog/prolog_towers_of_hanoi_problem.ht m
3) https://www.goeduhub.com/6687/implementation-dfs-for-water-jug-problem-using-lisp-prolog

Course Outcomes:

At the end of the course, the students will be able to:

CO-1: Identify problems that are amenable to solution by AI methods
CO-2: Identify appropriate AI methods to solve a given problem
CO-3: Use language/framework of different AI methods for solving problems
CO-4: Implement basic AI algorithms
CO-5: Design and carry out an empirical evaluation of different algorithms onproblem formalization, and state the conclusions that the evaluation supports

MAPPING OF CO's WITH PO's

PO / CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO 12
CO.1	3	3	3		3							
CO.2	3	3	2		3							
CO.3		3	2									
CO.4			3	2								
CO.5			3		3							

MAPPING OF COs WITH PSO's

PSO/CO NO:	PSO1	PSO2	PSO3
CO.1	2		
CO.2	3		
CO.3	3		
CO.4		2	
CO.5		2	

Note: 1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

Mapping Of Co's With Lab Experiments:

EXPERIMENT	CS6L03.1	CS6L03.2	CS6L03.3	CS6L03.4	CS6L03.5
EX1	3				
EX2	3				
EX3	3				
EX4		3			
EX5			3		
EX6			3		
EX7			3		
EX8			3		
EX9			3		
EX10			3		
EX11				3	
EX12				3	
EX13					3
ADDITIONAL EXP					3

Course Code	Course Name	L	T	P	C
20CS6T03	**Artificial Intelligence**	**3**	**0**	**0**	**3**

Course Objectives:
• To have a basic proficiency in a traditional AI language including an ability to write simple to intermediate programs and an ability to understand code written in that language.
• To have an understanding of the basic issues of knowledge representation and blind and heuristic search, as well as an understanding of other topics such as minimax, resolution, etc. that play an important role in AI programs.
• To have a basic understanding of some of the more advanced topics of AI such as learning, natural language processing, agents and robotics, expert systems, and planning.

Course Outcomes:
At the end of the course, the students will be able to:
CO-1: Outline problems that are amenable to solution by AI methods, and which AI methods may be suited to solving a given problem
CO-2: Apply the language/framework of different AI methods for a given problem
CO-3: Implement basic AI algorithms- standard search algorithms or dynamic programming
CO-4: Design and carry out an empirical evaluation of different algorithms on problem formalization, and state the conclusions that the evaluation supports.
CO-5: Outline various uncertainty measures, fuzzy sets and fuzzy logic applications.

UNIT I
Introduction:history, intelligent systems, foundations of AI, applications, tic-tac-toe game playing, development of AI languages, current trends.

UNIT II
Problem solving: state-space search and control strategies: Introduction, general problem solving, characteristics of problem, exhaustive searches, heuristic search techniques, iterative deepening A*, constraint satisfaction.
Problem reduction and game playing: Introduction, problem reduction, game playing, alpha beta pruning, two-player perfect information games.

UNIT III
Logic concepts: Introduction, propositional calculus, proportional logic, natural deduction system, axiomatic system, semantic tableau system in proportional logic, resolution refutation in proportional logic, predicate logic.

UNIT IV
Knowledge representation: Introduction, approaches to knowledge representation, knowledge representation using semantic network, extended semantic networks for KR, knowledge representation using frames.
Advanced knowledge representation techniques: Introduction, conceptual dependency theory, script structure, CYC theory, case grammars, semantic web

UNIT V
Expert system and applications: Introduction phases in building expert systems, expert system versus traditional systems
Uncertainty measure: probability theory: Introduction, probability theory, Bayesian belief networks, certainty factor theory, dempster-shafer theory.
Fuzzy sets and fuzzy logic:Introduction, fuzzy sets, fuzzy set operations, types of membership functions, multi valued logic, fuzzy logic, linguistic variables and hedges, fuzzy propositions, inference rules for fuzzy propositions, fuzzy systems.

Text Books:
1) Artificial Intelligence- SarojKaushik, CENGAGE Learning
2) Artificial intelligence, A modern Approach, 2nded, Stuart Russel, Peter Norvig, PEA

Reference Books:
1) Artificial Intelligence- Deepak Khemani, TMH, 2013
2) Introduction to Artificial Intelligence, Patterson, PHI
3) Artificial intelligence, structures and Strategies for Complex problem solving, - George F Lugar, 5thed, PEA

E-Resources:
1) https://nptel.ac.in/courses/106/105/106105077/
2) http://aima.cs.berkeley.edu/

ARTIFICIAL INTELLIGENCE LAB USING PYTHON

LAB MANUAL

EXPERIMENT-1: OBJECTIVE: Study of Prolog.

PROLOG-PROGRAMMING IN LOGIC

PROLOG stands for Programming, In Logic — an idea that emerged in the early 1970's to use logic as programming language. The early developers of this idea included Robert Kowaiski at Edinburgh (on the theoretical side), Marrten van Emden at Edinburgh (experimental demonstration) and Alian Colmerauer at Marseilles (implementation).

David D.H. Warren's efficient implementation at Edinburgh in the mid -1970's greatly contributed to the popularity of PROLOG. PROLOG is a programming language centred around a small set of basic mechanisms, Including pattern matching, tree based data structuring and automatic backtracking. This Small set constitutes a surprisingly powerful and flexible programming framework. PROLOG is especially well suited for problems that involve objects- in particular, structured objects- and relations between them.

SYMBOLIC LANGUAGE

PROLOG is a programming language for symbolic, non-numeric computation. It is especially well suited for solving problems that involve objects and relations between objects. For example, it is an easy exercise in prolog to express spatial relationship between objects, such as the blue sphere is behind the green one. It is also easy to state a more general rule: if object X is closer to the observer than object Y. and object Y is closer than Z, then X must be closer than Z. PROLOG can reason about the spatial relationships and their consistency with respect to the general rule. Features like this make PROLOG a powerful language for ArtJIcia1 LanguageA1,) and non- numerical programming.

There are well-known examples of symbolic computation whose implementation in other standard languages took tens of pages of indigestible code, when the same algorithms were implemented in PROLOG, the result was a crystal-clear program easily fitting on one page.

FACTS, RULES AND QUERIES

Programming in PROIOG is accomplished by creating a database of facts and rules about objects, their properties, and their relationships to other objects. Queries then can be posed about the objects and valid conclusions will be determined and returned by the program Responses to user queries are determined through a form of inference control known as resolution.

FOR EXAMPLE:

a) FACTS:

Some facts about family relationships could be written as: sister(sue,bill)
parent(ann.sam) male(jo)
female(riya)

b) RULES:

To represent the general rule for grandfather, we write: grand f.gher(X2)
parent(X,Y) parent(Y,Z) male(X)

c) QUERIES:

Given a database of facts and rules such as that above, we may make queries by typing after a query a symbol’?’ statements such as:
?-parent(X,sam) Xann
?grandfather(X,Y) X=jo, Y=sam

PROLOG IN DISGINING EXPERT SYSTEMS

An expert system is a set of programs that manipulates encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An expert system’s knowledge is obtained from expert sources such as texts, journal articles. databases etc and encoded in a form suitable for the system to use in its inference or reasoning processes. Once a sufficient body of expert knowledge has been acquired, it must be encoded in some form, loaded into knowledge base, then tested, and refined continually throughout the life of the system PROLOG serves as a powerful language in designing expert systems because of its following features.

- **Use of knowledge rather than data**
- **Modification of the knowledge base without recompilation of the control programs.**
- **Capable of explaining conclusion.**

META PROGRAMMING

A meta-program is a program that takes other programs as data. Interpreters and compilers are examples of meta-programs. Meta-interpreter is a particular kind of meta-program: an interpreter for a language written in that language. So a PROLOG interpreter is an interpreter for PROLOG, itself written in PROLOG. Due to its symbol- manipulation capabilities, PROLOG is a powerful language for meta-programming. Therefore, it is often used as an implementation language for other languages. PROLOG is particularly suitable as a language for rapid prototyping where we are interested in implementing new ideas quickly. New ideas are rapidly implemented and experimented with.

OUTCOME: Students will get the basic idea of how to program in prolog and its working environment.

EXPERIMENT-2: OBJECTIVE: Write simple fact for following:

a. **Ram likes mango.**
b. **Seema is a girl.**
c. **Bill likes Cindy.**
d. **Rose is red.**
e. **John owns gold.**

Program:Clauses

```
likes(ram ,mango). girl(seema). red(rose). likes(bill ,cindy). owns(john ,gold).
```

Output:

Goal queries

```
?-likes(ram,What). What= mango
?-likes(Who,cindy). Who= cindy
?-red(What). What= rose
?-owns(Who,What). Who= john
What= gold.
```

OUTCOME: Student will understand how to write simple facts using prolog.

EXPERIMENT-3: OBJECTIVE: Write predicates One converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.
Production rules:
Arithmetic:

```
c_to_f   f is c * 9 / 5 +32
freezing            f < = 32
```

Rules:

```
c_to_f(C,F) :-
F is C * 9 / 5 + 32. freezing(F) :-
F =< 32.
```

Output:
Queries:

```
?- c_to_f(100,X). X = 212
Yes
?- freezing(15)
.Yes
?- freezing(45). No
```

OUTCOME: Student will understand how to write a program using the rules.

EXPERIMENT-4: OBJECTIVE: Write a program to solve the Monkey Banana problem.
Imagine a room containing a monkey, chair and some bananas. That have been hanged from the centre of ceiling. If the monkey is clever enough he can reach the bananas by placing the chair directly below the bananas and climb on the chair .The problem is to prove the monkey can reach the bananas.The monkey wants it, but cannot jump high enough from the floor. At the window of the room there is a box that the monkey can use. The monkey can perform the following actions:-

1) **Walk on the floor.**
2) **Climb the box.**
3) **Push the box around (if it is beside the box).**
4) **Grasp the banana if it is standing on the box directly under the banana.**

Production Rules
can_reach clever,close.
get_on: can_climb.
under in room,in_room, in_room,can_climb.
Close get_on,under| tall Parse Tree

Clauses:
in_room(bananas). in_room(chair). in_room(monkey). clever(monkey). can_climb(monkey, chair). tall(chair). can_move(monkey, chair, bananas).

```
can_reach(X, Y):-
clever(X),close(X, Y).
get_on(X,Y):- can_climb(X,Y). under(Y,Z):-
in_room(X),in_room(Y),in_room(Z),can_climb(X,Y,Z). close(X,Z):-get_on(X,Y),
under(Y,Z);
tall(Y). Output:
Queries:
?- can_reach(A, B). A = monkey.
B = banana.
?- can_reach(monkey, banana).Yes.
```

OUTCOME: Student will understand how to solve monkey banana problem using rules in prolog.

EXPERIMENT-5: OBJECTIVE: WAP in turbo prolog for medical diagnosis and show t he advantage and disadvantage of green and red cuts.

Program:

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:-
write("sorry, i am not ina 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).

Output:

makewindow(1,7,7"Expert Medical Diagnosis",2,2,23,70), go.

OUTCOME: Student will understand how to create a expert system using prolog.

EXPERIMENT-6: OBJECTIVE: WAP to implement factorial, fibonacci of a given number.

Program:

Factorial:

```
factorial(0,1). factorial(N,F) :-
N>0,
N1 is N-1,
factorial(N1,F1), F is N * F1.
```

Output:
Goal:

```
?- factorial(4,X). X=24
```

Fibonacci:

```
fib(0, 0).
fib(X, Y) :- X > 0, fib(X, Y, _). fib(1, 1, 0).
fib(X, Y1, Y2) :- X > 1,
X1 is X - 1, fib(X1, Y2, Y3), Y1 is Y2 + Y3.
```

Output:
Goal:

```
?-fib(10,X). X=55
```

OUTCOME: Student will understand the implementation of Fibonacci and factorial series using prolog.

EXPERIMENT-7: OBJECTIVE: Write a program to solve 4-Queen problem.

Program:
In the 4 Queens problem the object is to place 4 queens on a chessboard in such a way that no queens can capture a piece. This means that no two queens may be placed on the same row, column, or diagonal.

The n Queens Chessboard.
domains
queen = q(integer, integer) queens = queen*
freelist = integer*
board = board(queens, freelist, freelist, freelist, freelist) predicates
nondeterm placeN(integer, board, board) nondeterm place_a_queen(integer, board, board) nondeterm nqueens(integer)
nondeterm makelist(integer, freelist)
nondeterm findandremove(integer, freelist, freelist) nextrow(integer, freelist, freelist)
clauses nqueens(N):- makelist(N,L), Diagonal=N*2-1,
makelist(Diagonal,LL), placeN(N,board([],L,L,LL,LL),Final), write(Final).
placeN(_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.

placeN(N,Board1,Result):- place_a_queen(N,Board1,Board2), placeN(N,Board2,Result). place_a_queen(N, board(Queens,Rows,Columns,Diag1,Diag2), board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):- nextrow(R,Rows,NewR), findandremove(C,Columns,NewC),
D1=N+C-R,findandremove(D1,Diag1,NewD1), D2=R+C-1,findandremove(D2,Diag2,NewD2).
findandremove(X,[X|Rest],Rest). findandremove(X,[Y|Rest],[Y|Tail]):- findandremove(X,Rest,Tail).
makelist(1,[1]).
makelist(N,[N|Rest]) :- N1=N-1,makelist(N1,Rest).
nextrow(Row,[Row|Rest],Rest). Output:
Goal:
?-nqueens(4),nl. board([q(1,2),q(2,4),q(3,1),q(4,3),[],[],[7,4,1],[7,4,1])
yes

OUTCOME: Student will implement 4-Queen problem using prolog.

EXPERIMENT-8: OBJECTIVE: Write a program to solve traveling salesman problem.

The following is the simplified map used for the prototype:

Program:

Production Rules:-

```
route(Town1,Town2,Distance) road(Town1,Town2,Distance). route(Town1,Town2,Distance)
road(Town1,X,Dist1),route(X,Town2,Dist2),Distance=Dist1+Dist2, domains
town = symbol distance = integer predicates
nondeterm road(town,town,distance) nondeterm route(town,town,distance)

clauses road("tampa","houston",200).
road("gordon","tampa",300).
road("houston","gordon",100). road("houston","kansas_city",120).

road("gordon","kansas_city",130). route(Town1,Town2,Distance):- road(Town1,Town2,Distance).
route(Town1,Town2,Distance):- road(Town1,X,Dist1), route(X,Town2,Dist2), Distance=Dist1+Dist2,!.
```

Output:

Goal:

```
route("tampa", "kansas_city", X),
write("Distance from Tampa to Kansas City is ",X),nl.

Distance from Tampa to Kansas City is 320 X=320
```

OUTCOME: Student will implement travelling salesmen problem using prolog.

EXPERIMENT-9: OBJECTIVE: Write a program to solve water jug problem using LISP.

```
;returns the quantity in first jug (defun get-first-jug (state) (car state))

;returns the quantity in second jug
(defun get-second-jug (state) (cadr state))

;returns the state of two jugs (defun get-state (f s) (list f s))

;checks whether a given state is a goal
; GOAL IS TO GET 4 IN SECOND JUG
(defun is-goal (state)
(eq (get-second-jug state) 4))

;returns all possible states that can be derived
;from a given state (defun child-states (state)
(remove-null (list
(fill-first-jug state) (fill-second-jug state)
(pour-first-second state) (pour-second-first state) (empty-first-jug state) (empty-second-jug state))))

;remove the null states (defun remove-null (x)
(cond ((null x) nil)
((null (car x)) (remove-null (cdr x)))
((cons (car x) (remove-null (cdr x))))))

;return the state when the first jug is filled (first jug can hold 3) (defun fill-first-jug (state)

(cond
((< (get-first-jug state) 3) (get-state 3 (get-second-jug state))))))

;returns the state when the second jug is filled (second jug can hold 5) (defun fill-second-jug (state)
(cond
((< (get-second-jug state) 5) (get-state (get-first-jug state) 5))))

;returns the state when quantity in first
;is poured to second jug
(defun pour-first-second (state) (let ( (f (get-first-jug state)) (s (get-second-jug state)))
(cond
((zerop f) nil) ; first jug is empty ((= s 5) nil) ; Second jug is full ((<= (+ f s) 5)
(get-state 0 (+ f s)))
(t        ; pour to first from second (get-state (- (+ f s) 5) 5)))))
```

```
;returns the state when second jug is poured to first (defun pour-second-first (state)
(let ( (f (get-first-jug state)) (s (get-second-jug state)))
(cond
((zerop s) nil) ; second jug is empty ((= f 3) nil) ; second jug is full
((<= (+ f s) 3)
(get-state (+ f s) 0))
(t        ;pour to second from first (get-state 3 (- (+ f s) 3))))))

;returns the state when first jug is emptied (defun empty-first-jug (state)
(cond
((> (get-first-jug state) 0) (get-state 0 (get-second-jug state)))))

;returns the state when second jug is emptied (defun empty-second-jug (state)
(cond
((> (get-second-jug state) 0) (get-state (get-first-jug state) 0))))

;;;MAIN FUNCTION
(defun dfs (start-state depth lmt) (setf *node* 0)
(setf *limit* lmt)
(dfs-node start-state depth)
)

;dfs-node expands a node and calls dfs-children to recurse on it (defun dfs-node (state depth)
(setf *node* (+ 1 *node*)) (cond
((is-goal state) (list state)) ((zerop depth) nil)
((> *node* *limit*) nil)
((let ((goal-child (dfs-children (child-states state) (- depth 1))))
(and goal-child (cons state goal-child)))) ;for back-tracking if the branch don't have a goal state
))
;dfs-children expands each node recursively and give it
;to dfs-node to process
(defun dfs-children (states depth) (cond
((null states) nil)
((dfs-node (car states) depth)) ((dfs-children (cdr states) depth))))
(print "ENTER YOUR INPUT AS")
(print "(dfs start_state depth limit)")
```

OUTCOME: Student will implement water-jug problem using Lisp.

EXPERIMENT-10: OBJECTIVE: Implementation of A* Algorithm using LISP/PROLOG

```
commands :-
  write("List of commands:"), nl, nl,
  write("print_coordinates"), nl,
  write("print_board"), nl,
  nl.

print_coordinates :-
  solution(A),
  display_coordinates(A, 1).

print_board :-
  solution(A),
  print_boundry,
  print_columns(A, 1),
  print_column_letters,
  nl.

print_square :- write("| ").
print_queen :- write("|Q").
print_end_quare :- write("|"), nl.
print_boundry :- write("   - - - - - - - - "), nl.
print_column_letters :- write("   h g f e d c b a "), nl.

print_columns([], _).
print_columns([H|T], C) :-
  write(C), write(" "),
  print_row(H, 1),
  C1 is C + 1,
  print_columns(T, C1).

print_row(_, 9) :-
  print_end_quare,
  print_boundry, !.
print_row(X, Col) :-
  X is Col, print_queen,
  Col1 is Col + 1, print_row(X, Col1).
```

```
print_row(X, Col) :-
  X =\= Col, print_square,
  Col1 is Col + 1, print_row(X, Col1).

solution(Queens) :-
 permutation([1,2,3,4,5,6,7,8], Queens),
 safe(Queens).

permutation([],[]).
permutation([H|T], P) :-
 permutation(T, P1),
 mydelete(H, P, P1).

mydelete(X, [X|L], L).
mydelete(X, [Y|L], [Y|L1]) :-
 mydelete(X, L, L1).

safe([]).
safe([Queen|Others]) :-
 safe(Others),
 noattack(Queen,Others,1).

noattack(_,[],_).
noattack(Y, [Y1|L], X1) :-
 Y1 - Y =\= X1,
 Y - Y1 =\= X1,
 X2 is X1 + 1,
 noattack(Y, L, X2).
display_coordinates([], 9).
display_coordinates([H|T], Y) :-
  nth1(H, [h,g,f,e,d,c,b,a], X),
  write("["), write(X),
  write(Y), write("]"), nl,
  Y1 is Y + 1,
  display_coordinates(T, Y1).
```

<u>OUTPUT:</u>

```
?- display_coordinates([],9).
True
```

EXPERIMENT-11: OBJECTIVE: Implementation of Hill Climbing Algorithm using LISP/PROLOG

```
link(Puzzle1, Puzzle2, Cost) :-
  linkOneWay(Puzzle1, Puzzle2, Cost).
linkOneWay( A/0/C/D/E/F/H/I/J , 0/A/C/D/E/F/H/I/J , 1).
linkOneWay( A/B/C/D/0/F/H/I/J , A/B/C/0/D/F/H/I/J , 1).
linkOneWay( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/0/H/J , 1).
linkOneWay( A/B/0/D/E/F/H/I/J , A/0/B/D/E/F/H/I/J , 1).
linkOneWay( A/B/C/D/E/0/H/I/J , A/B/C/D/0/E/H/I/J , 1).
linkOneWay( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/F/H/0/I , 1).

linkOneWay( A/B/C/0/E/F/H/I/J , 0/B/C/A/E/F/H/I/J , 1).
linkOneWay( A/B/C/D/0/F/H/I/J , A/0/C/D/B/F/H/I/J , 1).
linkOneWay( A/B/C/D/E/0/H/I/J , A/B/0/D/E/C/H/I/J , 1).
linkOneWay( A/B/C/D/E/F/0/I/J , A/B/C/0/E/F/D/I/J , 1).
linkOneWay( A/B/C/D/E/F/H/0/J , A/B/C/D/0/F/H/E/J , 1).
linkOneWay( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/0/H/I/F , 1).

linkOneWay( A/0/C/D/E/F/H/I/J , A/C/0/D/E/F/H/I/J , 1).
linkOneWay( A/B/C/D/0/F/H/I/J , A/B/C/D/F/0/H/I/J , 1).
linkOneWay( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/H/J/0 , 1).
linkOneWay( 0/B/C/D/E/F/H/I/J , B/0/C/D/E/F/H/I/J , 1).
linkOneWay( A/B/C/0/E/F/H/I/J , A/B/C/E/0/F/H/I/J , 1).
linkOneWay( A/B/C/D/E/F/0/I/J , A/B/C/D/E/F/I/0/J , 1).

linkOneWay( A/B/C/0/E/F/H/I/J , A/B/C/H/E/F/0/I/J , 1).
linkOneWay( A/B/C/D/0/F/H/I/J , A/B/C/D/I/F/H/0/J , 1).
linkOneWay( A/B/C/D/E/0/H/I/J , A/B/C/D/E/J/H/I/0 , 1).
linkOneWay( 0/B/C/D/E/F/H/I/J , D/B/C/0/E/F/H/I/J , 1).
linkOneWay( A/0/C/D/E/F/H/I/J , A/E/C/D/0/F/H/I/J , 1).
linkOneWay( A/B/0/D/E/F/H/I/J , A/B/F/D/E/0/H/I/J , 1).
h(state(P,_,_,_,_,Hvalue)) :- h2(state(P,_,_,_,_,Hvalue)).link
h1(state(P,_,_,_,_,Hvalue)) :- manhattan(P, Hvalue).
h2(state(P,_,_,_,_,Hvalue)) :- misplaced_tiles(P, Hvalue).
manhattan(A/B/C/D/E/F/G/H/I, P) :-
   a(A,Pa), b(B,Pb), c(C,Pc),
   d(D,Pd), e(E,Pe), f(F,Pf),
   g(G,Pg), h(H,Ph), i(I,Pi),
   P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pg+Pi.
misplaced_tiles(A/B/C/D/E/F/G/H/I, P) :-
   ma(A,Pa), mb(B,Pb), mc(C,Pc),
   md(D,Pd), me(E,Pe), mf(F,Pf),
   mg(G,Pg), mh(H,Ph), mi(I,Pi),
 P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pg+Pi.
a(0,0). a(1,0). a(2,1). a(3,2). a(4,3). a(5,4). a(6,3). a(7,2). a(8,1).
b(0,0). b(1,0). b(2,0). b(3,1). b(4,2). b(5,3). b(6,2). b(7,3). b(8,2).
c(0,0). c(1,2). c(2,1). c(3,0). c(4,1). c(5,2). c(6,3). c(7,4). c(8,3).
d(0,0). d(1,1). d(2,2). d(3,3). d(4,2). d(5,3). d(6,2). d(7,2). d(8,0).
e(0,0). e(1,2). e(2,1). e(3,2). e(4,1). e(5,2). e(6,1). e(7,2). e(8,1).
f(0,0). f(1,3). f(2,2). f(3,1). f(4,0). f(5,1). f(6,2). f(7,3). f(8,2).
g(0,0). g(1,2). g(2,3). g(3,4). g(4,3). g(5,2). g(6,2). g(7,0). g(8,1).
h(0,0). h(1,3). h(2,3). h(3,3). h(4,2). h(5,1). h(6,0). h(7,1). h(8,2).
i(0,0). i(1,4). i(2,3). i(3,2). i(4,1). i(5,0). i(6,1). i(7,2). i(8,3).

ma(0,0). ma(1,0). ma(2,1). ma(3,1). ma(4,1). ma(5,1). ma(6,1). ma(7,1). ma(8,1).
mb(0,0). mb(1,1). mb(2,0). mb(3,1). mb(4,1). mb(5,1). mb(6,1). mb(7,1). mb(8,1).
mc(0,0). mc(1,1). mc(2,1). mc(3,0). mc(4,1). mc(5,1). mc(6,1). mc(7,1). mc(8,1).
md(0,0). md(1,1). md(2,1). md(3,1). md(4,1). md(5,1). md(6,1). md(7,1). md(8,0).
me(0,0). me(1,1). me(2,1). me(3,1). me(4,1). me(5,1). me(6,1). me(7,1). me(8,1).
mf(0,0). mf(1,1). mf(2,1). mf(3,1). mf(4,0). mf(5,1). mf(6,1). mf(7,1). mf(8,1).
mg(0,0). mg(1,1). mg(2,1). mg(3,1). mg(4,1). mg(5,1). mg(6,1). mg(7,0). mg(8,1).
mh(0,0). mh(1,1). mh(2,1). mh(3,1). mh(4,1). mh(5,1). mh(6,0). mh(7,1). mh(8,1).
mi(0,0). mi(1,1). mi(2,1). mi(3,1). mi(4,1). mi(5,0). mi(6,1). mi(7,1). mi(8,1).
extract(state(NextP,_,_,_,_,_), NextP).
notIn(_, []).
notIn(P, [P1|R]) :- P \= P1, notIn(P, R).
repeat(P, [P|_]).
insert(P, A, A) :- repeat(P, A), !.
insert(P, [P1|R], [P1|S]) :- insert(P, R, S), !.
insert(P, [], [P]).
```

OUTPUT:

```
---------
link(Puzzle, Puzzle2, Cost).
Puzzle = _A/0/_B/_C/_D/_E/_F/_G/_H,
Puzzle2 = 0/_A/_B/_C/_D/_E/_F/_G/_H,
Cost = 1 ;
Puzzle = _A/_B/_C/_D/0/_E/_F/_G/_H,
Puzzle2 = _A/_B/_C/0/_D/_E/_F/_G/_H,
Cost = 1 ;
Puzzle = _A/_B/_C/_D/_E/_F/_G/0/_H,
Puzzle2 = _A/_B/_C/_D/_E/_F/0/_G/_H,
Cost = 1 ;

Puzzle = _A/_B/0/_C/_D/_E/_F/_G/_H,
Puzzle2 = _A/0/_B/_C/_D/_E/_F/_G/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/_D/_E/0/_F/_G/_H,
Puzzle2 = _A/_B/_C/_D/0/_E/_F/_G/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/_D/_E/_F/_G/_H/0,
Puzzle2 = _A/_B/_C/_D/_E/_F/_G/0/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/0/_D/_E/_F/_G/_H,
Puzzle2 = 0/_B/_C/_A/_D/_E/_F/_G/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/_D/0/_E/_F/_G/_H,
Puzzle2 = _A/0/_C/_D/_B/_E/_F/_G/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/_D/_E/0/_F/_G/_H,
Puzzle2 = _A/_B/0/_D/_E/_C/_F/_G/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/_D/_E/_F/0/_G/_H,
Puzzle2 = _A/_B/_C/0/_E/_F/_D/_G/_H,
Cost = 1 ;
Puzzle = _A/_B/_C/_D/_E/_F/_G/0/_H,
Puzzle2 = _A/_B/_C/_D/0/_F/_G/_E/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/_D/_E/_F/_G/_H/0,
Puzzle2 = _A/_B/_C/_D/_E/0/_G/_H/_F,
Cost = 1 ;

Puzzle = _A/0/_B/_C/_D/_E/_F/_G/_H,
Puzzle2 = _A/_B/0/_C/_D/_E/_F/_G/_H,
Cost = 1 ;

Puzzle = _A/_B/_C/_D/0/_E/_F/_G/_H,
Puzzle2 = _A/_B/_C/_D/_E/0/_F/_G/_H,
Cost = 1 .
```

EXPERIMENT 12: Implementation of DFS and BFS for water jug problem using LISP /PROLOG

```
water_jug(X,Y):- X>4,Y<3,write('4L water jug overflowed'),nl.
water_jug(X,Y):- X<4,Y>3,write('3L water jug overflowed'),nl.
water_jug(X,Y):- X>4,Y>3,write('Both water jugs overflowed'),nl.
water_jug(X,Y):-(X=:=0,Y=:=0,nl,write('4L:0 & 3L:3 (Action: Fill 3L jug)'),YY is 3,
water_jug(X,YY));

 (X=:=0,Y=:=0,nl,write('4L:4 & 3L:0 (Action: Fill 4L jug)'),XX is 4, water_jug(XX,Y));
(X=:=2,Y=:=0,nl,write('4L:2 & 3L:0 (Action: Goal state is reached)'));
(X=:=4,Y=:=0,nl,write('4L:1 & 3L:3 (Action: Pour water from 4L to 3L jug)'),XX is X-
3,YY is 3,water_jug(XX,YY));

(X=:=0,Y=:=3,nl,write('4L:3 & 3L:0 (Action: Pour water from 3L to 4L jug)'),
XX is 3,YY is 0,water_jug(XX,YY));

 (X=:=1,Y=:=3,nl,write('4L:1 & 3L:0 (Action: Empty 3L jug)'),
 YY is 0,water_jug(X,YY));
 (X=:=3,Y=:=0,nl,write('4L:3 & 3L:3 (Action: Fill 3L jug)'),YY is 3,water_jug(X,YY));
 (X=:=3,Y=:=3,nl,write('4L:4 & 3L:2 (Action: Pour water from 3L to 4L jug until 4L
jug is full)'),XX is X+1,YY is Y-1,water_jug(XX,YY));
(X=:=1,Y=:=0,nl,write('4L:0 & 3L:1 (Action: Pour water from 4L to 3L jug)'),XX is
Y,YY is X,water_jug(XX,YY));

(X=:=0,Y=:=1,nl,write('4L:4 & 3L:1 (Action: Fill 4L jug)'),XX is 4,water_jug(XX,Y));
(X=:=4,Y=:=1,nl,write('4L:2 & 3L:3 (Action: Pour water from 4L to 3L jug until 3L
jug is full)'),XX is X-2,YY is Y+2,water_jug(XX,YY));
(X=:=2,Y=:=3,nl,write('4L:2 & 3L:0 (Action: Empty 3L jug)'),
 YY is 0,water_jug(X,YY));
(X=:=4,Y=:=2,nl,write('4L:0 & 3L:2 (Action: Empty 4L jug)'),
 XX is 0, water_jug(XX,Y));
(X=:=0,Y=:=2,nl,write('4L:2 & 3L:0 (Action: Pour water from 3L jugto 4L jug)'),XX is
Y,YY is X, water_jug(XX,YY)).
```

output:

```
water_jug(10,0).
water_jug(0,10).
water_jug(10,10).
water_jug(0,0).
water_jug(2,0).
```

EXPERIMENT 13: Implementation of Towers of Hanoi Problem using LISP /PROLOG

```
move(1,X,Y,_) :-
   write('Move top disk from '),
   write(X),
   write(' to '),
   write(Y),
   nl.

move(N,X,Y,Z) :-
   N>1,
   M is N-1,
   move(M,X,Z,Y),
   move(1,X,Y,_),
   move(M,Z,Y,X).
```

OUTPUT:

```
?-move(3,left,right,center).
```

ADDITIONAL EXPERIMENT:

Objective: Implementation Expert System with backward chaining using RVD/PROLOG.

Program with three facts and one rule:

```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
```

<u>OUTPUT:</u>

Query and response:

```
?- snowy(rochester).
yes
```

Query and response:

```
?- snowy(seattle).
no
```

Query and response:

```
?- snowy(paris).
No
```