ARTIFICIAL INTELLIGENCE AND APPLICATIONS IN CLOUD COMPUTING

(Code 18CSC312J)

B.Tech (CSE) – 3nd year/6th Semester

Name: Rohan Singh

Registration No.:RA2011028030003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY,

DELHINCR CAMPUS, MODINAGAR

SIKRI KALAN, DELHI MEERUT ROAD, DIST. – GHAZIABAD - 201204

https://www.srmup.in/

Even Semester (2022-2023)

BONAFIDE CERTIFICATE

Registration no.RA2011028030003

Certified to be the bonafide record of work done by **Rohan Singh** *of 6th semester3rd year B.TECH degree course in SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, NCR Campus of Department of Computer Science & Engineering in ARTIFICIAL INTELLIGENCE AND APPLICATIONS IN CLOUD COMPUTING, during the academic year 2022-2023.*

Assistant Professor **Head of the Department(CSE)**

Submitted for university examination held on___/____/___ at SRM IST, NCR Campus.

Internal Examiner-I **Internal Examiner-II**

INDEX

Exp. No.	Title of Experiment	Page No.	Date of Experiment	Date of Completion of Experiment	Teacher's Signature
1	Implementation of Toy problem Example-Implement water jug problem		**05/01/2023**	**12/01/2023**	
2	Developing Agent Program for Real WorldProblem.		**12/01/2023**	**19/01/2023**	
3	Implementation of Constraint satisfaction problem, Example: Implement N- queen Problem		**19/01/2023**	**09/02/2023**	
4	To Implementation and Analysis of BFS andDFS for Application.		**09/02/2023**	**23/02/2023**	
5	To implement Best first search and A* algorithm.		**23/02/2023**	**02/03/2023**	
6	To implement Minimax Algorithm.		**02/03/2023**	**09/03/2023**	
7	Implementation of unification and resolution for real world problems.		**09/03/2023**	**16/03/2023**	
8	Implementation of knowledge representation schemes – use cases.		**16/03/2023**	**23/03/2023**	
9	Implementation of uncertain methods for an application.		**23/03/2023**	**06/04/2023**	
10	Implementation of block world problem.		**06/04/2023**	**13/04/2023**	
11	Implementation of Learning Algo		**13/04/2023**	**20/04/2023**	
12	Development of ensemble model		**20/04/2023**	**20/04/2023**	

INDEX

Exp. No.	Title of Experiment	Page No.	Date of Experiment	Date of Completion of Experiment	Teacher's Signature
1	Implementation of Toy problem Example-Implement water jug problem				
2	Developing Agent Program for Real World Problem.				
3	Implementation of Constraint satisfaction problem, Example: Implement N- queen Problem				
4	To Implementation and Analysis of BFS and DFS for Application.				
5	To implement Best first search and A* algorithm.				
6	To implement Minimax Algorithm.				
7	Implementation of unification and resolution for real world problems.				
8	Implementation of knowledge representation schemes – use cases.				
9	Implementation of uncertain methods for an application.				
10	Implementation of block world problem.				
11	Implementation of Learning Algo				
12	Development of ensemble model				

Experiment 1

Aim – Implementation of Toy problem Example-Implement water jug problem.

Algorithm –

Rule	State	Process
1	(X,Y | X<4)	(4,Y) {Fill 4-gallon jug}
2	(X,Y |Y<3)	(X,3) {Fill 3-gallon jug}
3	(X,Y |X>0)	(0,Y) {Empty 4-gallon jug}
4	(X,Y | Y>0)	(X,0) {Empty 3-gallon jug}
5	(X,Y | X+Y>=4 ^ Y>0)	(4,Y-(4-X)) {Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}
6	(X,Y | X+Y>=3 ^X>0)	(X-(3-Y),3) {Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}
7	(X,Y | X+Y<=4 ^Y>0)	(X+Y,0) {Pour all water from 3-gallon jug into 4-gallon jug}
8	(X,Y | X+Y <=3^ X>0)	(0,X+Y) {Pour all water from 4-gallon jug into 3-gallon jug}
9	(0,2)	(2,0) {Pour 2 gallon water from 3 gallon jug into 4 gallon jug}

Code –

```
print("Water jug problem")
x=int(input("Enter X : "))
y=int(input("Enter Y : "))
while True:
  rn=int(input("Enter the rule no. : "))
  if rn==2:
    if y<3:
      x=0
      y=3
  if rn==3:
    if x>0:
      x=0
      y=3
  if rn==5:
    if x+y>4:
      x=4
      y=y-(4-x)
  if rn==7:
```

```
        if x+y<4:
            x=x+y
            y=0
    if rn==9:
        x=2
        y=0
    print("X=",x)
    print("Y=",y)
    if x==2:
        print("The result is a goal state")
        break
```

Result –

```
>_ Console    Shell
Water jug problem
Enter X : 0
Enter Y : 0
Enter the rule no. : 2
X= 0
Y= 3
Enter the rule no. : 3
X= 0
Y= 3
Enter the rule no. : 5
X= 0
Y= 3
Enter the rule no. : 7
X= 3
Y= 0
Enter the rule no. : 9
X= 2
Y= 0
The result is a goal state
```

Experiment 2

Aim – Developing Agent Program for Real World Problem.

Algorithm –

Code –

```
import random

class Environment:
    def __init__(self):
        self.locationCondition = {'A': '0', 'B': '0'}
        self.locationCondition['A'] = random.randint(0, 1)
        self.locationCondition['B'] = random.randint(0, 1)

class SimpleReflexVacuumAgent(Environment):
    def __init__(self, Environment):
        super().__init__()
        print(Environment.locationCondition)
        self.Score = 0
        self.vacuumLocation = random.randint(0, 1)

        if self.vacuumLocation == 0:
            print("Vacuum is randomly placed at Location A")
            if Environment.locationCondition['A'] == 1:
                print("Location A is Dirty.")
                Environment.locationCondition['A'] = 0
                self.Score += 1
                print("Location A has been Cleaned. :D")
                if Environment.locationCondition['B'] == 1:
                    print("Location B is Dirty.")
                    print("Moving to Location B...")
                    self.Score -= 1
                    Environment.locationCondition['B'] = 0
                    self.Score += 1
                    print("Location B has been Cleaned :D.")
            else:
                if Environment.locationCondition['B'] == 1:
                    print("Location B is Dirty.")
                    print("Moving to Location B...")
                    self.Score -= 1
                    Environment.locationCondition['B'] = 0
                    self.Score += 1
                    print("Location B has been Cleaned. :D")

        elif self.vacuumLocation == 1:
            print("Vacuum is randomly placed at Location B.")
            if Environment.locationCondition['B'] == 1:
                print("Location B is Dirty")
                Environment.locationCondition['B'] = 0
                self.Score += 1
                print("Location B has been Cleaned")
                if Environment.locationCondition['A'] == 1:
```

```
                    print("Location A is Dirty")
                    self.Score -= 1
                    print("Moving to Location A")
                    Environment.locationCondition['A'] = 0
                    self.Score += 1
                    print("Location A has been Cleaned")
            else:
                if Environment.locationCondition['A'] == 1:
                    print("Location A is Dirty")
                    print("Moving to Location A")
                    self.Score -= 1
                    Environment.locationCondition['A'] = 0
                    self.Score += 1
                    print("Location A has been Cleaned")

        print(Environment.locationCondition)
        print("Performance Measurement: " + str(self.Score))

theEnvironment = Environment()
theVacuum = SimpleReflexVacuumAgent(theEnvironment)
```

Result –

```
>_ Console  ×   Shell  ×   +

{'A': 1, 'B': 1}
Vacuum is randomly placed at Location B.
Location B is Dirty
Location B has been Cleaned
Location A is Dirty
Moving to Location A
Location A has been Cleaned
{'A': 0, 'B': 0}
Performance Measurement: 1
>
```

Experiment 3

Aim – Implementation of Constraint satisfaction problem Example: Implement N- queen Problem

Algorithm –

```
while there are untried configurations
{
        generate the next configuration
        if queens don't attack in this configuration then
        {
                print this configuration;
        }
}
```

Code –

```
global N
N = int(input("Enter number of queens: "))

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=" ")
        print(" ")

def isSafe(board, row, col):
    for i in range(col):
        if board[row][i] == 'Q':
            return False
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 'Q':
            return False
    for i, j in zip(range(row, N, 1), range(col, -1, -1)):
        if board[i][j] == 'Q':
            return False
    return True

def solveNQUtil(board, col):
    if col >= N:
        return True
    for i in range(N):
        if isSafe(board, i, col):
            board[i][col] = 'Q'
            if solveNQUtil(board, col+1) == True:
                return True
            board[i][col] = 0
    return False

def solveNQ():
    board = [[0 for i in range(N)] for j in range(N)]
    if solveNQUtil(board, 0) == False:
        print("Solution does not exist")
        return False
    printSolution(board)
    return True
```

solveNQ()

Result –

```
Enter number of queens: 4
0 0 Q 0
Q 0 0 0
0 0 0 Q
0 Q 0 0
```

Experiment 4

Aim – To Implementation and Analysis of BFS and DFS for Application.

Algorithm –

1. Create a node list (Queue) that initially contains the first node N and mark it as visited.
2. Visit the adjacent unvisited vertex of N and insert it in a queue.
3. If there are no remaining adjacent vertices left, remove the first vertex from the queue mark it as visited, display it.
4. Repeat step 1 and step 2 until the queue is empty or the desired node is found.

Code –

```
graph = {
    'S': ['A', 'B'],
    'A': ['C', 'D'],
    'B': ['G', 'H'],
    'C': ['E', 'F'],
    'D': [],
    'G': ['I'],
    'H': [],
    'E': ['K'],
    'F': [],
    'I': [],
    'K': []
}

visited = []
queue = []

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        P = queue.pop(0)
        print(P, end=" ")
        for neighbour in graph[P]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

def dfs(avisit, graph, node):
    if node not in avisit:
        print(node, end=" ")
        avisit.add(node)
        for neighbour in graph[node]:
            dfs(avisit, graph, neighbour)

print("Breadth first search:")
bfs(visited, graph, 'S')

print("\nDepth first search:")
dfs(set(), graph, 'S')
```

Result –

```
>_ Console    Shell
Breadth first search:
S A B C D G H E F I K
Depth first search:
S A C E K F D B G I H
```

Experiment 5

Aim-To implement Best First Search and A* algorithm.

Algorithm-

1. Best First Search-

Step 1: Place the starting node into the OPEN list.
Step 2: If the OPEN list is empty, Stop and return failure.
Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list. If node n is goal then return else
Step 4: Expand the node n, and generate and check the successors of node n. and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 5.
Step 5: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
Step 6: Return to Step 2.

2. A*-

Step1: Place the starting node in the OPEN list.
Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
Step 4:Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
Step 6: Return to Step 2.

Code-

1. Best First Search-

```
from queue import PriorityQueue
import networkx as nx

def best_first_search(source, target, n):
    visited = [0] * n
    visited[source] = True
    pq = PriorityQueue()
    pq.put((0, source))

    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))

    print()

def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

G = nx.Graph()
v = int(input("Enter the number of nodes: "))
graph = [[] for i in range(v)]
```

```
e = int(input("Enter the number of edges: "))
print("Enter the edges along with their weights:")
for i in range(e):
    x, y, z = list(map(int, input().split()))
    addedge(x, y, z)
    G.add_edge(x, y, weight=z)

source = int(input("Enter the Source Node: "))
target = int(input("Enter the Target/Destination Node: "))
print("\nPath: ", end="")
best_first_search(source, target, v)
```

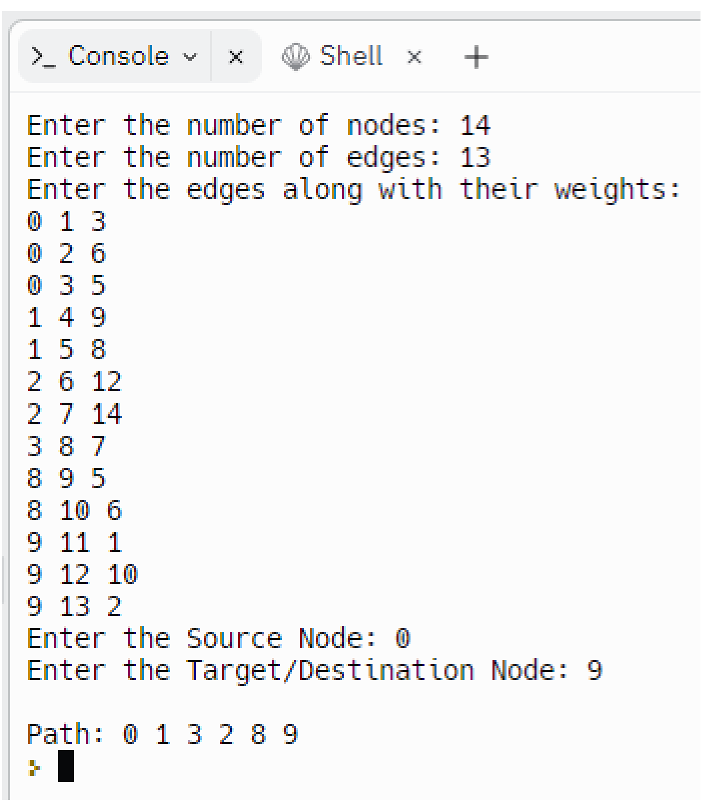
2. A*-

```
from collections import deque

class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list

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

    def h(self, n):
        H = {'A': 1, 'B': 1, 'C': 1, 'D': 1}
        return H[n]

    def a_star_algorithm(self, start_node, stop_node):
        open_list = set([start_node])
        closed_list = set([])
        g = {}
        g[start_node] = 0
        parents = {}
        parents[start_node] = start_node

        while len(open_list) > 0:
            n = None
            for v in open_list:
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v
            if n == None:
                print('Path does not exist!')
                return None

            if n == stop_node:
                reconst_path = []
                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]
                reconst_path.append(start_node)
                reconst_path.reverse()
                print('Path found: {}'.format(reconst_path))
                return reconst_path

            for (m, weight) in self.get_neighbors(n):
```

```
            if m not in open_list and m not in closed_list:
                open_list.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n
                    if m in closed_list:
                        closed_list.remove(m)
                        open_list.add(m)
        open_list.remove(n)
        closed_list.add(n)

    print('Path does not exist!')
    return None

adjacency_list = {'A': [('B', 1), ('C', 3), ('D', 7)],
             'B': [('D', 5)],
             'C': [('D', 12)]}

graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

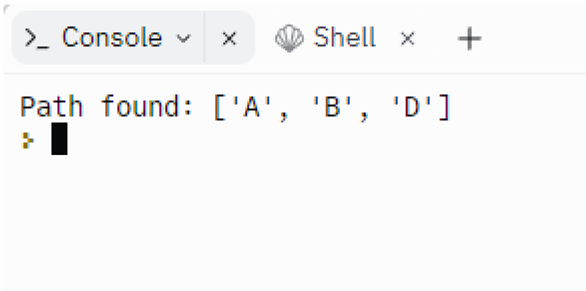
Result-

1. Best First Search-



```
>_ Console     Shell
Enter the number of nodes: 14
Enter the number of edges: 13
Enter the edges along with their weights:
0 1 3
0 2 6
0 3 5
1 4 9
1 5 8
2 6 12
2 7 14
3 8 7
8 9 5
8 10 6
9 11 1
9 12 10
9 13 2
Enter the Source Node: 0
Enter the Target/Destination Node: 9

Path: 0 1 3 2 8 9
>
```

2. A*-



Experiment 6

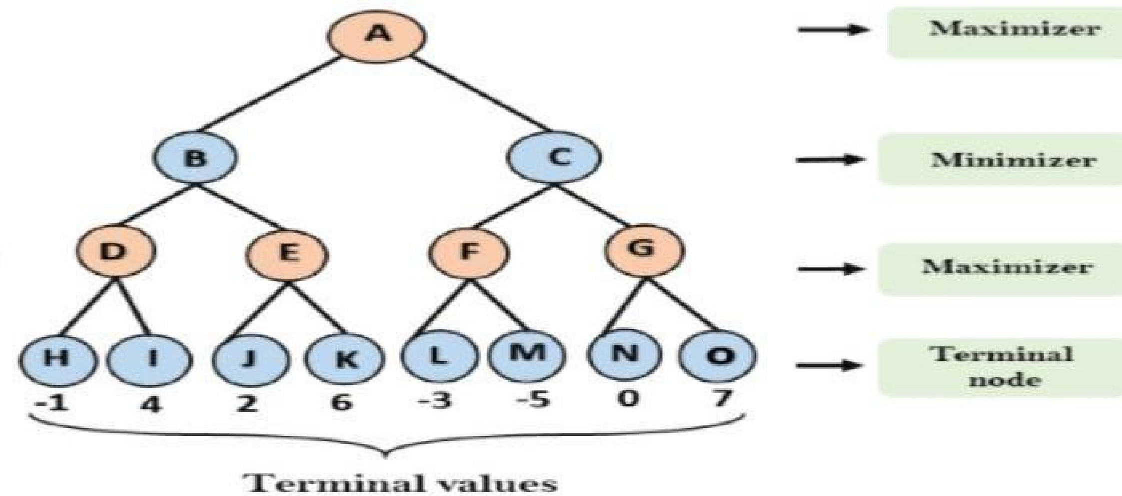
Aim – To implement Minimax Algorithm.

Algorithm –

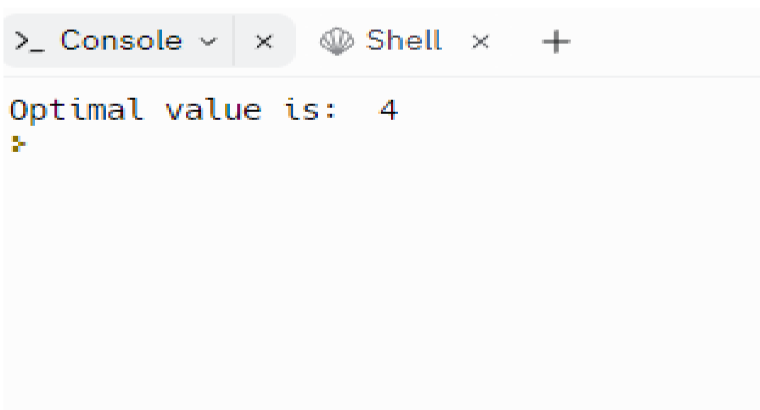
function minimax(node, depth, Player)
if depth ==0 or node is a terminal node then return value(node)
If Player =‘Max’ // for Maximizer Player
set $\alpha = -\infty$ //worst case value for MAX
for each child of node do
value= minimax(child, depth-1, ’MIN’)
α= max(α, Value) //gives Maximum of the values return (α)
else // for Minimizer player
set $\alpha = +\infty$ //worst case value for MIN
for each child of node do
value= minimax(child, depth-1, ’MAX’)
α = min(α, Value)
//gives minimum of the values return (α)

Code –

```
import math

def minimax(curDepth, nodeIndex, maxTurn, scores, targetDepth):
    if curDepth == targetDepth:
        return scores[nodeIndex]
    if maxTurn:
        return max(minimax(curDepth+1, nodeIndex*2, False, scores, targetDepth),
                   minimax(curDepth+1, nodeIndex*2+1, False, scores, targetDepth))
    else:
        return min(minimax(curDepth+1, nodeIndex*2, True, scores, targetDepth),
                   minimax(curDepth+1, nodeIndex*2+1, True, scores, targetDepth))

scores = [-1, 4, 2, 6, -3, -5, 0, 7]
treeDepth = math.log(len(scores), 2)
print("Optimal value is:", minimax(0, 0, True, scores, treeDepth))
```


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal
node

Result –


>_ Console
Shell
Optimal value is: 4

Experiment 7

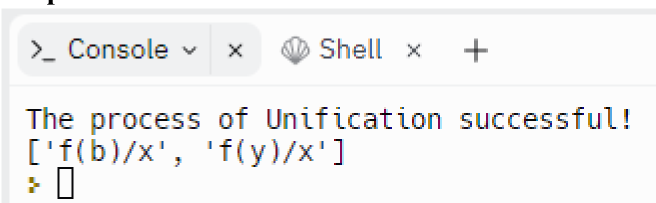
Aim – Implementation of unification and resolution for real world problems.

Unification
Code:-
```
def get_index_comma(string):
    index_list = list()
    par_count = 0
    for i in range(len(string)):
        if string[i] == ',' and par_count == 0:
            index_list.append(i)
        elif string[i] == '(':
            par_count += 1
        elif string[i] == ')':
            par_count -= 1
    return index_list

def is_variable(expr):
    for i in expr:
        if i == '(' or i == ')':
            return False
    return True

def process_expression(expr):
    expr = expr.replace(' ', '')
    index = None
    for i in range(len(expr)):
        if expr[i] == '(':
            index = i
            break
    predicate_symbol = expr[:index]
    expr = expr.replace(predicate_symbol, '')
    expr = expr[1:len(expr) - 1]
    arg_list = list()
    indices = get_index_comma(expr)
    if len(indices) == 0:
        arg_list.append(expr)
    else:
        arg_list.append(expr[:indices[0]])
        for i, j in zip(indices, indices[1:]):
            arg_list.append(expr[i + 1:j])
        arg_list.append(expr[indices[len(indices) - 1] + 1:])
    return predicate_symbol, arg_list

def get_arg_list(expr):
    _, arg_list = process_expression(expr)
    flag = True
    while flag:
        flag = False
        for i in arg_list:
            if not is_variable(i):
                flag = True
                _, tmp = process_expression(i)
                for j in tmp:
```

```
                if j not in arg_list:
                    arg_list.append(j)
            arg_list.remove(i)
    return arg_list

def check_occurs(var, expr):
    arg_list = get_arg_list(expr)
    if var in arg_list:
        return True
    return False

def unify(expr1, expr2):
    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
            return 'Null'
        else:
            return False
    elif is_variable(expr1) and not is_variable(expr2):
        if check_occurs(expr1, expr2):
            return False
        else:
            tmp = str(expr2) + '/' + str(expr1)
            return tmp
    elif not is_variable(expr1) and is_variable(expr2):
        if check_occurs(expr2, expr1):
            return False
        else:
            tmp = str(expr1) + '/' + str(expr2)
            return tmp
    else:
        predicate_symbol_1, arg_list_1 = process_expression(expr1)
        predicate_symbol_2, arg_list_2 = process_expression(expr2)
        if predicate_symbol_1 != predicate_symbol_2:
            return False
        elif len(arg_list_1) != len(arg_list_2):
            return False
        else:
            sub_list = list()
            for i in range(len(arg_list_1)):
                tmp = unify(arg_list_1[i], arg_list_2[i])
                if not tmp:
                    return False
                elif tmp == 'Null':
                    pass
                else:
                    if type(tmp) == list:
                        for j in tmp:
                            sub_list.append(j)
                    else:
                        sub_list.append(tmp)
            return sub_list

f1 = 'Q(a, g(x, a), f(y))'
f2 = 'Q(a, g(f(b), a), x)'
result = unify(f1, f2)
f1 = 'Q(a, g(x, a), f(y))'
```

```
f2 = 'Q(a, g(f(b), a), x)'
result = unify(f1, f2)
if not result:
    print('The process of Unification failed!')
else:
    print('The process of Unification successful!')
    print(result)
```

Output:-

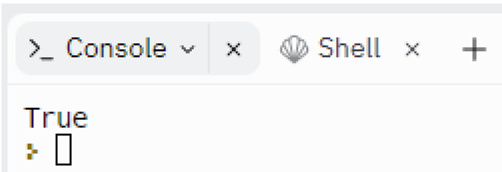


Resolution
Code:-

```
from itertools import combinations

def resolve(c1, c2):
    """
    Given two clauses c1 and c2, returns a set of new clauses
    that can be inferred using the resolution rule.
    """
    resolvents = set()
    for literal in c1:
        if frozenset([-literal]) in c2:
            new_c1 = c1.difference([literal])
            new_c2 = c2.difference([frozenset([-literal])])
            resolvent = frozenset(new_c1.union(new_c2))
            resolvents.add(resolvent)
    return resolvents

def resolve_all(clauses):
    """
    Given a set of clauses, repeatedly applies the resolution rule
    until no new clauses can be inferred.
    """
    new_clauses = frozenset(map(frozenset, clauses))
    while True:
        all_resolvents = set()
        for c1, c2 in combinations(new_clauses, 2):
            resolvents = resolve(c1, c2)
            all_resolvents.update(resolvents)
        if not all_resolvents.difference(new_clauses):
            return new_clauses
        new_clauses = frozenset(new_clauses.union(all_resolvents))

def is_satisfiable(clauses):
    """
    Given a set of clauses, returns True if they are satisfiable
    (i.e. there exists a truth assignment that satisfies all clauses),
    and False otherwise.
    """
```

```
    new_clauses = resolve_all(clauses)
    if frozenset() in new_clauses:
        return False
    else:
        return True

# Example usage:
clauses = [{1, 2, 3}, {-1, -2}, {-1, -3}]
print(is_satisfiable(clauses))  # True
```

Output:-



Experiment 8

- **Aim** – Write a python program to implement the knowledge representation schemes using test cases.

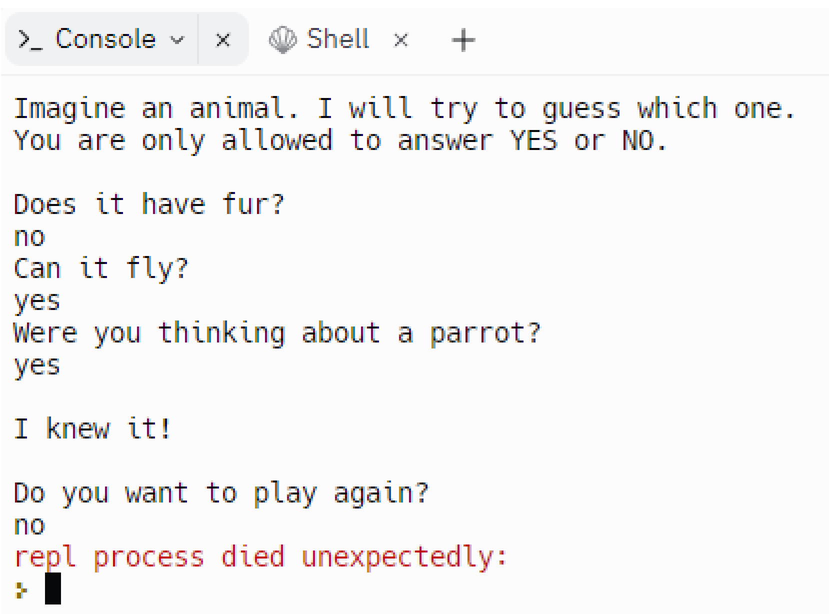
- **code –**

```
import sys

def definiteNoun(s):
    s = s.lower().strip()
    if s in ['a', 'e', 'i', 'o', 'u', 'y']:
        return "an " + s
    else:
        return "a " + s

def removeArticle(s):
    s = s.lower().strip()
    if s[0:3] == "an ":
        return s[3:]
    if s[0:2] == "a ":
        return s[2:]
    return s

def makeQuestion(question, yes, no):
    return [question, yes, no]

def isQuestion(p):
    return type(p).__name__ == "list"

def askQuestion(question):
    print ("\r%s " % question,)
    return sys.stdin.readline().strip().lower()

def getAnswer(question):
    if isQuestion(question):
        return askQuestion(question[0])
    else:
        return askQuestion("Were you thinking about %s?" % definiteNoun(question))

def answeredYes(answer):
    if len(answer) > 0:
        return answer.lower()[0] == "y"
    return False

def gameOver(message):
    global tries
    print ("")
    print ("\r%s" % message)
    print ("")

def playAgain():
    return answeredYes(askQuestion("Do you want to play again?"))

def correctGuess(message):
    global tries
    gameOver(message)
```

```
    if playAgain():
        print ("")
        tries = 0
        return Q
    else:
        sys.exit(0)

def nextQuestion(question, answer):
    global tries
    tries += 1
    if isQuestion(question):
        if answer:
            return question[1]
        else:
            return question[2]
    else:
        if answer:
            return correctGuess("I knew it!")
        else:
            return makeNewQuestion(question)

def replaceAnswer(tree, find, replace):
    if not isQuestion(tree):
        if tree == find:
            return replace
        else:
            return tree
    else:
        return makeQuestion(tree[0],
                            replaceAnswer(tree[1], find, replace),
                            replaceAnswer(tree[2], find, replace))

def makeNewQuestion(wrongAnimal):
    global Q, tries
    correctAnimal = removeArticle(askQuestion("I give up. What did you think about?"))
    newQuestion = askQuestion("Enter a question that would distinguish %s from %s:"
                              % (definiteNoun(correctAnimal), definiteNoun(wrongAnimal))).capitalize()
    yesAnswer = answeredYes(askQuestion("If I asked you this question " +
                                        "and you thought about %s, what would the correct answer be?" %
      definiteNoun(correctAnimal)))
    # Create new question node
    if yesAnswer:
        q = makeQuestion(newQuestion, correctAnimal, wrongAnimal)
    else:
        q = makeQuestion(newQuestion, wrongAnimal, correctAnimal)
    Q = replaceAnswer(Q, wrongAnimal, q)
    tries = 0
    return Q

def addNewQuestion(wrongAnimal, newques, correct):
 global Q
 q = makeQuestion(newques, correct, wrongAnimal)
 Q = replaceAnswer(Q, wrongAnimal, q)
 return Q

tries = 0
```

```
Q = (makeQuestion('Does it have fur?', 'Tiger', 'Penguin'))
q = addNewQuestion('Tiger', 'Does it have dark spots?', 'Leopard')
q = addNewQuestion('Leopard', 'Is it the fastest animal?', 'Cheetah')
q = addNewQuestion('Penguin', 'Can it fly?', 'Parrot')
q = Q

print("Imagine an animal. I will try to guess which one.")
print("You are only allowed to answer YES or NO.")
print("")

try:
    while True:
        ans = answeredYes(getAnswer(q))
        q = nextQuestion(q, ans)
except KeyboardInterrupt:
    sys.exit(0)
except Exception:
    sys.exit(1)
```

Result :-

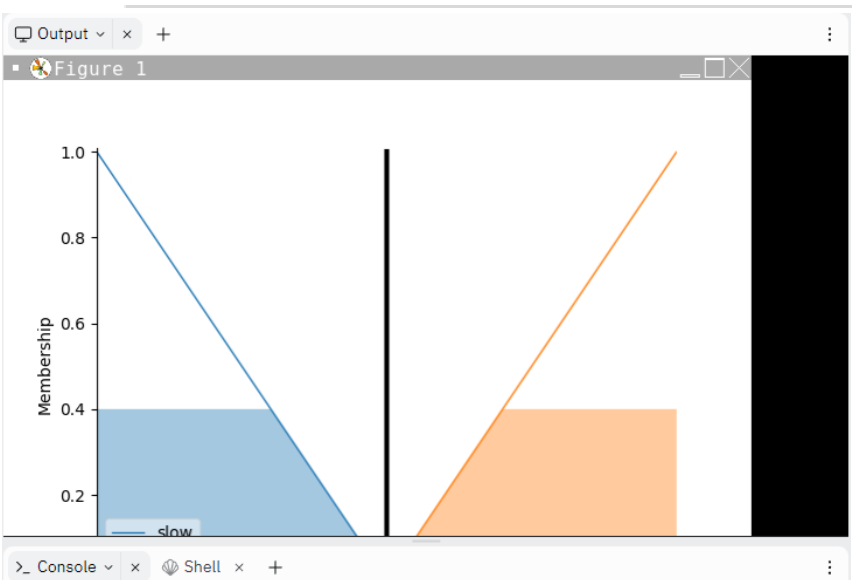


Experiment 9

Aim – Implementation of uncertain methods for an application.

Code –
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

temp = ctrl.Antecedent(np.arange(0, 101, 1), 'Temperature')
humidity = ctrl.Antecedent(np.arange(0, 101, 1), 'Humidity')
speed = ctrl.Consequent(np.arange(0, 101, 1), 'Speed')

temp['cold'] = fuzz.trimf(temp.universe, [0, 0, 50])
temp['hot'] = fuzz.trimf(temp.universe, [50, 100, 100])
humidity['dry'] = fuzz.trimf(humidity.universe, [0, 0, 50])
humidity['wet'] = fuzz.trimf(humidity.universe, [50, 100, 100])
speed['slow'] = fuzz.trimf(speed.universe, [0, 0, 50])
speed['fast'] = fuzz.trimf(speed.universe, [50, 100, 100])

rule1 = ctrl.Rule(temp['cold'] | humidity['dry'], speed['slow'])
rule2 = ctrl.Rule(temp['hot'] | humidity['wet'], speed['fast'])
rule3 = ctrl.Rule(humidity['dry'] & temp['hot'], speed['fast'])
rule4 = ctrl.Rule(humidity['wet'] & temp['cold'], speed['slow'])

speed_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4])
speed_simulation = ctrl.ControlSystemSimulation(speed_ctrl)

speed_simulation.input['Temperature'] = 30
speed_simulation.input['Humidity'] = 70
speed_simulation.compute()

speed.view(sim=speed_simulation)
plt.show()
```

Result :-



Experiment 10

Aim – Write a python program to implement the block world problem using correct artificial intelligence optimization techniques.

Code –

```
import time

class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
        print("Stack created")
        self.stack_pointer = None

    def push(self, x):
        if not isinstance(x, Node):
            x = Node(x)
        print(f"Adding {x.data} to the top of stack")
        if self.is_empty():
            self.stack_pointer = x
        else:
            x.next = self.stack_pointer
            self.stack_pointer = x

    def pop(self):
        if not self.is_empty():
            print(f"Removing node on top of stack")
            curr = self.stack_pointer
            self.stack_pointer = self.stack_pointer.next
            curr.next = None
            return curr.data
        else:
            return "Stack is empty"

    def is_empty(self):
        return self.stack_pointer == None

    def peek(self):
        if not self.is_empty():
            return self.stack_pointer.data

    def __str__(self):
        print("Printing Stack state...")
        to_print = ""
        curr = self.stack_pointer
        while curr is not None:
            to_print += str(curr.data) + "->"
            curr = curr.next
```

```
        if to_print:
            print("Stack Pointer")
            print(" |")
            print(" V")
            return "[" + to_print[:-2] + "]"
        return "[]"

print ("INITIAL STATE : {[1], [2], [3], [4], [5]}")
print("-"*70)
print ("FINAL STATE :[4->3->2->1]")

my_stack = Stack()
print("Checking if stack is empty:", my_stack.is_empty())

my_stack.push(1)
time.sleep(1)
my_stack.push(2)
print(my_stack)
time.sleep(1)
my_stack.push(3)
time.sleep(1)
my_stack.push(4)
time.sleep(1)
print("Checking item on top of stack:", my_stack.peek())
time.sleep(1)
my_stack.push(5)
print(my_stack)
time.sleep(1)
print(my_stack.pop())
time.sleep(1)
print(my_stack.pop())
print(my_stack)
time.sleep(1)
my_stack.push(4)
print(my_stack)
time.sleep(1)
```

Result :-

>_ Console × Shell × +

```
INITIAL STATE : {[1], [2], [3], [4], [5]}
----------------------------------------------------------------------
FINAL STATE :[4->3->2->1]
Stack created
Checking if stack is empty: True
Adding 1 to the top of stack
Adding 2 to the top of stack
Printing Stack state...
Stack Pointer
 |
 V
[2->1]
Adding 3 to the top of stack
Adding 4 to the top of stack
Checking item on top of stack: 4
Adding 5 to the top of stack
Printing Stack state...
Stack Pointer
 |
 V
[5->4->3->2->1]
Removing node on top of stack
5
Removing node on top of stack
4
Printing Stack state...
Stack Pointer
 |
 V
[3->2->1]
Adding 4 to the top of stack
Printing Stack state...
Stack Pointer
 |
 V
[4->3->2->1]
```

Experiment 11

Aim:-Write a python program for Implementation of Learning Algorithm

Code:-

```
import requiredLibraries
import functionFiles

ticker = Select_a_stock_ticker_of_your_choice

# fetch the ohlc data in a pandas dataframe
df = fetch_data_from_api

# performing feature engineering using functions from ta library

# add ema 100 and ema 200 with signal
ema_100(df)
ema_200(df)
ema_signal(df)

# add vwap with signal
vwap(df)
vwap_signal(df)

# add stochastic RSI with signal
stochasticRSI(df)
StochRSI_signal(df)

# add MACD with signal
macd(df)
macd_signal(df)

# add absolute change and change percentage to all features
abs_change(df)
abs_pct_change(df)

# finally add the target variable
trend(df)

# clean and normalise the final dataframe
clean(df)
normalise(df) # techniques used are min max scaling and polarizing
reindex(df)

# initialize empty np arrays for all the features
initialize_np_arrays()

# put data from the df into the numpy arrays in batches of 230 datapoints also including the target variable
for i in range (0, df.shape[0] - 230):
   np_arrays.append(df.iloc[i:i+230, featurePositionNumber])
   y.append(df.iloc[i+230, targetVariablePositionNumber])

# reshape target variable as it has variable dimensionality
y = np.reshape(y, (len(y), 1))
```

```
# put all the np arrays except target variable into another np array X
X = np.stack([all_np_arrays])

# split the data into training and testing data, 80% is training and 20% is testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=0)

# initialize the lstm model
model = Sequential()

# defining the architecture of the lstm model
model.add(LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(LSTM(128, return_sequences=False))
model.add(Dense(128, activation='tanh'))
model.add(Dense(1))

# Compile the model with appropriate optimizer, loss function and metrics
optimizers.SGD(momentum=0.9)
model.compile(optimizer='SGD', loss='mse', metrics=['mae'])

# training the model
model.fit(X_train, y_train, validation_split=0.2, epochs=20, batch_size=12)

# evaluating the model
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# graph to look at the performance

predictions = model.predict(X_test)
cmp = [1 if x > 0.35 else -1 if x < -0.35 else 0 for x in predictions]

plt.figure(figsize=(24,12))
plt.plot(cmp[-200:-100])
plt.plot(y_test[-200:-100],'r', linestyle='--' )
plt.show
```

Implementation:-

Implementing an LSTM model for predicting stock trends involves collecting and pre-processing relevant data. Then, the model is trained to identify patterns and relationships between the indicators and stock trends. Once trained, it can make predictions with evaluated accuracy using various metrics. Further refinements can be made by adjusting hyperparameters and modifying the indicators to improve performance, providing valuable insights for investors."

Application :-

LSTM models are a type of RNN used in applications like NLP, image processing, and time series forecasting. In NLP, they're used for tasks such as language translation, speech recognition, and sentiment analysis. In image and video processing, they can be used for image captioning, object recognition, and action recognition. They're also useful in finance for stock price prediction and risk management. LSTM's flexibility makes them a powerful tool for various fields.

Experiment 12

AIM- Write a python program for the development of ensemble model.

Code :-

```
# Import required libraries
import numpy as np
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Load dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Create individual classifiers
lr = LogisticRegression(random_state=1)
dt = DecisionTreeClassifier(random_state=1)
svm = SVC(random_state=1)

# Create an ensemble of classifiers
ensemble = VotingClassifier(estimators=[('lr', lr), ('dt', dt), ('svm', svm)], voting='hard')

# Train the ensemble on the dataset
ensemble.fit(X, y)

# Test the ensemble on a new dataset
X_test = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_pred = ensemble.predict(X_test)

# Print the predicted classes
print(y_pred)
```

Result :-