A T M E
College of Engineering

COLLEGE OF ENGINEERING
13thKM Stone, Bannur Road, Mysore - 560 028

DEPARTMENT OF COMPUTER SCIENCE AND NGINEERING–AI & ML

(ACADEMIC YEAR 2023-24)

LABORATORY MANUAL

SUBJECT: ARTIFICIAL INTELLIGENCE LAB

SUB CODE: BAD402

SEMESTER:IV SCHEME: 2022

Prepared By	Verified By	Approved by
Ms.GEETHA.B	**Dr. HUSSANA JOHAR R B**	**Dr. M S SUNITHA PATEL**
INSTRUCTOR	**FACULTIES CO_ORDINATORS**	**HOD,CSE-AI &ML**

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow’s society.
- To strive to attain ever-higher benchmarks of educational excellence.

Department of Computer Science & Engineering-AI & ML

Vision of the Department

- To develop highly talented individuals in Computer Science and Engineering to deal withreal world challenges in industry, education, research and society.

Mission of the Department

- To inculcate professional behavior, strong ethical values, innovative research capabilities and leadership abilities in the young minds & to provide a teaching environment that emphasizes depth, originality and critical thinking.

- Motivate students to put their thoughts and ideas adoptable by industry or to pursue higher studies leading to research.

Program outcomes (POs)
Engineering Graduates will be able to:

PO1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. **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.

PO3. **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.

PO4. **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.

PO5. **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.

PO6. **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.

PO7. **Environment and sustainability**: Understand the impact of the professionalengineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. **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.

PO11. **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.

PO12. 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 Outcomes (PSOs)

PSO1: Ability to apply skills in the field of algorithms, database design, web design, cloud computing and data analytics.

PSO2: Apply knowledge in the field of computer networks for building network and internetbased applications

Program Educational Objectives (PEOs):

1. Empower students with a strong basis in the mathematical, scientific and engineering fundamentals to solve computational problems and to prepare them for employment, higher learning and R&D.
2. Gain technical knowledge, skills and awareness of current technologies of computer science engineering and to develop an ability to design and provide novel engineering solutions for software/hardware problems through entrepreneurial skills.
3. Exposure to emerging technologies and work in teams on interdisciplinary projects with effective communication skills and leadership qualities.
4. Ability to function ethically and responsibly in a rapidly changing environment by applying innovative ideas in the latest technology, to become effective professionals in Computer Science to bear a life-long career in relate dare as.

ARTIFICIAL INTELLIGENCE		Semester	IV
Course Code	BAD402	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours Theory + 8-10 Lab slots	Total Marks	100
Credits	04	Exam Hours	
Examination nature (SEE)	Theory/		

PRACTICAL COMPONENT OF IPCC PROGRAMS NEED TO BE IMPLEMENTED IN PYTHON

Sl.NO	Experiments
1	Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem
2	Implement and Demonstrate Best First Search Algorithm on Missionaries-Cannibals Problems using Python
3	Implement A* Search algorithm
4	Implement AO* Search algorithm
5	Solve 8-Queens Problem with suitable assumptions
6	Implementation of TSP using heuristic approach
7	Implementation of the problem solving strategies: either using Forward Chaining or Backward Chaining
8	Implement resolution principle on FOPL related problems
9	Implement Tic-Tac-Toe game using Python
10	Build a bot which provides all the information related to text in search box
11	Implement any Game and demonstrate the Game playing strategies

Course outcomes (Course Skill Set):

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

CO 1: Apply knowledge of agent architecture, searching and reasoning techniques for different applications.

CO 2. Compare various Searching and Inferencing Techniques.

CO 3. Develop knowledge base sentences using propositional logic and first order logic

CO 4. Describe the concepts of quantifying uncertainty.

CO5: Use the concepts of Expert Systems to build applications

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. Theminimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed tohave satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
- 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theorycomponent of IPCC (that is for **25 marks)**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluationof the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.

CONTENTS

Introduction to Artificial Intelligence

AI is a combination of computer science, physiology, and philosophy. Al is a broad topic, consisting of differentfields, from machine vision to expert systems. John McCarthy was one of the founders of Al field who stated thatAl is the science and engineering of making intelligent machines, especially intelligent computer programs.

Different people think of Al differently and there is no unique definition. Various authors have defined Al differently as *g*iven below.

- AI is the study of mental faculties through the use computational models (Charniak and McDermott, 1985).
- The art of creating machines that perform functions which require intelligence when performed by people(Kurzweil, 1990).

- AI is a field of study that seeks to **explain and emulate intelligent behaviour** in terms of computationalprocesses (**Schalkoff, 1990**).
- AI is the study of how to make computers do things at which, at the moment, people are better (**Rich andKnight, 1991**).
- AI is the **study of the computations** that make it possible to perceive, reason, and act (**Winston, 1992**).
- AI is the branch of computer science that is concerned with the **automation of intelli*g*ent** behaviour(**Luger and Stubblefield, 1993**).

Applications

AI finds applications in almost all areas of real-life applications. Broadly speaking, business, engineering,medicine, education and manufacturing are the main areas.

- Business: financial strategies, give advice
- Engineering: check design, offer suggestions to create new product, expert systems for all
- engineering applications
- Manufacturing: assembly, inspection, and maintenance
- Medicine: monitoring, diagnosing, and prescribing
- Education: in teaching
- Fraud detection
- Object identification
- Space shuttle scheduling
- Information retrieval

General Problem Solving

Production System: It is one of the formalisms that helps Al programs to do search process more conveniently in state-space problems. This system comprises of start (initial) state(s) and goal (final) state(s) of the problem along with one or more databases consisting of suitable and necessary information for the particular task.

PS consists of number of production rules in which each *production rule* has left side that determines the applicability of the rule and a right side that describes the action to be performed if the rule is applied.

Left side of the rule is current state, whereas the right side describes the new state that is obtained from applyingthe rule. These production rules operate on the databases that change as these rules are applied.

Let us consider an example of water jug problem and formulate production rules to solve this problem.

Water Jug Problem:

Problem statement: We have two jugs, a 5-gallon (5-g) and the other 3-gallon (3-g) with no measuring marker on them. There is endless supply of water through tap. Our task is to get 4 gallon of water in the 5-g jug.
Solution: State space for this problem can be described as the set of ordered pairs of integers (X, Y) such that Xrepresents the number of gallons of water in 5-g jug and Y for 3-g jug.

1. Start state is (0,0)
2. 2. Goal state is (4, N) for any value of NS3.

The possible operations that can be used in this problem are listed as follows:

- Fill 5-g jug from the tap and empty the 5-g jug by throwing water down the drain
- Fill 3-g jug from the tap and empty the 3-g jug by throwing water down the drain
- Pour some or 3-g water from 5-g jug into the 3-g jug to make it full
- Pour some or full 3-g jug water into the 5-g jug

These operations can formally be defined as production rules as given in table.

RuleNo	**Left of rule**	**Right of rule**	**Description**
1	$(X, Y \mid X < 5)$	$(5, Y)$	Fill 5-g jug
2	$(X, Y \mid Y > 0)$	$(0, Y)$	Empty 5-g jug
3	$(X, Y \mid Y < 3)$	$(X, 3)$	Fill 3-g jug
4	$(X, Y \mid Y > 0)$	$(X, 0)$	Empty 3-g jug
5	$(X, Y \mid X + Y \leq 5 \wedge Y > 0)$	$(X + Y, 0)$	Empty 3-g into 5-g jug
6	$(X, Y \mid X + Y \leq 3 \wedge X > 0)$	$(0, X + Y)$	Empty 5-g into 3-g jug
7	$(X, Y \mid X + Y \geq 5 \wedge Y > 0)$	$(5, Y – (5 – X))$ until 5-g jug is full	Pour water from 3-g jug into 5-g jug
8	$(X, Y \mid X + Y \geq 3 \wedge X > 0)$	$(X – (3 – Y), 3)$	Pour water from 5-g jug into 3-g juguntil 3-g jug is full

Depth-First Search

In the depth-first search (DFS), we go as far down as possible into the search tree/graph before backing up and trying alternatives. It works by always generating a descendent of the most recently expanded node until some depth cut off is reached and then backtracks to next most recently expanded node and generates one of its descendants.

We can implement DFS by using two lists called OPEN and CLOSED. The OPEN list contains those states thatare to be expanded, and CLOSED list keeps track of states already expanded Here OPEN and CLOSED lists are maintained as stacks.

Missionaries and Cannibals Problem:

Problem statement: Three missionaries and three cannibals want to cross a river. There is a boat on their side ofthe river that can be used by either one or two persons. How should they use this boat to cross the river in such away that cannibals never outnumber missionaries on either side of the river? If the cannibals ever outnumber themissionaries (on either bank) then the missionaries will be eaten. How can they all cross over without anyone being eaten?

Solution: State space for this problem can be described as the set of ordered pairs of left and right banks of the river as (L, R) where each bank is represented as a list [nM, mC, B). Here n is the number of missionaries M, mis the number of cannibals C, and B represents the boat.

Let us consider another problem of 'Missionaries and Cannibals' and see how we can we solve this using production system.

1.Start state: ([3M, 3C, 1B], [OM, OC, OB]), 1B means that beat is present and OB means it is absent.

2.Any state: {{n, M, mc, _), n2M, mC, _]), with constraints/conditions at any state as nj (#0) Zm; n2 (+0)2 m2; n, + n2 = 3, m + m2 = 3; boat can be either side.

3.Goal state: ([OM, OC, OB], [3M, 3C, 1B])

It should be noted that by no means, this representation is unique. In fact, one may have number of representationsfor the same problem. **Table** consists of production rules based on the chosen representation. States on the left orright sides of river should be valid states satisfying the constraints given in (2) above.

RN	Left side of rule	→	Right side of rule
	Rules for boat going from left bank to right bank of the river		
L1	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([(n_1-2)M, m_1C, 0B], [(n_2+2)M, m_2C, 1B])$
L2	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([(n_1-1)M,(m_1-1)C,0B],[(n_2+1)M,(m_2+1)C, 1B])$
L3	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([n_1M, (m_1-2)C, 0B], [n_2M, (m_2+2)C, 1B])$
L4	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([(n_1-1)M, m_1C, 0B],[(n_2+1)M, m_2C, 1B])$
L5	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([n_1M, (m_1-1)C, 0B], [n_2M, (m_2+1)C, 1B])$
	Rules for boat coming from right bank to left bank of the river		
R1	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([(n_1+2)M, m_1C, 1B], [(n_2-2)M, m_2C, 0B])$
R2	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([(n_1+1)M,(m_1+1)C,1B],[(n_2-1)M,(m_2-1)C, 0B])$
R3	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([n_1M, (m_1+2)C, 1B], [n_2M, (m_2-2)C, 0B])$
R4	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([(n_1+1)M, m_1C,1B],[(n_2-1)M, m_2C, 0B])$
R5	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([n_1M, (m_1+1)C, 1B], [n_2M, (m_2-1)C, 0B])$

Breadth-First Search

The breadth-first search (BFS) expands all the states one step away from the start state, and then expands all states two steps from start state, then three steps, etc., until a goal state is reached. All successor states are examined at the same depth before going deeper. The BFS always gives an optimal path or solution.

This search is implemented using two lists called OPEN and CLOSED. The OPEN list contains those states thatare to be expanded and CLOSED list keeps track of states already expanded. Here OPEN list is maintained as a *queue* and CLOSED list as a *stack*.

Predicate Logic

The predicate logic is a logical extension of propositional logic, which deals with the validity, satisfiability, and unsatisfiability (inconsistency) of a formula along with the inference rules for derivation of a new formula.

Predicate calculus is the study of predicate systems; when inference rules are added to predicate calculus, itbecomes predicate logic.

First-Order Predicate Calculus

If the quantification in predicate formula is only on simple variables and not on predicates or functions then it is called *first-order predicate calculus.*

*O*n the other hand, if the quantification is over first-order predicates and functions, then it becomes *second-orderpredicate calculus.*

The first-order predicate calculus is a formal language in which a wide variety of statements are expressed. Theformulae in predicate calculus are formed using rules similar to those used in propositional calculus.

When inference rules are added to first-order predicate calculus, it becomes *first-order predicate logic* (FOL).Using inference rules, one can derive new formulae from the existing ones.

SAMPLE PROGRAM 1

(a) Write a python program to print the multiplication table for the given number

(b) Write a python program to check whether the given number is prime or not?

(c) Write a python program to find factorial of the given number?

(a)
```
num = int(input("Enter the number to print its MULTIPLICATION TABLE"))
print("The multiplication table of:",num)
for count in range(1,11):
        print(num,"X",count,"=", num*count)
```

Output:

Enter the number to print its MULTIPLICATION
TABLE5The 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

(b)
```
num=int(input("Enter the number to check for prime number"))
if(num>1):
   for i in range(2,num):
      if(num%i == 0):
         print("The number",num,"is not prime")
         break
      else:
         print(num,"is a prime number")
else:
   print(num,"is a prime number")
```

Output:

Enter the number to check for prime number 9
The number 9 is not prime
Enter the number to check for prime number 5
5 is a prime number

(a)
```
num = int(input("Enter a number: "))
factorial = 1
If num < 0:
        print(" Factorial does not exist for negative numbers")
elif num == 0:
        print("The factorial of 0 is 1")
else:
        for i in range(1,num + 1):
                factorial = factorial*i
        print("The factorial of",num,"is",factorial)
```

Output:

```
Enter a number: 5
The factorial of 5 is 120
Enter a number: 8
The factorial of 8 is 40320
```

SAMPLE PROGRAM 2

(a) Write a python program to implement List operations (Nested List, Length, Concatenation, Membership, Iteration, Indexing and Slicing)

(b) Write a python program to implement List methods (Add, Append, Extend & Delete).

a)

```
list1=[1,2,3,4]
list2=[5,6,7,8]

def length():
   length=len(list1)
   print('length of list is',length)

def concatenation():
   concat=list1+list2
   print("concatenated list is",concat)

def membership():
   print(1 in list1)
   print(2 in list1)
   print(3 not in list1)
def iteration():
      for i in list1:
       print(i)

def indexing():
i=int(input('enter any index'))
   print(list1[i],'is the number in index',i)

def slicing():
     s_index=int(input('enter starting index'))
     e_index=int(input('enter ending index'))
     print(list1[s_index:e_index])
```

```
def nestedlist()
  list1.append(list2)
  print(list1)

while(True):
   print("Menu for list operations")
   print('1.nested list')
   print('2.length')
   print('3.concatenation')
   print('4.membership')
   print('5.iteration')
   print('6.indexing')
   print('7.slicing')
   print('exit')
   choice=int(input('enter choice'))
   if choice==1:
      nestedlist()
   elif choice==2:
      length()
   elif choice==3:
      concatenation()
   elif choice==4:
      membership()
   elif choice==5:
      iteration()
   elif choice==6:
      indexing()
   elif choice==7:
      slicing()
   else:
      break
```

Output:

Menu for list operations

1. Nested list

2. Length

3. Concatenation

4. Membership

5. Iteration

6. Indexing

7. Slicing

8. Exit

Enter choice: 1

[1, 2, 3, 4, [5, 6, 7, 8]]

Enter Choice: 2

length of list is 5

Enter Choice: 3

Concatenated list is [1, 2, 3, 4, [5, 6, 7, 8], 5, 6, 7, 8]

Enter Choice: 4

True

True

False

Enter Choice: 5

1

2

3

4

[5, 6, 7, 8]

Enter Choice: 6

Enter any index: 2

3 is the number in index 2

Enter Choice: 7

Enter the starting index: 2

Enter the ending index: 5

[3, 4, [5, 6, 7, 8]]

Enter Choice: 8

Exit

b)

```
num=[1,2,3,4,5]
num2=[6,7,8,9,10]

def append():
        n=int(input("enter any number"))
        num.append(n)
    print(num)

def extend():
        num.extend(num2)
        print(num)

def delete():
        del num[int(input("enter the index of the number to be deleted from num"))]
        print(num)

def add():
        print("enter position")
        n=int(input())
        print("enter value")
        v=int(input())
        num.insert(n,v)
        print(num)

while True:
        print("menue")
        print("1.add")
        print("2.append")
        print("3.extend")
        print("4.delete")
        print("5.exit")
        print("enter the choice")

        choice=int(input())

        if choice==1:
        add()
```

```
elif choice==2:

append()
elif choice==3:
extend()

elif choice==4:
delete()
```

Output:

Menu

1. add

2. append

3. extend

4. delete

5.e xit

Enter the choice

1

Enter position

1

Enter value

9

[1, 9, 2, 3, 4, 5]

Menu

1. add

2. append

3. extend

4. delete

5. exit

Enter the choice

2

Enter any number 8

[1, 9, 2, 3, 4, 5, 8]

Menu

1. add

2. append

3.extend

4. delete

5. exit

Enter the choice

3

[1, 9, 2, 3, 4, 5, 8, 6, 7, 8, 9, 10]

Menu

1. add

2. append

3. extend

4. delete

5. exit

Enter the choice

4

Enter the index of the number to be deleted from num 1

[1, 2, 3, 4, 5, 8, 6, 7, 8, 9, 10]

Menu

1. add

2. append

3. extend

4. delete

5. exit

Enter the choice

5

SAMPLE PROGRAM 3

Write a python program to implement simple Chatbot with minimum 10 conversations

```
import spacy
from chatterbot.trainers import ListTrainer
from chatterbot import chatbot
nlp = spacy.load("en_core_web_sm")
nlp = spacy.blank("en")
chatbot = chatbot("New_Bot")

conversation = [
                "Hello",
                "Hi, there!",
                "How are you doing?",
                "That is good to hear",
                "How can you I help you":,
                "Thank you",
                "You're welcome"
            ]
```

```
conversation = [
                "Hello",
                "Hi, there!",
                "How are you doing?",
                "That is good to hear",
                "How can you I help you":,
                "Thank you",
                "You're welcome"
            ]

trainer = ListTrainer(chatbot)
trainer.train(conversation)
print("Enter 'quit' to stop")
while True:
    text_input = input ("You:")
    if text_input == 'quit':
        break
    print("Bot:", chatbot.get_response(text_input))
```

Installation Procedure for executing the program:

1. **Pip install chatterbot**
2. **Pip install chatterbot_corpus**
3. **Python –m spacy download en_core_web_sm**
4. **Python –m spacy download en**
5. **Python –m spacy download en_core_web_sm**

Output:

Chatbot : Hi.

ThereYou: Hello

Chatbot: How are you

doing?You: I'm good

Chatbot: That is good to

hearYou: I need a help

Chatbot: How can I help

you?You: I feel depressed

Chatbot: That is good to

hear You: quit

SAMPLE PROGRAM 4

Write a python program to Illustrate Different Set Operations

```
def create():
        global set1,set2
        user_input=input("Enter numbers for set1:")
        set1=set(int(item) for item in user_input.split())
        user_input2=input("Enter numbers for set2:")
        set2=set(int(item) for item in user_input2.split())

def add():
        set1.add(8)
        print("set1 after addition")
        print(set1)
        set2.add(5)
        print("set2 after addition")
        print(set2)

def remove():
        set1.remove(0)
        print("set after addition")
        print(set1)

def display():
        print("set1 is",set1)
        print("set2 is",set2)
```

```
def union():
        print("union of set1 and set2 is",set1|set2)


def intersection():
        print("Intersection of set1 and set2 is",set1&set2)

def difference():

        print("Difference of set1 and set2 is",set1-set2)
def symmetric_difference():
        print("Symmetric difference of set1 and set2 is",set1^set2)
while(True):
        print("Menu for set operations")
        print("1.Create")
        print("2.Add")
        print("3.Remove")
        print("4.Display")
        print("5.Union")
        print("6.Intersection")
        print("7.Difference")
        print("8.Symmetric Difference")
        print("9.Exit")
        choice=int(input("Make the choice"))
        if choice==1:
                create()
        elif choice==2:
                add()
        elif choice==3:
                remove()
        elif choice==4:
                display()
        elif choice==5:
                union()
        elif choice==6:
                intersection()
        elif choice==7:
                difference()
        elif choice==8:
                symmetric_difference()
        elif choice==9:

                exit()
```

Output:

Menu for set operations
1. Create
2. Add
3. Remove
4. Display
5. Union
6.Intersection
7. Difference
8. Symmetric Difference
9. Exit
Make the choice 1
Enter the values for set1: 0 2 4 6
Enter the values for set2: 1 2 3 4
Menu for set operations
1. Create
2. Add

3. Remove
4. Display
5. Union
6. Intersection
7. Difference
8. Symmetric Difference
9. Exit
Make the choice 4
Set1 is {0, 2, 4, 6}
Set2 is {1, 2, 3, 4}

Menu for set operations
1. Create
2. Add
3. Remove
4. Display
5. Union
6. Intersection
7. Difference
8. Symmetric Difference
9. Exit
Make the choice 2
Set1 after addition
{0, 2, 4, 6, 8}
Set2 after addition
{1, 2, 3, 4, 5}

Menu for set operations
1. Create
2. Add
3. Remove
4. Display

5. Union
6. Intersection
7. Difference
8. Symmetric Difference
9. Exit
Make the choice 5
Union of set1 and set2 is {1, 2, 3, 4, 5, 6, 8}

Menu for set operations
1. Create
2. Add
3. Remove
4. Display
5. Union
6. Intersection
7. Difference
8. Symmetric Difference
9. Exit
Make the choice 6
Intersection of set1 and set2 is {2, 4}

Menu for set operations
1. Create
2. Add
3. Remove
4. Display
5. Union
6. Intersection
7. Difference
8. Symmetric Difference
9. Exit
Make the choice 7
Difference of set1 and set2 is {8, 6}

Menu for set operations
1. Create
2. Add
3. Remove
4. Display
5. Union
6. Intersection
7. Difference
8. Symmetric Difference
9. Exit
Make the choice 8
Symmetric difference of set1 and set2 is {1, 3, 5, 6, 8}

Menu for set operations
1. Create
2. Add
3. Remove
4. Display
5. Union

6. Intersection
7. Difference
8. Symmetric Difference
9. Exit

Make the choice 9

Exit

SAMPLE PROGRAM 5

(a) Write a python program to implement a function that counts the number of times a string(s1) occurs in another string(s2)
(b) Write a program to illustrate Dictionary operations([],in,traversal)and methods: keys(),values(),items()

```
S='Programming in python program'
str='Programming'
count=0
sub_len=len (str)
for i in range (len(s)):
        if s[i:i+sub_len]==str:
                count+=1
print("The number of times a string(s1) occurs in string(s2) is:", count)
```

Output:

The number of times a string(s1) occurs in string(s2) is: 2

```
import sys
def create():
        global employees
        employees = {}
        for i in range(s):
                name = input("Enter employees names:")
                salary = input("Enter employees salary:")
                employees[name] = salary
                print(employees)

def presence():
        key = input("Enter the key to be searched")
        if key in employees.keys():
                print("key is present")
          else:
                print("Key is not present")

   def traversal():
          for i in employees:
                print(I,":",employees[i])

   def keys():
          print("Keys are",employees.keys())
```

```
    def values():
        print("values are",employees.values())

    def items():
        print(" items are",employees.items())

while(True):
    print("dictionary operations")
    print("1.create")
    print("2.in")
    print("3.traversal")
    print("4.keys")
    print("5.values")
    print("6.items")
    print("7.exit")
    print("enter the choice")
    choice=int(input())
    if choice==1:
        create()
    elif choice==2:
        presence()
    elif choice==3:
        traversal()
    elif choice==4:
        keys()
    elif choice==5:
        values()
    elif choice==6:
        items()
    elif choice==7:
        sys.exit()
```

Output:

Dictionary operations
1. create
2. in
3. traversal
4. keys
5. values
6. items
7. exit
Enter the choice 1
Enter employees name: likitha
Enter employees salary: 30000
{'likitha': '30000'}
Enter employees name: Rohan
Enter employees salary: 50000
{'likitha': '30000', 'Rohan': '50000'}
Enter employees name: 'Ruchitha'
Enter employees salary: 60000
{'likitha': '30000', 'Rohan': '50000', 'Ruchitha': '60000'}

Dictionary operations
1. create
2. in
3. traversal
4. keys()
5. values()
6. items()
7. exit
Enter the choice 2
Enter the key to be searched likitha
Key is present

Dictionary operations
1. create
2. in
3. traversal
4. keys()
5. values()
6. items()
7. exit
Enter the choice 3
likitha : 30000
Rohan : 50000
Ruchitha : 60000

Dictionary operations
1. create
2. in
3. traversal
4. keys()
5. values()
6. items()
7. exit
Enter the choice 4

Keys are dict_keys(['likitha', 'Rohan', 'Ruchitha'])

Dictionary operations
1. create
2. in
3. traversal
4. keys()
5. values()
6. items()
7. exit
Enter the choice 5
Values are dict_values(['30000', '50000', '60000'])

Dictionary operations
1. create
2. in
3. traversal
4. keys()

5. values()
6. items()
7. exit
Enter the choice 6
Items are dict_items([('likitha', '30000'),('Rohan', '50000'),('Ruchitha', '60000')])

Dictionary operations
1. create
2. in
3. traversal
4. keys()
5. values()
6. items()
7. exit

Enter the choice 7

PROGRAM 1

Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem.

```
import queue
import time
import random

dfsq = queue.Queue()

class node:
    def __init__(self, data):
        self.x = 0
        self.y = 0
        self.parent = data
    def printnode(self):
        print("(", self.x, ",", self.y, ")")

    def generateAllSuccessors(self,cnode):
        list1 = []
        list_rule = []

        while len(list_rule) < 8:
            rule_no = random.randint(1, 8)
            if (not rule_no in list_rule):
                list_rule.append(rule_no)
                nextnode = self.operation(cnode, rule_no)
                if nextnode != None and not self.IsNodeInlist(nextnode, visitednodelist):

                    list1.append(nextnode)
            return list1

        def operation(self, cnode, rule):
            x = cnode.x
            y = cnode.y
            if rule == 1:
                if x < maxjug1:
                    x = maxjug1
                else:
                    return None
            elif rule == 2:
```

```
    if y < maxjug2:
        y = maxjug2
    else:
        return None
elif rule == 3:
    if x > 0:
        x = 0
    else:
        return None
elif rule == 4:
    if y > 0:
        y = 0
    else:
        return None
elif rule == 5:
    if x + y >= maxjug1:
        y = y - (maxjug1 - x)
        x = maxjug1
    else:
        return None
elif rule == 6:
    if x + y >= maxjug2:
        x = x - (maxjug2 - y)
        y = maxjug2
    else:

    if x + y >= maxjug2:
        x = x - (maxjug2 - y)
        y = maxjug2
    else:
        return None
elif rule == 7:
    if x + y < maxjug1:
        x = x + y
        y = 0
    else:
        return None
elif rule == 8:
```

```
        if x + y < maxjug2:
          x = 0
          y = x + y
        else:
          return None

            if (x == cnode.x and y == cnode.y):
                return None
            nextnode = node(cnode)
            nextnode.x = x
            nextnode.y = y
            nextnode.parent = cnode
            return nextnode

        def pushlist(self, list1):
            for m in list1:
                dfsq.put(m)

        def popnode(self):
            if (dfsq.empty()):
                return None
            else:
                return dfsq.get()

        def isGoalNode(self, cnode, gnode):
            if (cnode.x == gnode.x and cnode.y == gnode.y):
                return True
            return False
    def dfsMain(self,initialNode, GoalNode):
        dfsq.put(initialNode)
        while not dfsq.empty():
            visited_node = self.popnode()
            print("Pop node:")
            visited_node.printnode()
            if self.isGoalNode(visited_node, GoalNode):
                return visited_node
            successor_nodes = self.generateAllSuccessors(visited_node)
            self.pushlist(successor_nodes)
```

```
    return None

def IsNodeInlist(self, cnode, list1):
    for m in list1:
        if (cnode.x == m.x and cnode.y == m.y):


                return True
            return False
        def printpath(self, cnode):
            temp = cnode
            #list2 = []
            while (temp != None):
                list2.append(temp)
                temp = temp.parent
            list2.reverse()
            for i in list2:
                i.printnode()
            print("Path Cost:", len(list2))
    if __name__ == '__main__':
        list2 = []
        visitednodelist = []
        maxjug1 = int(input("Enter value of maxjug1:"))
        maxjug2 = int(input("Enter value of maxjug2:"))
        initialNode = node(None)
        initialNode.x = 0
        initialNode.y = 0
        initialNode.parent = None
        GoalNode = node(None)
        GoalNode.x = int(input("Enter value of Goal in jug1:"))
        GoalNode.y = 0
        GoalNode.parent = None
        start_time = time.time()
        node1 = node(None)
        solutionNode = node1.dfsMain(initialNode, GoalNode)
        end_time = time.time()
        if (solutionNode != None):
            print("Solution can Found:")
```

```
else:
    print("Solution can't be found.")

    GoalNode.x = int(input("Enter value of Goal in jug1:"))
    GoalNode.y = 0
    GoalNode.parent = None
    start_time = time.time()
    node1 = node(None)
    solutionNode = node1.dfsMain(initialNode, GoalNode)
    end_time = time.time()
    if (solutionNode != None):
        print("Solution can Found:")
    else:
        print("Solution can't be found.")
    node1.printpath(solutionNode)
    diff = end_time - start_time
    print("Execution Time:", diff * 1000, "ms")
```

Output:
Enter the value of Maxjug1: 5
Enter the value of Maxjug2: 3
Enter value of goal in jug1: 4
Solution can Found:
(0 , 0)
(0 , 3)
(3 , 0)
(3 , 3)
(5 , 1)
(0 , 1)
(1 , 0)
(1 , 3)
(4 , 0)
Path Cost: 9
Execution Time: 70518.57161521912 ms

PROGRAM 2

Implement and Demonstrate Best First Search Algorithm on any AI problem

```
import copy
class CoastState:
   def __init__(self, c, m):
      self.cannibals = c
      self.missionaries = m
# This is an intermediate state of Coast where the missionaries have to outnumber the cannibals

   def valid_coast(self):
      if self.missionaries >= self.cannibals or self.missionaries == 0:
         return True
      else:
         return  False
   def goal_coast(self):
      if self.cannibals == 3 and self.missionaries == 3:
         return True
      else:
         return False
class GameState:
   def __init__(self, data):
      self.data = data
      self.parent = None

# Creating the Search Tree
   def building_tree(self):
      children = []
      coast = ""
      across_coast = ""
      temp = copy.deepcopy(self.data)

      if self.data["boat"] == "left":
         coast =  "left"
         across_coast = "right"
        elif self.data["boat"] == "right":
           coast = "right"
           across_coast = "left"

        # MOVING 2 CANNIBALS (CC)
         if temp[coast].cannibals >= 2:
```

```
temp[coast].cannibals = temp[coast].cannibals - 2
temp[across_coast].cannibals = temp[across_coast].cannibals + 2
temp["boat"] = across_coast
if temp[coast].valid_coast() and temp[across_coast].valid_coast():
  child = GameState(temp)
  child.parent = self
  children.append(child)

 temp = copy.deepcopy(self.data)
 # MOVING 1 CANNIBAL (C)
 if temp[coast].cannibals >= 1:
   temp[coast].cannibals = temp[coast].cannibals - 1
   temp[across_coast].cannibals = temp[across_coast].cannibals + 1
   temp["boat"] = across_coast
   if temp[coast].valid_coast() and temp[across_coast].valid_coast():
     child = GameState(temp)
     child.parent = self
     children.append(child)

           child = GameState(temp)
           child.parent = self
           children.append(child)
       temp = copy.deepcopy(self.data)

       # MOVING 1 CANNIBAL AND 1 MISSIONARY (CM && MM)
       if temp[coast].missionaries >= 1 and temp[coast].cannibals >= 1:
         temp[coast].missionaries = temp[coast].missionaries - 1
         temp[across_coast].missionaries = temp[across_coast].missionaries + 1
         temp[coast].cannibals = temp[coast].cannibals - 1
         temp[across_coast].cannibals = temp[across_coast].cannibals + 1
         temp["boat"] = across_coast
         if temp[coast].valid_coast() and temp[across_coast].valid_coast():
           child = GameState(temp)
           child.parent = self
           children.append(child)
       return children

     def breadth_first_search(self):
       left = CoastState(3, 3)
       right = CoastState(0, 0)
```

```
        root_data = {"left": left, "right": right, "boat": "left"}
        explored = []
        nodes = []
        path = []
        nodes.append(GameState(root_data))
        while len(nodes) > 0:
            g = nodes.pop(0)
            explored.append(g)
            if g.data["right"].goal_coast():
                path.append(g)
                return g
            else:
                next_children = g.building_tree()
                for x in next_children:
                    if (x not in nodes) or (x not in explored):
                        nodes.append(x)
        return None

    def print_path(self, g):
        path = [g]
        while g.parent:
            g = g.parent
            path.append(g)
        print(" " + "Left Side" + " " + "Right Side" + " " + "Boat ")
        print(" Cannibals" + " Missionaries" + " " + "Cannibals" + " Missionaries" + "Boat Position")
        counter = 0
        for p in reversed(path):
            print("State " + str(counter) + " Left C: " + str(p.data["left"].cannibals) + ". Left M: "
+str(p.data["left"].missionaries) + ". | Right C: " + str(p.data["right"].cannibals) + ". Right M: " +
str(p.data["right"].missionaries) + ". | Boat: " + str(p.data["boat"]))
            counter = counter + 1
        print("End of Path!")
    def main(self):
        solution = self.breadth_first_search()
        print("Missionaries and Cannibals AI Problem Solution using Breath - First Search:")
        self.print_path(solution)
if __name__ == "_main_":
    mc = GameState(None)
    mc.main()
```

Output:

Missionaries and Cannibals AI Problem Solution using Breath - First Search:

Left Side Right Side Boat

Cannibals Missionaries Cannibals MissionariesBoat Position

State 0 Left C: 3. Left M: 3. | Right C: 0. Right M: 0. | Boat: left

State 1 Left C: 1. Left M: 3. | Right C: 2. Right M: 0. | Boat: right

State 2 Left C: 2. Left M: 3. | Right C: 1. Right M: 0. | Boat: left

State 3 Left C: 0. Left M: 3. | Right C: 3. Right M: 0. | Boat: right

State 4 Left C: 1. Left M: 3. | Right C: 2. Right M: 0. | Boat: left

State 5 Left C: 1. Left M: 1. | Right C: 2. Right M: 2. | Boat: right

State 6 Left C: 2. Left M: 2. | Right C: 1. Right M: 1. | Boat: left

State 7 Left C: 2. Left M: 0. | Right C: 1. Right M: 3. | Boat: right

State 8 Left C: 3. Left M: 0. | Right C: 0. Right M: 3. | Boat: left

State 9 Left C: 1. Left M: 0. | Right C: 2. Right M: 3. | Boat: right

State 10 Left C: 2. Left M: 0. | Right C: 1. Right M: 3. | Boat: left

State 11 Left C: 0. Left M: 0. | Right C: 3. Right M: 3. | Boat:
rightEnd of Path!

PROGRAM 3

Implement A* Search algorithm.

```
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
```

Input:

```
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']
['A', 'B', 'D']
```

PROGRAM 4

Implement AO* Search algorithm.

```
class Graph:
  def __init__(self, graph, heuristicNodeList, startNode):
    self.graph = graph
    self.H=heuristicNodeList
    self.start=startNode
    self.parent={}
    self.status={}
    self.solutionGraph={}
  def applyAOStar(self):
    self.aoStar(self.start, False)

  def getNeighbors(self, v):
    return self.graph.get(v,'')

  def getStatus(self,v):
    return self.status.get(v,0)

  def setStatus(self,v, val):
    self.status[v]=val

def getHeuristicNodeValue(self, n):
  return self.H.get(n,0) # always return the heuristic value of a given node

def setHeuristicNodeValue(self, n, value):
  self.H[n]=value # set the revised heuristic value of a given node

def printSolution(self):
  print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE:",self.start)
  print("------------------------------------------------------------")
  print(self.solutionGraph)
  print("------------------------------------------------------------")

def computeMinimumCostChildNodes(self, v):
  minimumCost=0
  costToChildNodeListDict={}
  costToChildNodeListDict[minimumCost]=[]
```

```
flag=True
for nodeInfoTupleList in self.getNeighbors(v):
    cost=0
    nodeList=[]
    #print(nodeInfoTupleList)
    for c, weight in nodeInfoTupleList:
        cost=cost+self.getHeuristicNodeValue(c)+weight
        #print(cost)
        nodeList.append(c)
    if flag==True:
        minimumCost=cost
        costToChildNodeListDict[minimumCost]=nodeList
        flag=False
    else:
        if minimumCost>cost:
            minimumCost=cost
            costToChildNodeListDict[minimumCost]=nodeList
return minimumCost, costToChildNodeListDict[minimumCost]

def aoStar(self, v, backTracking):
    print("HEURISTIC VALUES :", self.H)
    print("SOLUTION GRAPH :", self.solutionGraph)
    print("PROCESSING NODE :", v)


print("-----------------------------------------------------------------------------------------")
if self.getStatus(v) >= 0:
    minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
    print(minimumCost, childNodeList)
    self.setHeuristicNodeValue(v, minimumCost)
    self.setStatus(v,len(childNodeList))
    solved=True
    for childNode in childNodeList:
        self.parent[childNode]=v
        if self.getStatus(childNode)!=-1:
            solved=solved & False
    if solved==True:
        self.setStatus(v,-1)
        self.solutionGraph[v]=childNodeList
```

```
            if v!=self.start:
                self.aoStar(self.parent[v], True)

            if backTracking==False:
                for childNode in childNodeList:
                    self.setStatus(childNode,0)
                    self.aoStar(childNode, False)

print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

print ("Graph - 2")
h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
graph2 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'D': [[('E', 1), ('F', 1)]]
}
G2 = Graph(graph2, h2, 'A')
G2.applyAOStar()
G2.printSolution()
```

Output:

Graph - 1

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

10 ['B', 'C']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J':

SOLUTION GRAPH : {}

PROCESSING NODE : B

----

6 ['G']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J':

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

10 ['B', 'C']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J':

SOLUTION GRAPH : {}

PROCESSING NODE : G

----

8 ['I']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J':

SOLUTION GRAPH : {}

PROCESSING NODE : B

8 ['H']

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

12 ['B', 'C']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : I

----

0 []

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': []}
PROCESSING NODE : G

----

1 ['I']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I']}
PROCESSING NODE : B

----

2 ['G']
HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A

----

6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : C

----

2 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A

----

6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : J

----

0 []
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE : C

----

1 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE : A

----

5 ['B', 'C']

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

Graph - 2

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

11 ['D']

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : D

----

10 ['E', 'F']

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

11 ['D']

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {}

PROCESSING NODE : E

----

0 []

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': []}

PROCESSING NODE : D

6 ['E', 'F']

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': []}

PROCESSING NODE : A

----

7 ['D']

HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}

SOLUTION GRAPH : {'E': []}

```
PROCESSING NODE : F
-----------------------------------------------------------------------
0 []
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': []}
PROCESSING NODE : D
-----------------------------------------------------------------------
2 ['E', 'F']
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE : A
-----------------------------------------------------------------------
3 ['D']
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A
------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
```

PROGRAM 5

Solve 8-Queens Problem with suitable assumptions

```
print ("Enter the number of queens")
N = int(input())
board = [[0]*N for _ in range(N)]
def is_attack(i, j):
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False
def N_queen(n):
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(is_attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1

                if N_queen(n-1)==True:
                    return True
                board[i][j] = 0
    return False
N_queen(N)
for i in board:
     print (i)
```

Output:

```
Enter the number of queens
8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

PROGRAM 6

Implementation of TSP using heuristic approach.

```
# Traveling Salesman Problem using Branch and Bound.
import math
maxsize = float('inf')
# Function to copy temporary solution to the final solution
def copyToFinal(curr_path):
    final_path[:N + 1] = curr_path[:]
    final_path[N] = curr_path[0]
# Function to find the minimum edge cost having an end at the vertex i
def firstMin(adj, i):
    min = maxsize
    for k in range(N):
        if adj[i][k] < min and i != k:
            min = adj[i][k]
    return min
# function to find the second minimum edge cost having an end at the vertex i
def secondMin(adj, i):
    first, second = maxsize, maxsize
    for j in range(N):
        if i == j:
            continue
        if adj[i][j] <= first:
            second = first
            first = adj[i][j]
        elif(adj[i][j] <= second and adj[i][j] != first):
            second = adj[i][j]
    return second
# function that takes as arguments:
# curr_bound -> lower bound of the root node
# curr_weight-> stores the weight of the path so far
# level-> current level while moving in the search space tree
# curr_path[] -> where the solution is being stored which would later be copied to final_path[]
def TSPRec(adj, curr_bound, curr_weight,level, curr_path, visited):
    global final_res
# base case is when we have reached level N which means we have covered all the nodes once
    if level == N:
# check if there is an edge from last vertex in path back to the first vertex
```

```
        if adj[curr_path[level - 1]][curr_path[0]] != 0:
# curr_res has the total weight of the solution we got
            curr_res = curr_weight + adj[curr_path[level - 1]][curr_path[0]]
            if curr_res < final_res:
                copyToFinal(curr_path)
                final_res = curr_res
        return
# for any other level iterate for all vertices to build the search space tree recursively
    for i in range(N):
        # Consider next vertex if it is not same (diagonal entry in adjacency matrix and not visited already)
        if (adj[curr_path[level-1]][i] != 0 and visited[i] == False):

    temp = curr_bound
    curr_weight += adj[curr_path[level - 1]][i]
    # different computation of curr_bound for level 2 from the other levels
    if level == 1:
        curr_bound -= ((firstMin(adj, curr_path[level - 1]) + firstMin(adj, i)) / 2)
    else:
        curr_bound -= ((secondMin(adj, curr_path[level - 1]) + firstMin(adj, i)) / 2)

    # curr_bound + curr_weight is the actual lower bound for the node that we have arrived on.
    # If current lower bound < final_res, we need to explore the node further
    if curr_bound + curr_weight < final_res:
        curr_path[level] = i
        visited[i] = True
        # call TSPRec for the next level
        TSPRec(adj, curr_bound, curr_weight, level + 1, curr_path, visited)
        # Else we have to prune the node by resetting all changes to curr_weight and curr_bound
        curr_weight -= adj[curr_path[level - 1]][i]
        curr_bound = temp
        # Also reset the visited array
        visited = [False] * len(visited)
        for j in range(level):
            if curr_path[j] != -1:
                visited[curr_path[j]] = True

            # This function sets up final_path
            def TSP(adj):
            # Calculate initial lower bound for the root node using the formula 1/2 * (sum of first min +
            # second min) for all edges. Also initialize the curr_path and visited array
```

```
curr_bound = 0
curr_path = [-1] * (N + 1)
visited = [False] * N
# Compute initial bound
for i in range(N):
    curr_bound += (firstMin(adj, i) + secondMin(adj, i))
    # Rounding off the lower bound to an integer
    curr_bound = math.ceil(curr_bound / 2)
# We start at vertex 1 so the first vertex in curr_path[] is 0
visited[0] = True
curr_path[0] = 0
   # Call to TSPRec for curr_weight equal to 0 and level 1
   TSPRec(adj, curr_bound, 0, 1, curr_path, visited)
   # Driver code
   # Adjacency matrix for the given graph
adj = [[0, 4, 12, 7],
     [5, 0, 0, 18],
     [11, 0, 0, 6],
     [10, 2, 3, 0]]
N = 4
# final_path[] stores the final solution i.e. the // path of the salesman.
final_path = [None] * (N + 1)
# visited[] keeps track of the already
# visited nodes in a particular path
visited = [False] * N
# Stores the final minimum weight of shortest tour.
final_res = maxsize
TSP(adj)
print("Minimum cost :", final_res)
print("Path Taken : ", end = ' ')
for i in range(N + 1):
   print(final_path[i], end = ' ')
```

Output:

Minimum cost: 25

Path Taken: 0 2 3 1 0

PROGRAM 7

Implementation of the problem solving strategies: either using Forward Chaining or Backward Chaining

```
from collections import deque
import copy
file=open(input('Enter the file name:'))
line=file.readlines()
line=list(map(lambda s: s.strip(),line)) #A lambda function can take any number of arguments,
# but can only have one expression.
R = []
for i in range(len(line)):
    k=i+1
        if line[i]=='1) Rules':
            while line[k] != '2) Facts':
                r = deque(line[k].split())
                rhs = r.popleft()
                r.append(rhs)
                R.append(list(r))
                k = k + 1
        elif line[i]=='2) Facts':
            Fact=line[k].split()
        elif line[i]=='3) Goal':
            Goal=line[k]
    #.............................................
    print('PART1.  Data')
    print(' 1)Rules')
    for i in range(len(R)):
        print(' R', i+1, ': ', end='')
        for j in range(len(R[i])-1):
            print(R[i][j], end= ' ')
        print('->', R[i][-1])
    print()
    print(' 2)Facts')
    print(' ', end='')
    for i in Fact:
        print(i,' ',end='')
    print();print()
```

```
print(' 3)Goal')
print(' ', Goal)
#.......................................................
Path=[]
Flag=[]
origin_fact = copy.deepcopy(Fact)
print('PART2. Trace')
# Set initial value
count=0
Yes = False
while Goal not in Fact and Yes==False:
 #fact When the final element is added to or when itdoesn't work evenafter finishing it.

    count += 1
    print(' ', end='')
    print('ITERATION',count)
    K=-1
    apply = False
    while K<len(R)-1 and not apply: #until it finds one applicable rule.
        K=K+1
        print(' R', K + 1, ': ', end='')
        for i, v in enumerate(R[K]): # Print Kth rule (R[K])
            if i < len(R[K]) -1:
                print(v, '', end='')
            else:
                print('->',v, end='')
        if str(K+1) in Flag: #if there is a flag
            b = Flag.index(str(K+1)) +1
            if Flag[b]==[1]:
                print(', skip, because flag1 raised')
            elif Flag[b]==[2]:
                print(', skip, because flag2 raised')
        else: #no flag
            for i, v in enumerate(R[K]): # Are all the left sides of the kth rule present?
                if i == len(R[K]) -1:
                    continue
                if v in Fact:
                    if R[K][-1] in Fact: # If the right-hand side already exists
```

```
            print(' not applied, because RHS in facts. Raise flag2')
            Flag.append(str(K + 1)); Flag.append([2])
            break
        elif v == R[K][-2]:
            apply = True
            P=K+1
            break
    else:
        print(', not applied, because of lacking ', v)
        break
if apply:
    Fact.append(R[P-1][-1])

            Flag.append(str(P)); Flag.append([1])
            Path.append(P)
            print(', apply, Raise flag1. Facts ', end='')
            for i in Fact:
                print(i,' ', end='')
            print()
        elif K== len(R)-1:
            Yes=True
print()
print('PART3. Results')
if Goal in origin_fact:
    print(' ', end='')
    print('Goal A in facts. Empty path.')
else:
    if Goal in Fact:
        print(' ',end='')
        print('1) Goal',Goal,'achieved')
        print(' ', end='')
        print('2) Path:', end='')
        for i in Path:
            print('R', i, ' ', end='')
    else:
        print('1) Goal',Goal,' not achieved')
```

Output:

Enter the file name: test.txt

PART1. Data

1)Rules

R 1 : A -> L

R 2 : L -> K

R 3 : D -> A

R 4 : D -> M

R 5 : F B -> Z

R 6 : C D -> F

R 7 : A -> D

2)Facts

A B C

3)Goal

Z

PART2. Trace

ITERATION 1

R 1 : A -> L, apply, Raise flag1. Facts A B C L

PART3. Results

1) Goal Z not achieved

ITERATION 2

R 1 : A -> L, skip, because flag1 raised

R 2 : L -> K, apply, Raise flag1. Facts A B C L K

PART3. Results

1) Goal Z not achieved

ITERATION 3

R 1 : A -> L, skip, because flag1 raised

R 2 : L -> K, skip, because flag1 raised

R 3 : D -> A, not applied, because of lacking D

R 4 : D -> M, not applied, because of lacking D

R 5 : F B -> Z, not applied, because of lacking F

R 6 : C D -> F, not applied, because of lacking D

R 7 : A -> D, apply, Raise flag1. Facts A B C L K D

PART3. Results

1) Goal Z not achieved

ITERATION 4

R 1 : A -> L, skip, because flag1 raised

R 2 : L -> K, skip, because flag1 raised

R 3 : D -> A not applied, because RHS in facts. Raise flag2

R 4 : D -> M, apply, Raise flag1. Facts A B C L K D M

PART3. Results

1) Goal Z not achieved

ITERATION 5

R 1 : A -> L, skip, because flag1 raised

R 2 : L -> K, skip, because flag1 raised

R 3 : D -> A, skip, because flag2 raised

R 4 : D -> M, skip, because flag1 raised

R 5 : F B -> Z, not applied, because of lacking F

R 6 : C D -> F, apply, Raise flag1. Facts A B C L K D M F

PART3. Results

1) Goal Z not achieved

ITERATION 6

R 1 : A -> L, skip, because flag1 raised

R 2 : L -> K, skip, because flag1 raised

R 3 : D -> A, skip, because flag2 raised

R 4 : D -> M, skip, because flag1 raised

R 5 : F B -> Z, apply, Raise flag1. Facts A B C L K D M F Z

PART3. Results

1) Goal Z achieved

2) Path: R 1 R 2 R 7 R 4 R 6 R 5

PROGRAM 8

Implementation resolution principle on FOPL related problems.

```
import time
start_time = time.time()
import re
import itertools
import collections
import copy
import queue

p=open("input5.txt","r")
data=list()
data1= p.readlines()
count=0

n=int(data1[0])
queries=list()
for i in range(1,n+1):
    queries.append(data1[i].rstrip())
k=int(data1[n+1])
kbbefore=list()
def CNF(sentence):
    temp=re.split("=>",sentence)
    temp1=temp[0].split('&')
    for i in range(0,len(temp1)):
        if temp1[i][0]=='~':
            temp1[i]=temp1[i][1:]
        else:
            temp1[i]='~'+temp1[i]
    temp2='|'.join(temp1)
    temp2=temp2+'|'+temp[1]
    return temp2
    variableArray = list("abcdefghijklmnopqrstuvwxyz")
    variableArray2 = []
    variableArray3 = []
    variableArray5 = []
    variableArray6 = []
    for eachCombination in itertools.permutations(variableArray, 2):
```

```
    variableArray2.append(eachCombination[0] + eachCombination[1])
for eachCombination in itertools.permutations(variableArray, 3):
    variableArray3.append(eachCombination[0] + eachCombination[1] + eachCombination[2])
for eachCombination in itertools.permutations(variableArray, 4):
    variableArray5.append(eachCombination[0] + eachCombination[1] + eachCombination[2]+
eachCombination[3])
for eachCombination in itertools.permutations(variableArray, 5):
    variableArray6.append(eachCombination[0] + eachCombination[1] + eachCombination[2] +
eachCombination[3] + eachCombination[4])
variableArray = variableArray + variableArray2 + variableArray3 + variableArray5 + variableArray6
capitalVariables = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
number=0

def standardizationnew(sentence):
    newsentence=list(sentence)
    i=0
    global number
    variables=collections.OrderedDict()
     positionsofvariable=collections.OrderedDict()
     lengthofsentence=len(sentence)
     for i in range(0,lengthofsentence-1):
        if(newsentence[i]==',' or newsentence[i]=='('):
            if newsentence[i+1] not in capitalVariables:
                substitution=variables.get(newsentence[i+1])
                positionsofvariable[i+1]=i+1
                if not substitution :
                    variables[newsentence[i+1]]=variableArray[number]
                    newsentence[i+1]=variableArray[number]
                    number+=1
                else:
                    newsentence[i+1]=substitution
        return "".join(newsentence)

            def insidestandardizationnew(sentence):
                lengthofsentence=len(sentence)
                newsentence=sentence
                variables=collections.OrderedDict()
                positionsofvariable=collections.OrderedDict()
```

```
global number
i=0
while i <=len(newsentence)-1 :
    if(newsentence[i]==',' or newsentence[i]=='('):
        if newsentence[i+1] not in capitalVariables:
          j=i+1
          while(newsentence[j]!=',' and newsentence[j]!=')' ):
                j+=1
          substitution=variables.get(newsentence[i+1:j])
          if not substitution :
              variables[newsentence[i+1:j]]=variableArray[number]
              newsentence=newsentence[:i+1]+variableArray[number]+newsentence[j:]
              i=i+len(variableArray[number])
              number+=1
          else:
              newsentence=newsentence[:i+1]+substitution+newsentence[j:]
              i=i+len(substitution)
    i+=1
return newsentence


def replace(sentence,theta):
    lengthofsentence=len(sentence)
    newsentence=sentence
    i=0
    while i <=len(newsentence)-1 :
     if(newsentence[i]==',' or newsentence[i]=='('):
         if newsentence[i+1] not in capitalVariables:
           j=i+1
           while(newsentence[j]!=',' and newsentence[j]!=')' ):
                 j+=1
           nstemp=newsentence[i+1:j]
           substitution=theta.get(nstemp)
           if substitution :

                     newsentence=newsentence[:i+1]+substitution+newsentence[j:]
                     i=i+len(substitution)

             i+=1

           return newsentence
```

```
repeatedsentencecheck=collections.OrderedDict()

def insidekbcheck(sentence):
    lengthofsentence=len(sentence)
    newsentence=pattern.split(sentence)
    newsentence.sort()
    newsentence="|".join(newsentence)
    global repeatedsentencecheck
    i=0
    while i <=len(newsentence)-1 :
        if(newsentence[i]==',' or newsentence[i]=='('):
            if newsentence[i+1] not in capitalVariables:
                j=i+1
                while(newsentence[j]!=',' and newsentence[j]!=')' ):
                    j+=1
                newsentence=newsentence[:i+1]+'x'+newsentence[j:]
        i+=1
    repeatflag=repeatedsentencecheck.get(newsentence)
    if repeatflag :


      return True
   repeatedsentencecheck[newsentence]=1
   return False

    for i in range(n+2,n+2+k):
        data1[i]=data1[i].replace(" ","")
        if "=>" in data1[i]:
            data1[i]=data1[i].replace(" ","")
            sentencetemp=CNF(data1[i].rstrip())
            kbbefore.append(sentencetemp)
        else:
            kbbefore.append(data1[i].rstrip())
    for i in range(0,k):
    kbbefore[i]=kbbefore[i].replace(" ","")


kb={}
pattern=re.compile("\||&|=>") #we can remove the '\|" to speed up as 'OR' doesnt come in the KB
pattern1=re.compile("[(,]")
for i in range(0,k):
```

```
        kbbefore[i]=standardizationnew(kbbefore[i])
        temp=pattern.split(kbbefore[i])
        lenoftemp=len(temp)
        for j in range(0,lenoftemp):
          clause=temp[j]
          clause=clause[:-1]
          predicate=pattern1.split(clause)
          argumentlist=predicate[1:]
          lengthofpredicate=len(predicate)-1
          if predicate[0] in kb:
            if lengthofpredicate in kb[predicate[0]]:
              kb[predicate[0]][lengthofpredicate].append([kbbefore[i],temp,j,predicate[1:]])
            else:
              kb[predicate[0]][lengthofpredicate]=[kbbefore[i],temp,j,predicate[1:]]
          else:
            kb[predicate[0]]={lengthofpredicate:[[kbbefore[i],temp,j,predicate[1:]]]}

            for qi in range(0,n):
              queries[qi]=standardizationnew(queries[qi])

            def substituevalue(paramArray, x, y):
              for index, eachVal in enumerate(paramArray):
                if eachVal == x:
                  paramArray[index] = y
              return paramArray


def unificiation(arglist1,arglist2):
  theta = collections.OrderedDict()
  for i in range(len(arglist1)):
    if arglist1[i] != arglist2[i] and (arglist1[i][0] in capitalVariables) and (arglist2[i][0] in capitalVariables):
      return []
    elif arglist1[i] == arglist2[i] and (arglist1[i][0] in capitalVariables) and (arglist2[i][0] in capitalVariables):
      if arglist1[i] not in theta.keys():
        theta[arglist1[i]] = arglist2[i]
 elif (arglist1[i][0] in capitalVariables) and not (arglist2[i][0] in capitalVariables):
   if arglist2[i] not in theta.keys():
     theta[arglist2[i]] = arglist1[i]
     arglist2 = substituevalue(arglist2, arglist2[i], arglist1[i])
 elif not (arglist1[i][0] in capitalVariables) and (arglist2[i][0] in capitalVariables):
```

```
        if arglist1[i] not in theta.keys():
            theta[arglist1[i]] = arglist2[i]
            arglist1 = substituevalue(arglist1, arglist1[i], arglist2[i])
    elif not (arglist1[i][0] in capitalVariables) and not (arglist2[i][0] in capitalVariables):
        if arglist1[i] not in theta.keys():
            theta[arglist1[i]] = arglist2[i]
            arglist1 = substituevalue(arglist1, arglist1[i], arglist2[i])
        else:
            argval=theta[arglist1[i]]
            theta[arglist2[i]]=argval
            arglist2 = substituevalue(arglist2, arglist2[i], argval)
    return [arglist1,arglist2,theta]
def resolution():
    global repeatedsentencecheck
    answer=list()
    qrno=0
    for qr in queries:
        qrno+=1
        repeatedsentencecheck.clear()
        q=queue.Queue()
        query_start=time.time()
        kbquery=copy.deepcopy(kb)
        ans=qr
        if qr[0]=='~':
            ans=qr[1:]
        else:
            ans='~'+qr
        q.put(ans)
        label:outerloop
        currentanswer="FALSE"
        counter=0
        while True:
            counter+=1
            if q.empty():
                break
            ans=q.get()
            label:outerloop1
            ansclauses=pattern.split(ans)
            lenansclauses=len(ansclauses)
```

```
                    flagmatchedwithkb=0
                    innermostflag=0
                    for ac in range(0,lenansclauses):
                        insidekbflag=0
                        ansclausestruncated=ansclauses[ac][:-1]
                        ansclausespredicate=pattern1.split(ansclausestruncated)
                        lenansclausespredicate=len(ansclausespredicate)-1
                        if ansclausespredicate[0][0]=='~':
                            anspredicatenegated=ansclausespredicate[0][1:]
                        else:
                            anspredicatenegated="~"+ansclausespredicate[0]
                        x=kbquery.get(anspredicatenegated,{}).get(lenansclausespredicate)
                        if not x:
                            continue
                        else:
                            lenofx=len(x)
                            for numofpred in range(0,lenofx):
                                insidekbflag=0
                                putinsideq=0

                                sentenceselected=x[numofpred]
thetalist=unificiation(copy.deepcopy(sentenceselected[3]),copy.deepcopy(ansclausespredicate[1:]))
    if(len(thetalist)!=0):
  for key in thetalist[2]:
      tl=thetalist[2][key]
      tl2=thetalist[2].get(tl)
          if tl2:
              thetalist[2][key]=tl2
      flagmatchedwithkb=1
      notincludedindex=sentenceselected[2]
      senclause=copy.deepcopy(sentenceselected[1])

      mergepart1=""
      del senclause[notincludedindex]
      ansclauseleft=copy.deepcopy(ansclauses)
      del ansclauseleft[ac]
      for am in range(0,len(senclause)):
          senclause[am]=replace(senclause[am],thetalist[2])
          mergepart1=mergepart1+senclause[am]+'|'
      for remain in range(0,len(ansclauseleft)):
```

```
            listansclauseleft=ansclauseleft[remain]
            ansclauseleft[remain]=replace(listansclauseleft,thetalist[2])
            if ansclauseleft[remain] not in senclause:
                mergepart1=mergepart1+ansclauseleft[remain]+'|'
        mergepart1=mergepart1[:-1]
        if mergepart1=="":
          currentanswer="TRUE"
          break
        ckbflag=insidekbcheck(mergepart1)
        if not ckbflag:
                mergepart1=insidestandardizationnew(mergepart1)
                ans=mergepart1
                temp=pattern.split(ans)
                lenoftemp=len(temp)
                for j in range(0,lenoftemp):


                        clause=temp[j]
                        clause=clause[:-1]
                        predicate=pattern1.split(clause)
                        argumentlist=predicate[1:]
                        lengthofpredicate=len(predicate)-1
                        if predicate[0] in kbquery:
                            if lengthofpredicate in kbquery[predicate[0]]:

kbquery[predicate[0]][lengthofpredicate].append([mergepart1,temp,j,argumentlist])
                            else:
                                kbquery[predicate[0]][lengthofpredicate]=[[mergepart1,temp,j,argumentlist]]
                        else:
                            kbquery[predicate[0]]={lengthofpredicate:[[mergepart1,temp,j,argumentlist]]}
                    q.put(ans)
    if(currentanswer=="TRUE"):

                            break
                if(currentanswer=="TRUE"):
                  break
                if(counter==2000 or (time.time()-query_start)>20):
                   break
            answer.append(currentanswer)
        return answer
```

```
if __name__ == '_main_':
    finalanswer=resolution()
    o=open("output.txt","w+")
    wc=0
    while(wc < n-1):
        o.write(finalanswer[wc]+"\n")
        wc+=1
    o.write(finalanswer[wc])
    o.close()
```

Output:

Input1.txt

1

Take(Alice,NSAIDs)

2

Take(x,Warfarin) => ~Take(x,NSAIDs)

Take(Alice,Warfarin)

Input2.txt

2

Alert(Bob,NSAIDs)

Alert(Bob,VitC)

5

Take(x,Warfarin) => ~Take(x,NSAIDs)

HighBP(x) => Alert(x,NSAIDs)

Take(Bob,Antacids)

Take(Bob,VitA)

HighBP(Bob)

Output2.txt

TRUE

FALSE

Input3.txt

3

Alert(Alice,VitE)

Alert(Bob,VitE)

Alert(John,VitE)

9

Migraine(x) & HighBP(x) => Take(x,Timolol)

Take(x,Warfarin) & Take(x,Timolol) => Alert(x,VitE)
Migraine(Alice)
Migraine(Bob)
HighBP(Bob)
OldAge(John)
HighBP(John)
Take(John,Timolol)
Take(Bob,Warfarin)

Output3.txt

FALSE
TRUE
FALSE

Input4.txt

1
Ancestor(Liz,Bob)
6

Input4.txt

1
Ancestor(Liz,Bob)
6
Mother(Liz,Charley)
Father(Charley,Billy)
~Mother(x,y) | Parent(x,y)
~Father(x,y) | Parent(x,y)
~Parent(x,y) | Ancestor(x,y)
Parent(x,y) & Ancestor(y,z) => Ancestor(x,z)

Output4.txt

FLASE

Input5.txt

6
F(Bob)
H(John)
~H(Alice)
~H(John)
G(Bob)
G(Tom)
14

A(x) => H(x)

D(x,y) => ~H(y)

B(x,y) & C(x,y) => A(x)

B(John,Alice)

B(John,Bob)

D(x,y) & Q(y) => C(x,y)

D(John,Alice)

Q(Bob)

D(John,Bob)

F(x) => G(x)

G(x) => H(x)

H(x) => F(x)

R(x) => H(x)

R(Tom)

Output5.txt

FALSE

TRUE

TRUE

FALSE

FALSE

TRUE

PROGRAM 9

Implementation of Two Player Tic-Tac-Toe game in Python.

''' We will make the board using dictionary in which keys will be the location (i.e : top-left,mid-right,etc.)and initialliy it's values will be empty space and then after every move we will change the value according to player's choice of move. '''

```
theBoard = {'7': ' ' , '8': ' ' , '9': ' ' ,
            '4': ' ' , '5': ' ' , '6': ' ' ,
            '1': ' ' , '2': ' ' , '3': ' ' }
board_keys = []
for key in theBoard:
    board_keys.append(key)

def printBoard(board):
    print(board['7'] + '|' + board['8'] + '|' + board['9'])
    print('-+-+-')
    print(board['4'] + '|' + board['5'] + '|' + board['6'])
    print('-+-+-')
    print(board['1'] + '|' + board['2'] + '|' + board['3'])

# Now we'll write the main function which has all the gameplay functionality.
def game():
    turn = 'X'
    count = 0
    for i in range(10):
        printBoard(theBoard)
        print("It's your turn," + turn + ".Move to which place?")

            move = input()
            if theBoard[move] == ' ':
                theBoard[move] = turn
                count += 1
            else:
                print("That place is already filled.\nMove to which place?")
                continue

        # Now we will check if player X or O has won,for every move after 5 moves.
            if count >= 5:
```

```
if theBoard['7'] == theBoard['8'] == theBoard['9'] != ' ': # across the top
  printBoard(theBoard)
  print("\nGame Over.\n")
      print("\nGame Over.\n")
      print(" **** " +turn + " won. ****")
      break
    elif theBoard['4'] == theBoard['5'] == theBoard['6'] != ' ': # across the middle
      printBoard(theBoard)
      print("\nGame Over.\n")
      print(" **** " +turn + " won. ****")
      break
    elif theBoard['1'] == theBoard['2'] == theBoard['3'] != ' ': # across the bottom
      printBoard(theBoard)
      print("\nGame Over.\n")
      print(" **** " +turn + " won. ****")
      break
    elif theBoard['1'] == theBoard['4'] == theBoard['7'] != ' ': # down the left side
      printBoard(theBoard)
      print("\nGame Over.\n")
      print(" **** " +turn + " won. ****")
      break
    elif theBoard['2'] == theBoard['5'] == theBoard['8'] != ' ': # down the middle
      printBoard(theBoard)
      print("\nGame Over.\n")
      print(" **** " +turn + " won. ****")
      break
    elif theBoard['3'] == theBoard['6'] == theBoard['9'] != ' ': # down the right side
      printBoard(theBoard)
      print("\nGame Over.\n")

        print("\nGame Over.\n")
        print(" **** " +turn + " won. ****")
        break
      elif theBoard['7'] == theBoard['5'] == theBoard['3'] != ' ': # diagonal
        printBoard(theBoard)
        print("\nGame Over.\n")
        print(" **** " +turn + " won. ****")
        break
```

```
elif theBoard['1'] == theBoard['5'] == theBoard['9'] != ' ': # diagonal
    printBoard(theBoard)
    print("\nGame Over.\n")

  print(" **** " +turn + " won. ****")
  break

        # If neither X nor O wins and the board is full, we'll declare the result as 'tie'.
      if count == 9:
        print("\nGame Over.\n")
        print("It's a Tie!!")

        # Now we have to change the player after every move.
      if turn =='X':
        turn = 'O'
      else:
        turn = 'X'

      # Now we will ask if player wants to restart the game or not.
    restart = input("Do want to play Again?(y/n)")
    if restart == "y" or restart == "Y":
      for key in board_keys:
        theBoard[key] = " "
      game()

  if__name__== "_main_":
    game()
```

Output:

```
 | |
-+-+-
 | |
-+-+-
 | |
It's your turn,X.Move to which place?
4
 | |
-+-+-
X| |
-+-+-
```

```
 | |
It's your turn,O.Move to which place?

3
 | |
-+-+-
X| |
-+-+-
 | |O
It's your turn,X.Move to which place?
5
 | |
-+-+-
X|X|
-+-+-
 | |O
It's your turn,O.Move to which place?
2
 | |
-+-+-
X|X|
-+-+-
 |O|O
It's your turn,X.Move to which place?
6
 | |
-+-+-
X|X|X
-+-+-
 |O|O

Game Over.

 **** X won. ****
Do want to play Again?(y/n)y
 | |
-+-+-
 | |
-+-+-
 | |
```

```
It's your turn,X.Move to which place?
6
 | |
-+-+-
 | |X
-+-+-
 | |

It's your turn,O.Move to which place?
3
 | |
-+-+-
 | |X
-+-+-
 | |O
It's your turn,X.Move to which place?
7
X| |
-+-+-
 | |X
-+-+-
 | |O
It's your turn,O.Move to which place?
2
X| |
-+-+-
 | |X
-+-+-
 |O|O
It's your turn,X.Move to which place?
8
X|X|
-+-+-
 | |X
-+-+-
 |O|O
It's your turn,O.Move to which place?
1
```

```
X|X|
-+-+-
 | |X
-+-+-
O|O|O

Game Over.
```

**** O won. ****

Do want to play Again?(y/n)n

VIVA QUESTIONS

1) **What is Artificial Intelligence?**

   Artificial Intelligence is an area of computer science that emphasizes the creation of intelligent machine thatwork and reacts like humans.

2) **What is an artificial intelligence Neural Networks?**

   Artificial intelligence Neural Networks can model mathematically the way biological brain works, allowing themachine to think and learn the same way the humans do- making them capable of recognizing things like speech, objects and animals like we do.

3) **What are the various areas where AI (Artificial Intelligence) can be used?**

   Artificial Intelligence can be used in many areas like Computing, Speech recognition, Bio-informatics,Humanoid robot, Computer software, Space and Aeronautics's etc.

4) **Which is not commonly used programming language for AI?**

   Perl language is not commonly used programming language for AI

5) **What is Prolog in AI?**

   In AI, Prolog is a programming language based on logic.

6) **Mention the difference between statistical AI and Classical AI ?**

   Statistical AI is more concerned with "inductive" thought like given a set of pattern, induce the trend etc.While, classical AI, on the other hand, is more concerned with " deductive" thought given as a set of constraints, deduce a

   conclusion etc.

7) **What does a production rule consist of?**

   The production rule comprises of a set of rule and a sequence of steps.

8) **Which search method takes less memory?**

   The "depth first search" method takes less memory.

9) **Which is the best way to go for Game playing problem?**

   Heuristic approach is the best way to go for game playing problem, as it will use the technique based onintelligent guesswork. For example, Chess between humans and computers as it will use brute force computation, looking at hundreds of thousands of positions.

10) **A* algorithm is based on which search method?**

    A* algorithm is based on best first search method, as it gives an idea of optimization and quick choose of path,and all characteristics lie in A* algorithm.

11) **What does a hybrid Bayesian network contain?**

    A hybrid Bayesian network contains both a discrete and continuous variables.

12) **What is agent in artificial intelligence?**

    Anything perceives its environment by sensors and acts upon an environment by effectors are known as Agent.Agent includes Robots, Programs, and Humans etc.

13) **What does Partial order or planning involve?**

    In partial order planning, rather than searching over possible situation it involves searching over the space ofpossible plans. The idea is to construct a plan piece by piece.

14) **What are the two different kinds of steps that we can take in constructing a plan?**

a) Add an operator (action)

b) Add an ordering constraint between operators

15) Which property is considered as not a desirable property of a logical rule-based system?

"Attachment" is considered as not a desirable property of a logical rule based system.

16) What is Neural Network in Artificial Intelligence?

In artificial intelligence, neural network is an emulation of a biological neural system, which receives the data,process the data and gives the output based on the algorithm and empirical data.

17) When an algorithm is considered completed?

An algorithm is said completed when it terminates with a solution when one exists.

18) What is a heuristic function?

A heuristic function ranks alternatives, in search algorithms, at each branching step based on the availableinformation to decide which branch to follow.

19) What is the function of the third component of the planning system?

In a planning system, the function of the third component is to detect when a solution to problem has beenfound.

20) What is "Generality" in AI ?

Generality is the measure of ease with which the method can be adapted to different domains of application.

21) What is a top-down parser?

A top-down parser begins by hypothesizing a sentence and successively predicting lower level constituentsuntil individual pre-terminal symbols are written.

22) Mention the difference between breadth first search and best first search in artificial intelligence? These are the two strategies which are quite similar. In best first search, we expand the nodes in accordancewith the evaluation function. While, in breadth first search a node is expanded in accordance to the cost function of the parent node.

23) What are frames and scripts in "Artificial Intelligence"?

Frames are a variant of semantic networks which is one of the popular ways of presenting non-procedural knowledge in an expert system. A frame which is an artificial data structure is used to divide knowledge into substructure by representing "stereotyped situations'. Scripts are similar to frames, except the values that fill theslots must be ordered. Scripts are used in natural language understanding systems to organize a knowledge basein terms of the situation that the system should understand.

24) What is FOPL stands for and explain its role in Artificial Intelligence?

FOPL stands for First Order Predicate Logic, Predicate Logic provides

a) A language to express assertions about certain "World"

b) An inference system to deductive apparatus whereby we may draw conclusions from such assertion

c) A semantic based on set theory

25) What does the language of FOPL consists of

a) A set of constant symbols

b) A set of variables

c) A set of function symbols

d) The logical connective

e) The Universal Quantifier and Existential Qualifier

f) A special binary relation of equality

26) For online search in 'Artificial Intelligence' which search agent operates by interleaving computation andaction?

In online search, it will first take action and then observes the environment.

27) Which search algorithm will use a limited amount of memory in online search?

RBFE and SMA* will solve any kind of problem that A* can't by using a limited amount of memory.

28) In 'Artificial Intelligence' where you can use the Bayes rule?

In Artificial Intelligence to answer the probabilistic queries conditioned on one piece of evidence, Bayes rulecan be used.

29) For building a Bayes model how many terms are required?

For building a Bayes model in AI, three terms are required; they are one conditional probability and twounconditional probability.

30) While creating Bayesian Network what is the consequence between a node and its predecessors?

While creating Bayesian Network, the consequence between a node and its predecessors is that a node can beconditionally independent of its predecessors.

31) Which is the most straight forward approach for planning algorithm?

State space search is the most straight forward approach for planning algorithm because it takes account ofeverything for finding a solution.