MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Department of CSE

(Emerging Technologies)

(DATA SCIENCE, INTERNET OF THINGS)

B.TECH(R-20 Regulation)

(III YEAR – I SEM)

(2023-24)

ARTIFICIAL INTELLIGENCE

(R20A0588)

MALLA REDDY COLLEGE OF ENGINEERING &TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12(B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE-Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad–500100, Telangana State, India

Department of Computer Science and Engineering

EMERGING TECHNOLOGIES

ARTIFICIAL INTELLIGENCE
(R20A0588)

LABORATORY MANUAL

Prepared by
D Kalpana, Assistant Professor

On
27.06.2022

Updated by
R. Shashi Rekha, Assistant Professor
A. Naveen Kumar, Assistant Professor

On
29.08.2023

Department of Computer Science and Engineering

Emerging Technologies

Vision

- “To be at the forefront of Emerging Technologies and to evolve as a Centre of Excellence in Research, Learning and Consultancy to foster the students into globally competent professionals useful to the Society.”

Mission

The department of CSE (Emerging Technologies) is committed to:

- To offer highest Professional and Academic Standards in terms of Personal growth and satisfaction.
- Make the society as the hub of emerging technologies and thereby capture opportunities in new age technologies.
- To create a benchmark in the areas of Research, Education and Public Outreach.
- To provide students a platform where independent learning and scientific study are encouraged with emphasis on latest engineering techniques.

QUALITY POLICY

- To pursue continual improvement of teaching learning process of Undergraduate and Post Graduate programs in Engineering & Management vigorously.
- To provide state of art infrastructure and expertise to impart the quality education and research environment to students for a complete learning experiences.
- Developing students with a disciplined and integrated personality.
- To offer quality relevant and cost effective programmes to produce engineers as per requirements of the industry need.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

- To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

- To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

- To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

- To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Engineering, the graduates will have the following Program Specific Outcomes:

1. Fundamentals and critical knowledge of the Computer System:- Able to understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. The comprehensive and Applicative knowledge of Software Development: Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3. Applications of Computing Domain & Research: Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

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, review 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 multi disciplinary 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.

COURSE OBJECTIVES:

- Learning basic concepts of Python through illustrative examples and small exercises
- To prepare students to become Familiarity with the Python programming in AI environment.
- To provide student with an academic environment aware of various AI Algorithms.
- To train Students with python programming as to comprehend, analyze, design and create AI platforms and solutions for the real life problems.

List of Experiments

1	a)	Write a program to print the multiplication table for the given number
	b)	Write a program to find factorial of a number
	c)	Write a program to check whether the given number is prime or not
2	a)	Write program to implement Simple Calculator program
	b)	Write a program to generate Calendar for the given month and year
	c)	Write a program to Illustrate Different Set Operations
3	Write a program to implement simple Chat bot	
4	a)	Write a program to remove punctuations from the given string
	b)	Write a program to sort the sentence in alphabetical order
5	Write a program to Implement of Towers of Hanoi Problem.	
6	Write a Program to Implement Breadth First Search.	
7	Write a Program to Implement Depth First Search.	
8	Write a program to implement Hill Climbing Algorithm	
9	Write a program to implement A* Algorithm.	
10	Write a program to implement Tic-Tac-Toe game.	
11	Write a program to implement Water Jug Problem	

COURSE OUTCOMES:

- Apply various AI search algorithms (uninformed, informed, heuristic, constraint satisfaction,)
- Understand the fundamentals of knowledge representation, inference
- Understand the fundamentals of theorem proving using AI tools
- Demonstrate working knowledge of reasoning in the presence of incomplete and/or uncertain information.
- Ability to apply knowledge representation, reasoning, and machine learning techniques to real-world problems

General laboratory instructions

1. Students are advised to come to the laboratory at least 5 minutes before (tothe starting time), those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab. c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

INDEX

PROGRAM-1(A)

1A) Aim: Write a program to print the multiplication table for the given number

Program:

```
# Python program to find the multiplication table (from 1 to 10) of a number input by the
 user
# Get input from the user

num = int(input("Display multiplication table of? "))

# use for loop to iterate 10 times

 for i in range(1,11):
           print(num,'x',i,'=',num*i)
```

Output:

```
Display multiplication table of? 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

PROGRAM-1(B)

1B) Aim: Write a program to find factorial of the given number

Program:

```
def recur_factorial(n):

if n == 1:
return n else:
return n*recur_factorial(n-1)
```

```
num = int(input("Enter a number: ")) # check is the number is negative
if num < 0:

print("Sorry, factorial does not exist for negative numbers") elif num == 0:
print("The factorial of 0 is 1") else:
print("The factorial of",num,"is",recur_factorial(num))
```

Output:

```
Enter a number: 5

The factorial of 5 is 120
```

PROGRAM-1(C)

1C) Aim: Write a program to check whether the given number is prime or not?

Program:

```
# Python program to check if the input number is prime or not

num = int(input("Enter a number: "))

# prime numbers are greater than 1 if num > 1:

 rem=1
 for i in range(2,num):
 rem=num%i

 if rem == 0:

 break

if rem == 0:
 print(num,"is not a prime number")
 else:
 print(num,"is a prime number")
else:
 print(num,"is not a prime number")
```

Output:

```
Enter a number: 5
5 is a prime number
```

Exercise Programs

1. Write the steps to install SWI-PROLOG .
2. Write the commands to create a directory, change a directory, the current directory
3. List the commands to load a prolog file

PROGRAM-2(A)

2A) Write a program to implement Simple Calculator program

Program:

```
# Program make a simple calculator that can add, subtract, multiply and divide using

functions

 # define functions

def add(x, y):

 """This function adds two numbers"""

return x + y

 def subtract(x, y):

 ""This function subtracts two numbers"""

 return x - y

 def multiply(x, y):

 """This function multiplies two numbers"""

 return x * y

def divide(x, y):

 """This function divides two numbers"""

 return x / y

 # take input from the user
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
 choice = input("Enter choice(1/2/3/4):")
  num1 = int(input("Enter first number: "))
 num2 = int(input("Enter second number: "))
  if choice == '1':
 print(num1,"+",num2,"=", add(num1,num2))
 elif choice == '2':
 print(num1,"-",num2,"=", subtract(num1,num2))
 elif choice == '3':
 print(num1,"*",num2,"=", multiply(num1,num2))
elif choice == '4':
 print(num1,"/",num2,"=", divide(num1,num2))
else:
 print("Invalid input")
```

Output:

```
Select operation.

1.Add

2.Subtract

3.Multiply

4.Divide

Enter choice(1/2/3/4):1

Enter first number: 21

Enter second number: 22

21 + 22 = 43
```

PROGRAM-2(B)

2B) Aim: Write a program to generate Calendar for the given month and year

Program:

```
# Python program to display calendar of given month of the year # import
module
import calendar
yy = 2014
mm = 11
```

```
# To ask month and year from the user# Python program to display calendar of given
month of the year
# import module import
calendar
 yy = 2014
mm = 11

# To ask month and year from the user
 # yy = int(input("Enter year: "))
# mm = int(input("Enter month: "))
# display the calendar
print(calendar.month(yy, mm))
# yy = int(input("Enter year: "))

# mm = int(input("Enter month: "))
# display the calendar print(calendar.month(yy, mm))
```

Output: November 2014

```
Mo Tu We Th Fr Sa Su

                    1    2

 3   4    5    6  7  8    9

10  11  12   13  14  15   16

17  18  19   20  21  22    23

24  25  26   27  28  29   30
```

PROGRAM-2(C)

2C) Aim: Write a program to Illustrate Different Set Operations?

Program:

```
# Program to perform different set operations like in
mathematics # define three sets

E = {0, 2, 4, 6, 8};
N = {1, 2, 3, 4, 5};
# set union
```

```
print("Union of E and N is",E | N)
# set intersection
print("Intersection of E and N is",E & N)

# set difference
print("Difference of E and N is",E - N)

# set symmetric difference
print("Symmetric difference of E and N is",E ^ N)
```

Output:

```
Union of E and N is {0, 1, 2, 3, 4, 5, 6, 8}
Intersection of E and N is {2, 4}
Difference of E and N is {0, 8, 6}
Symmetric difference of E and N is {0, 1, 3, 5, 6, 8}
```

Viva Questions:

1. **Define function with syntax?**
2. **Define module? Differentiate between built-in and user define modules?**
3. **Write the syntax for filter, map, reduce functions?**

Exercise Programs

1. Write a program (likes.pl)in Prolog that queries a database that contains data on food items members are liking.
2. Write a prolog code that constructs a family tree and queries a member present in the family.

PROGRAM-3

Aim:

Write a program to implement simple Chat bot

Program:

```
print("Simple Question and Answering Program")
print("=====================================")
print(" You may ask any one of these questions")
print("Hi")
print("How are you?")
print("Are you working?")
print("What is your name?")
print("what did you do yesterday?")
print("Quit")
while True:
question = input("Enter one question from above list:")
question = question.lower()
if question in ['hi']:
print("Hello")
elif question in ['how are you?','how do you do?']:
print("I am fine")
elif question in ['are you working?','are you doing any job?']:
print("yes. I'am working in MRCET")
elif question in ['what is your name?']:
print("My name is Emilia")
name=input("Enter your name?")
print("Nice name and Nice meeting you",name)
elif question in ['what did you do yesterday?']:
print("I saw Bahubali 5 times")
```

```
elif question in ['quit']:

break

else:

print("I don't understand what you said")
```

Output:

```
Simple Question and Answering Program

==============================

You may ask any one of these questions

Hi

How are you?

Are you working?

What is your name?

what did you do

yesterday? Quit

Enter one question from above list:hi

Hello

Enter one question from above list:how are

you? I am fine

Enter one question from above list:are you working?

yes. I'am working in MRCET

Enter one question from above list:what is your name?

My name is Emilia

Enter your name?sachin
Nice name and Nice meeting you sachin

Enter one question from above list:quit
```

Viva Questions:

1. **What is Natural Processing Language or NLP?**
2. **What is a chatbot?**
3. **What languages and technologies should a chatbot developer be to build chatbots?**

Exercise Programs

1. Write a code in prolog to find the factorial of a given number.

PROGRAM-4(A)

Aim: Write a program to remove punctuations from the given string

Program:

```
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
my_str = "Hello!!!, he said ---and went."
# To take input from the user
# my_str = input("Enter a string: ")
# remove punctuation from the string
no_punct = ""
for char in my_str:
if char not in punctuations:
no_punct = no_punct + char
# display the unpunctuated string
print(no_punct)
```

Output:

```
Hello he said and went
```

PROGRAM-4(B)

Aim: 4(B). Write a program to sort the sentence in alphabetical order

Program:

```
# Program to sort alphabetically the words form a string provided by the
user # change this value for a different result
my_str = "Hello this Is an Example With cased letters"
# uncomment to take input from the user
```

```
#my_str = input("Enter a string: ")

# breakdown the string into a list of words

 words = my_str.split()

#print(words)

# sort the list

words.sort()

# display the sorted words

print("The sorted words are:")

for word in words:

print(word)
```

Output:

```
The sorted words are:

Example

Hello

Is

With

an

cased

letter

s this
```

Viva Questions:

1. **List out String functions in Python?**
2. **List out the methods of list?**
3. **What is list indexing and slicing with an example?**

Exercise Programs

1. Write a prolog code that reverse a string.

2. Write a prolog code that checks a palindrome.

PROGRAM-5

Aim: Write a Program to Implement of Towers of Hanoi Problem.

Program:

```
# Recursive Python function to solve tower of hanoi
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
        if n == 0:
                return
        TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)

        print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
        TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)
# Driver code
n = 4
TowerOfHanoi(n, 'A', 'C', 'B') # A, C, B are the name of rods
```

Output:

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

Viva Questions:

1. **What is AI? What are the Applications of AI.**
2. **What is Problem Space?**
3. **What are the Properties of Environment?**

Exercise Programs:

1. Write a Program to Implement of Towers of Hanoi Problem in Prolog

PROGRAM-6

Aim:

Write a Program to Implement Breadth First Search.

Program:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

Output:

```
Following is the Breadth-First Search
5 3 7 2 4 8
```

Viva Questions:

1. **Differences between Informed and Uninformed Search.**

2. **What are the Properties Of Search Algorithms.**
3. **What is Breadth-First-Search.**

Exercise Programs

1. Write a code in prolog to find perform breadth first search.

PROGRAM-7

Aim:

Write a Program to Implement Depth First Search.

Program:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}


visited = set() # Set to keep track of visited nodes of graph.
def dfs(visited, graph, node):

 #function for dfs if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
```

# Driver Code

```
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

Output:

```
Following is the Depth-First Search
5
3
2
4
```

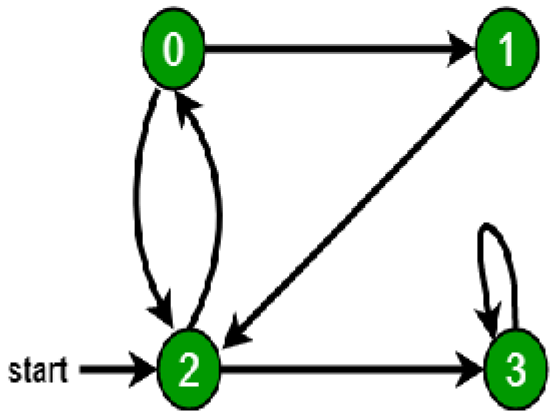
8
7

Viva Questions:

1. **What is Depth-First- Search.**
2. **Differences between BFS and DFS.**
3. **Write the Applications of Stack and Queue.**

Exercise Program

Write a Program to Implement Depth First Search for given graph



PROGRAM-8

Aim:

Write a program to implement Hill Climbing Algorithm

Program:

```
import random

def randomSolution(tsp):

    cities = list(range(len(tsp)))

    solution = []

    for i in range(len(tsp)):

        randomCity = cities[random.randint(0, len(cities) - 1)]

        solution.append(randomCity)

        cities.remove(randomCity)

    return solution

def routeLength(tsp, solution):

    routeLength = 0

    for i in range(len(solution)):

        routeLength += tsp[solution[i - 1]][solution[i]]

    return routeLength

def getNeighbours(solution):

    neighbours = []

    for i in range(len(solution)):

        for j in range(i + 1, len(solution)):
        neighbour = solution.copy()

        neighbour[i] = solution[j]

        neighbour[j] = solution[i]

        neighbours.append(neighbour)

    return neighbours

def getBestNeighbour(tsp, neighbours):

    bestRouteLength = routeLength(tsp, neighbours[0])

    bestNeighbour = neighbours[0]

    for neighbour in neighbours:

        currentRouteLength = routeLength(tsp, neighbour)

        if currentRouteLength < bestRouteLength:

            bestRouteLength =

            currentRouteLength bestNeighbour =

            neighbour

    return bestNeighbour, bestRouteLength


def hillClimbing(tsp):

    currentSolution = randomSolution(tsp)

    currentRouteLength = routeLength(tsp, currentSolution)

    neighbours = getNeighbours(currentSolution)

    bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp, neighbours)

    while bestNeighbourRouteLength < currentRouteLength:

        currentSolution = bestNeighbour

        currentRouteLength = bestNeighbourRouteLength
        neighbours = getNeighbours(currentSolution)

        bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp, neighbours)

    return currentSolution, currentRouteLength
```

```
def main():
    tsp = [
        [0, 400, 500, 300],
        [400, 0, 300, 500],
        [500, 300, 0, 400],
        [300, 500, 400, 0]
    ]
    print(hillClimbing(tsp))

if__name___== "_main_":
    main()
```

Output:

```
([1, 0, 3, 2], 1400)
```

Viva Questions:

1. **What is Heuristic Search.**
2. **List the problems of Hill Climbing.**
3. **Define IDDFS .**

Exercise Program

1. Implement hill climbing algorithm in Prolog

PROGRAM-9

Aim:

Write a program to implement A* Algorithm

Program:

```
from collections import deque

class Graph:
    def init (self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]

    # This is heuristic function which is having equal values for all nodes
    def h(self, n):
         H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }
        return H[n]

    def a_star_algorithm(self, start, stop):
        # In this open_lst is a lisy of nodes which have been visited, but who's
        # neighbours haven't all been always inspected, It starts off with the
  start #node
        # And closed_lst is a list of nodes which have been visited
        # and who's neighbors have been always inspected
        open_lst = set([start])
        closed_lst = set([])

        # poo has present distances from start to all other
```

```
nodes # the default value is +infinity
poo = {}
 poo[start] = 0

# par contains an adjac mapping of all nodes
par = {}
par[start] = start

while len(open_lst) > 0:
    n = None


    # it will find a node with the lowest value of f() -
    for v in open_lst:
        if n == None or poo[v] + self.h(v) < poo[n] + self.h(n): n
            = v;

    if n == None:
        print('Path does not exist!')
        return None

    # if the current node is the stop
     # then we start again from start if n == stop:
        reconst_path = []

        while par[n] != n:
            reconst_path.append(n) n = par[n]
        reconst_path.append(start)

        reconst_path.reverse()

        print('Path found: {}'.format(reconst_path))
        return reconst_path

    # for all the neighbors of the current node do
    for (m, weight) in self.get_neighbors(n):
      # if the current node is not presentin both open_lst and
        closed_lst
      # add it to open_lst and note n as it's par
        if m not in open_lst and m not in closed_lst:
            open_lst.add(m)
            par[m] = n
            poo[m] = poo[n] + weight

        # otherwise, check if it's quicker to first visit n, then m
        # and if it is, update par data and poo data
        # and if the node was in the closed_lst, move it to open_lst
        else:
            if poo[m] > poo[n] + weight:
                poo[m] = poo[n] + weight
                par[m] = n

                if m in closed_lst:
                    closed_lst.remove(m)
                    open_lst.add(m)
                    # remove n from the open_lst, and add it to
                    closed_lst # because all of his neighbors were
                    inspected open_lst.remove(n)
                    closed_lst.add(n)

                print('Path does not exist!')
                return None
        adjac_lis = {
            'A': [('B', 1), ('C', 3), ('D', 7)],
            'B': [('D', 5)],
            'C': [('D', 12)]
        }
        graph1 = Graph(adjac_lis)
        graph1.a_star_algorithm('A', 'D')
```

Output:

```
Path found: ['A', 'B', 'D']
```

Viva Questions:

1. **What is Best First Search?**
2. **Explain Heuristic Function?**
3. **Explain A* Search.**

   .

Exercise Program
1. Implement A* in Prolog

PROGRAM-10

Aim:

Write a program to implement Tic-Tac-Toe game.

Program:

```
import os
import time

board = [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ']
player = 1

########win Flags##########
Win = 1
Draw = -1
Running = 0
Stop = 1
############################
Game = Running
Mark = 'X'

#This Function Draws Game Board
def DrawBoard():
    print(" %c | %c | %c " % (board[1],board[2],board[3]))
    print("___|___|___")
    print(" %c | %c | %c " % (board[4],board[5],board[6]))
    print("___|___|___")
    print(" %c | %c | %c " % (board[7],board[8],board[9]))
    print(" | | ")

#This Function Checks position is empty or not
def CheckPosition(x):
    if(board[x] == ' '):
        return True
    else:
        return False

#This Function Checks player has won or not
def CheckWin():
    global Game
    #Horizontal winning condition
    if(board[1] == board[2] and board[2] == board[3] and board[1] != ' '):
        Game = Win
    elif(board[4] == board[5] and board[5] == board[6] and board[4] != ' '):
        Game = Win
    elif(board[7] == board[8] and board[8] == board[9] and board[7] != ' '):
        Game = Win


    #Vertical Winning Condition
    elif(board[1] == board[4] and board[4] == board[7] and board[1] != ' '):
        Game = Win
    elif(board[2] == board[5] and board[5] == board[8] and board[2] != ' '):
        Game = Win
    elif(board[3] == board[6] and board[6] == board[9] and board[3] != ' '):
        Game=Win
    #Diagonal Winning Condition
    elif(board[1] == board[5] and board[5] == board[9] and board[5] != ' '):
        Game = Win
    elif(board[3] == board[5] and board[5] == board[7] and board[5] != ' '):
        Game=Win
    #Match Tie or Draw Condition
    elif(board[1]!=' ' and board[2]!=' ' and board[3]!=' ' and board[4]!=' ' and board[5]!=' ' and
board[6]!=' ' and board[7]!=' ' and board[8]!=' ' and board[9]!=' '):
        Game=Draw
    else:
        Game=Running
print("Tic-Tac-Toe Game Designed By Sourabh Somani")
print("Player 1 [X] --- Player 2 [O]\n")
print()
print()
print("Please Wait...")
time.sleep(3)
while(Game == Running):
    os.system('cls')
    DrawBoard()
    if(player % 2 != 0):
        print("Player 1's chance")
```

```
            Mark = 'X'
        else:
            print("Player 2's chance")
            Mark = 'O'
        choice = int(input("Enter the position between [1-9] where you want to mark :
        "))
        if(CheckPosition(choice)):
            board[choice] = Mark
            player+=1 CheckWin()

    os.system('cls')
    DrawBoard()
    if(Game==Draw):
        print("Game Draw")
    elif(Game==Win):
        player-=1
        if(player%2!=0):
            print("Player 1 Won")
        else:
            print("Player 2 Won")
```

Output:

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 0 0]
 [0 1 0]
 [0 0 0]]
Board after 2 move
[[0 0 0]
 [0 1 0]
 [0 2 0]]
Board after 3 move
[[0 0 0]
 [0 1 0]
 [1 2 0]]
Board after 4 move
[[0 0 2]
 [0 1 0]
 [1 2 0]]
Board after 5 move
[[0 0 2]
```

```
 [0 1 0]
 [1 2 1]]
Board after 6 move
[[0 0 2]
 [0 1 2]
 [1 2 1]]
Board after 7 move
[[0 0 2]
 [1 1 2]
 [1 2 1]]
Board after 8 move
[[2 0 2]
 [1 1 2]
 [1 2 1]]
Board after 9 move
[[2 1 2]
 [1 1 2]
 [1 2 1]]
Winner is: -1
```

Viva Questions:

1. **What is Stochastic Search?**
2. **What is MIN-MAX Search?**
3. **What is Alpha-Beta Pruning.**

Exercise Program
1. Implement Tic Tac Toe game in Prolog

PROGRAM-11

Aim:

Write a program to implement Water Jug Problem

Program:

```
# This function is used to initialize the
# dictionary elements with a default value. from collections import defaultdict

# jug1 and jug2 contain the value
# for max capacity in respective jugs
# and aim is the amount of water to be measured.
jug1, jug2, aim = 4, 3, 2
```

```
# Initialize dictionary with # default value as false.
visited = defaultdict(lambda: False)

# Recursive function which prints the # intermediate steps to reach
the final # solution and return boolean value
# (True if solution is possible, otherwise False).
# amt1 and amt2 are the amount of water
present # in both jugs at a certain point of time.
def waterJugSolver(amt1, amt2):

        # Checks for our goal and
        # returns true if
        achieved.
        if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
                print(amt1, amt2)
                return True


        # Checks if we have already visited the
        # combination or not. If not, then it proceeds further.
        if visited[(amt1, amt2)] == False:
                print(amt1, amt2)

                # Changes the boolean value of # the combination as it is visited.
                visited[(amt1, amt2)] = True

                # Check for all the 6 possibilities and
                # see if a solution is found in any one of them.
                return (waterJugSolver(0, amt2) or
                waterJugSolver(amt1, 0) or
                waterJugSolver(jug1, amt2) or
                waterJugSolver(amt1, jug2) or
                waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                amt2 - min(amt2, (jug1-amt1)))
                or waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                amt2 + min(amt1, (jug2-amt2))))

        # Return False if the combination is
        # already visited to avoid repetition otherwise
        # recursion will enter an infinite loop.
        else:
                return False

print("Steps: ")
```

```
# Call the function and pass the
# initial amount of water present in both jugs.
waterJugSolver(0, 0)
```

Output:

```
Steps:

 0  0
 4  0
 4  3
 0  3
 3  0
 3  3
 4  2
 0  2

True
```

Viva Questions:

1. **What is AO* Search?**

Exercise Program
1. Implement Water Jug Problem in Prolog