University School of Automation and Robotics

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

East Delhi Campus, Surajmal Vihar

Delhi - 110092

ज्योतिर्वृणीत तमसो विजानन्

इ वि

USAR

UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS

ARTIFICIAL INTELLIGENCE LAB

File

COURSE CODE: ARD251

2022-2023

SUBMITTED TO:

Dr. Sanjay

SUBMITTED BY:

Vipul Goyal

IIOT-B1

ENROLL. NO: *02419011721*

INDEX

S.No	Program
1.	Implement Breadth First Search Techniques for given graph, water-jug problems.
2.	Implement Depth First Search Techniques for given graph, water-jug problems.
3.	Implement String construction Problem using A* Algorithm.
4.	Build 8-Puzzle solver using A* Algorithm.
5.	Solving region-coloring problem in Constraint Satisfaction framework.
6.	Implement various fuzzification and defuzzification method.
7.	Build a Fuzzy Inference system for restaurant tipping. Consider two input variable Service [0-10] and quality of food [0-10]. Consider tipping as output [0-25] % of bill amount. Consider all other assumptions by own.
8.	To implement logic programming, list processing, variable and constants in PROLOG
9.	Write a recursive program to compute factorial, Fibonacci, tower-of-Hanoi in PROLOG.
10.	To Implement family tree program in PROLOG

PROGRAM-1: Write a program to implement breadth first search for **“Water Jug Program”.**

CODE:

jupyter VIPUL GOYAL IIOT Last Checkpoint: 3 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) Logout

In [3]:

```
#NAME:       VIPUL GOYAL
#ENROLL NO: 02419011721
#BRANCH:     IIOT-B1
```

LAB-1: W.A.P TO IMPLEMENT BREADTH FIRST SEARCH FOR WATER JUG PROGRAM.

In [2]:

```
graph={'(0,0)':['(5,0)','(0,3)'],
       '(5,0)':['(5,3)','(2,3)'],
       '(5,3)':[],
       '(2,3)':['(2,0)'],
       '(2,0)':['(0,2)'],
       '(0,2)':['(5,2)'],
       '(5,2)':['(4,3)'],
       '(4,3)':['(4,0)'],
       '(4,0)':[],
       '(0,3)':['(3,0)'],
       '(3,0)':['(3,3)'],
       '(3,3)':['(5,1)'],
       '(5,1)':['(0,1)'],
       '(0,1)':['(1,0)'],
       '(1,0)':['(1,3)'],
       '(1,3)':['(4,0)'],
       '(4,0)':[]}

print("Graph:",graph)
print(" ")
print("Graph Keys:", graph.keys())
print(" ")
print("Graph Values:",graph.values())
print(" ")

#print(graph.items())
print("Graph Items:")
for i,j in graph.items():
    print(i, ':', j, sep=' ')


queue=[]
visited=[]
def bfs(visited,graph,node):
    visited.append(node)
    queue.append(node)
    while queue:
        m=queue.pop(0)
        print(m,end=" ")
        for neighbor in graph[m]:
            if neighbor not in visited:
                visited.append(neighbor)
                queue.append(neighbor)
print("")
print("bsf: ")
bfs(visited,graph,'(0,0)')
```

OUTPUT:

```
Graph: {'(0,0)': ['(5,0)', '(0,3)'], '(5,0)': ['(5,3)', '(2,3)'], '(5,3)': [], '(2,3)': ['(2,0)'], '(2,0)': ['(0,2)'], '(0,2)':
['(5,2)'], '(5,2)': ['(4,3)'], '(4,3)': ['(4,0)'], '(4,0)': [], '(0,3)': ['(3,0)'], '(3,0)': ['(3,3)'], '(3,3)': ['(5,1)'],
'(5,1)': ['(0,1)'], '(0,1)': ['(1,0)'], '(1,0)': ['(1,3)'], '(1,3)': ['(4,0)']}

Graph Keys: dict_keys(['(0,0)', '(5,0)', '(5,3)', '(2,3)', '(2,0)', '(0,2)', '(5,2)', '(4,3)', '(4,0)', '(0,3)', '(3,0)', '(3,
3)', '(5,1)', '(0,1)', '(1,0)', '(1,3)'])

Graph Values: dict_values([['(5,0)', '(0,3)'], ['(5,3)', '(2,3)'], [], ['(2,0)'], ['(0,2)'], ['(5,2)'], ['(4,3)'], ['(4,0)'],
[], ['(3,0)'], ['(3,3)'], ['(5,1)'], ['(0,1)'], ['(1,0)'], ['(1,3)'], ['(4,0)']])

Graph Items:
(0,0) : ['(5,0)', '(0,3)']
(5,0) : ['(5,3)', '(2,3)']
(5,3) : []
(2,3) : ['(2,0)']
(2,0) : ['(0,2)']
(0,2) : ['(5,2)']
(5,2) : ['(4,3)']
(4,3) : ['(4,0)']
(4,0) : []
(0,3) : ['(3,0)']
(3,0) : ['(3,3)']
(3,3) : ['(5,1)']
(5,1) : ['(0,1)']
(0,1) : ['(1,0)']
(1,0) : ['(1,3)']
(1,3) : ['(4,0)']

bsf:
(0,0) (5,0) (0,3) (5,3) (2,3) (3,0) (2,0) (3,3) (0,2) (5,1) (5,2) (0,1) (4,3) (1,0) (4,0) (1,3)

In [ ]:
```

PROGRAM-2: Write a program to implement “**Depth First Search**” for water jug program.

CODE:

jupyter VIPUL GOYAL IIOT Last Checkpoint: 25 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3 (ipykernel)

LAB-2: W.A.P TO IMPLEMENT DEPTH FIRST SEARCH FOR WATER JUG PROGRAM.

In [6]:

```
graph={'(0,0)':['(5,0)','(0,3)'],
       '(5,0)':['(5,3)','(2,3)'],
       '(5,3)':[],
       '(2,3)':['(2,0)'],
       '(2,0)':['(0,2)'],
       '(0,2)':['(5,2)'],
       '(5,2)':['(4,3)'],
       '(4,3)':['(4,0)'],
       '(4,0)':[],
       '(0,3)':['(3,0)'],
       '(3,0)':['(3,3)'],
       '(3,3)':['(5,1)'],
       '(5,1)':['(0,1)'],
       '(0,1)':['(1,0)'],
       '(1,0)':['(1,3)'],
       '(1,3)':['(4,0)'],
       '(4,0)':[]}

visited=set()
def dfs(visited,graph,node):
    if node not in visited:
        visited.add(node)
        print(node)
        for neighbor in graph[node]:
            dfs(visited,graph,neighbor)
print("dfs: ")
dfs(visited,graph,'(0,0)')
```

OUTPUT:

```
dfs:
(0,0)
(5,0)
(5,3)
(2,3)
(2,0)
(0,2)
(5,2)
(4,3)
(4,0)
(0,3)
(3,0)
(3,3)
(5,1)
(0,1)
(1,0)
(1,3)
```

In [ ]:

PROGRAM-3: Constructing a string using A*.

CODE:

jupyter Untitled2 Last Checkpoint: 12 minutes ago (autosaved)

LAB-3: Constructing a string using A*

In [3]:

```python
from simpleai.search import SearchProblem, astar

GOAL = 'HELLO WORLD'


class HelloProblem(SearchProblem):
    def actions(self, state):
        if len(state) < len(GOAL):
            return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ')
        else:
            return []

    def result(self, state, action):
        return state + action

    def is_goal(self, state):
        return state == GOAL

    def heuristic(self, state):
        # how far are we from the goal?
        wrong = sum([1 if state[i] != GOAL[i] else 0
                    for i in range(len(state))])
        missing = len(GOAL) - len(state)
        return wrong + missing

problem = HelloProblem(initial_state='')
result = astar(problem)

print(result.state)
print(result.path())
```

OUTPUT:

```
HELLO WORLD
[(None, ''), ('H', 'H'), ('E', 'HE'), ('L', 'HEL'), ('L', 'HELL'), ('O', 'HELLO'), (' ', 'HELLO '), ('W', 'HELLO W'), ('O', 'HELLO WO'),
```

PROGRAM-4: Building an 8-puzzle solver.

CODE:

LAB-4:Build 8-Puzzle solver using A* Algorithm.

In [3]:
```
from simpleai.search import astar, SearchProblem
# Class containing methods to solve the puzzle
class PuzzleSolver(SearchProblem):
    # Action method to get the list of the possible
# numbers that can be moved in to the empty space
    def actions(self, cur_state):
        rows = string_to_list(cur_state)
        row_empty, col_empty = get_location(rows, 'e')
        actions = []
        if row_empty > 0:
            actions.append(rows[row_empty - 1][col_empty])
        if row_empty < 2:
            actions.append(rows[row_empty + 1][col_empty])
        if col_empty > 0:
            actions.append(rows[row_empty][col_empty - 1])
        if col_empty < 2:
            actions.append(rows[row_empty][col_empty + 1])
        return actions
    def result(self, state, action):
        rows = string_to_list(state)
        row_empty, col_empty = get_location(rows, 'e')
        row_new, col_new = get_location(rows, action)
        rows[row_empty][col_empty], rows[row_new][col_new] = \
        rows[row_new][col_new], rows[row_empty][col_empty]
        return list_to_string(rows)
# Returns true if a state is the goal state
    def is_goal(self, state):
        return state == GOAL
    # Returns an estimate of the distance from a state to
# the goal using the manhattan distance
    def heuristic(self, state):
        rows = string_to_list(state)
        distance = 0
        for number in '12345678e':
            row_new, col_new = get_location(rows, number)
            row_new_goal, col_new_goal = goal_positions[number]
            distance += abs(row_new - row_new_goal) + abs(col_new - col_new_goal)
        return distance
    # Convert list to string
def list_to_string(input_list):
        return '\n'.join(['-'.join(x) for x in input_list])
    # Convert string to list
def string_to_list(input_string):
        return [x.split('-') for x in input_string.split('\n')]
    # Find the 2D location of the input element
def get_location(rows, input_element):
        for i, row in enumerate(rows):
            for j, item in enumerate(row):
                if item == input_element:
                    return i, j
# Final result that we want to achieve
GOAL = '''1-2-3
4-5-6
7-8-e'''
# Starting point
INITIAL = '''1-e-2
6-3-4
7-5-8'''
# Create a cache for the goal position of each piece
goal_positions = {}
rows_goal = string_to_list(GOAL)
for number in '12345678e':
    goal_positions[number] = get_location(rows_goal, number)
```

```
goal_positions
```

```
{'1': (0, 0),
 '2': (0, 1),
 '3': (0, 2),
 '4': (1, 0),
 '5': (1, 1),
 '6': (1, 2),
 '7': (2, 0),
 '8': (2, 1),
 'e': (2, 2)}
```

```
rows_goal
```

```
[['1', '2', '3'], ['4', '5', '6'], ['7', '8', 'e']]
```

```
ls=list_to_string(rows_goal)
print(ls)
```

```
1-2-3
4-5-6
7-8-e
```

```
In [ ]: # Create the solver object
        result = astar(PuzzleSolver(INITIAL))
```

```
In [ ]: print(result.state)
        print(result.path())
```

OUTPUT:

```
In [ ]: 1-2-3
        4-5-6
        7-8-e
        [(None, '1-e-2\n6-3-4\n7-5-8'), ('2', '1-2-e\n6-3-4\n7-5-8'), ('4', '1-2-4\n6-3-e\n7-5-8'), ('3', '1-2-4\n6-e-3\n7-5-8'), ('6', '
```

```python
# Print the results
for i, (action, state) in enumerate(result.path()):
    print()
    if action == None:
        print('Initial configuration')
    elif i == len(result.path()) - 1:
        print('After moving', action, 'into the empty space. Goal achieved!')
    else:
        print('After moving', action, 'into the empty space')
        print(state)
```

OUTPUT:

```
Initial configuration

After moving 2 into the empty space
1-2-e
6-3-4
7-5-8

After moving 4 into the empty space
1-2-4
6-3-e
7-5-8

After moving 3 into the empty space
1-2-4
6-e-3
7-5-8

After moving 6 into the empty space
1-2-4
e-6-3
7-5-8

After moving 1 into the empty space
e-2-4
...
4-5-6
7-e-8

After moving 8 into the empty space. Goal achieved!
```

PROGRAM-5: Solving region-coloring problem in Constraint Satisfaction framework.

CODE:

1/4/23, 8:52 PM Vipul IIOT - Jupyter Notebook

LAB-5: CSP.

In [3]:

```
from simpleai.search import CspProblem, backtrack
# Define the function that imposes the constraint
# that neighbors should be different
def constraint_func(names, values):
    return values[0] != values[1]

# Specify the variables
names = ('Mark', 'Julia', 'Steve', 'Amanda', 'Brian',
             'Joanne', 'Derek', 'Allan', 'Michelle', 'Kelly')
    # Define the possible colors
colors = dict((name, ['red', 'green', 'blue', 'gray']) for name
                  in names)
    # Define the constraints
constraints = [
    (('Mark', 'Julia'), constraint_func),
(('Mark', 'Steve'), constraint_func),
(('Julia', 'Steve'), constraint_func),
(('Julia', 'Amanda'), constraint_func),
(('Julia', 'Derek'), constraint_func),
(('Julia', 'Brian'), constraint_func),
(('Steve', 'Amanda'), constraint_func),
(('Steve', 'Allan'), constraint_func),
(('Steve', 'Michelle'), constraint_func),
(('Amanda', 'Michelle'), constraint_func),
(('Amanda', 'Joanne'), constraint_func),
(('Amanda', 'Derek'), constraint_func),
(('Brian', 'Derek'), constraint_func),
(('Brian', 'Kelly'), constraint_func),
(('Joanne', 'Michelle'), constraint_func),
(('Joanne', 'Amanda'), constraint_func),
(('Joanne', 'Derek'), constraint_func),
(('Joanne', 'Kelly'), constraint_func),
(('Derek', 'Kelly'), constraint_func),
]

# Solve the problem
problem = CspProblem(names, colors, constraints)

# Print the solution
output = backtrack(problem)
print('\nColor mapping:\n')
for k, v in output.items():
    print(k, '==>', v)
```

OUTPUT:

```
Color mapping:

Mark ==> red
Julia ==> green
Steve ==> blue
Amanda ==> red
Brian ==> red
Joanne ==> green
Derek ==> blue
Allan ==> red
Michelle ==> gray
Kelly ==> gray
```

PROGRAM-6: Implement various fuzzification and defuzzification method.

CODE:

jupyter Untitled5 Last Checkpoint: 5 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

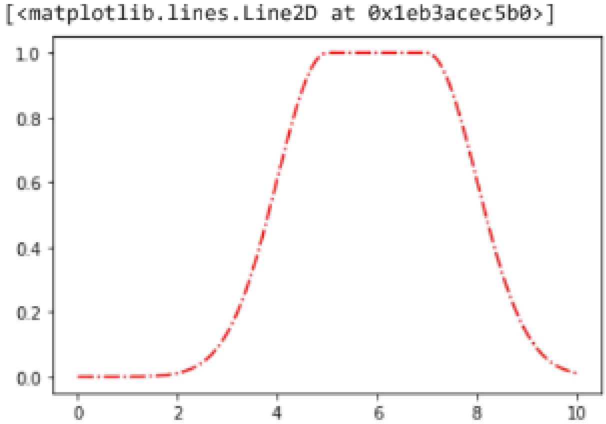
LAB-6: Membership function .

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz

# Generate triangular membership function on range [0, 1]
x = np.arange(0, 10.1, 0.1)
#mfx = fuzz.trimf(x, [0, 5, 10])
#mfx = fuzz.trapmf(x, [0, 5,7,10])
#mfx = fuzz.gaussmf(x, 5,1) # (x,mean,sigma)
mfx = fuzz.gauss2mf(x, 5,1,7,1) # (x,mean1,sigma1,mean2,sigma2)
#mfx = fuzz.gbellmf(x, 5,7,5) # (x,a,b,c)
plt.plot(x, mfx, 'r',ls='-.')
```

OUTPUT:

```
[<matplotlib.lines.Line2D at 0x1eb3acec5b0>]
```



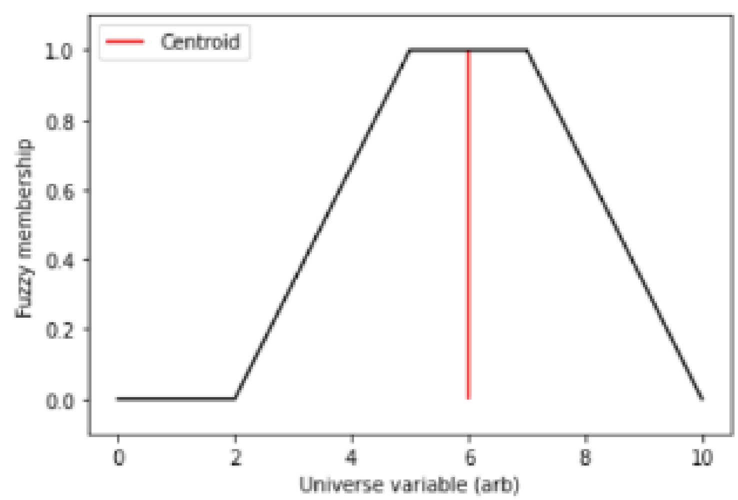
CODE:

LAB-6: Membership function .

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz

# Generate trapezoidal membership function on range [0, 10]
x = np.arange(0, 10.1, 0.1)
#mfx = fuzz.trimf(x, [0, 5, 10])
mfx = fuzz.trapmf(x, [2, 5, 7, 10])

# Defuzzify this membership function five ways
#Controls which defuzzification method will be used.
#* ‘centroid’: Centroid of area * ‘bisector’: bisector of area
#* ‘mom’ : mean of maximum * ‘som’ : min of maximum * ‘lom’ : max of maximum
defuzz_centroid = fuzz.defuzz(x, mfx, 'centroid')  # Same as skfuzzy.centroid
print(defuzz_centroid)
# Collect info for vertical lines
xv=[defuzz_centroid]
ymax = [fuzz.interp_membership(x, mfx, i) for i in xv]
# Display and compare defuzzification results against membership function
plt.plot(x, mfx, 'k')
plt.vlines(defuzz_centroid, 0, ymax, label='Centroid', color='r')
plt.ylabel('Fuzzy membership')
plt.xlabel('Universe variable (arb)')
plt.ylim(-0.1, 1.1)
plt.legend(loc=2)

plt.show()
```

OUTPUT:

```
6.0000000000000036
```



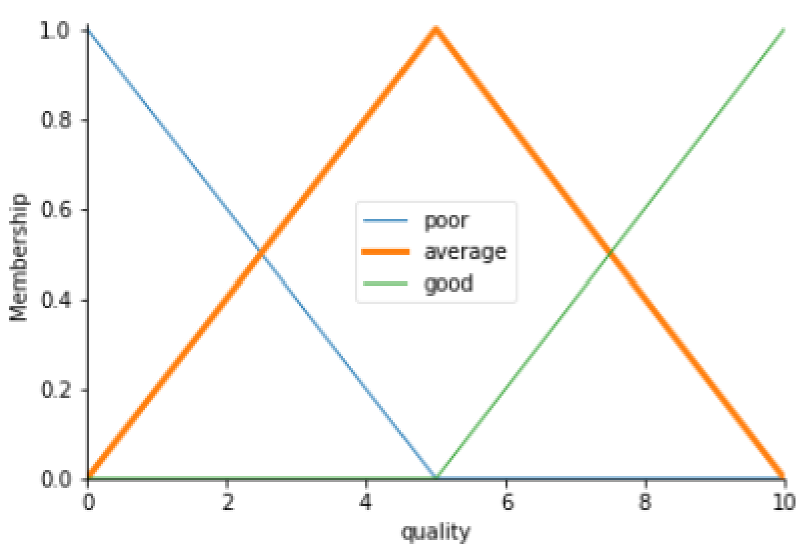
PROGRAM-7: Build a Fuzzy Inference system for restaurant tipping. Consider two input variable Service [0-10] and quality of food [0-10]. Consider tipping as output [0-25] % of bill amount. Consider all other assumptions by own.

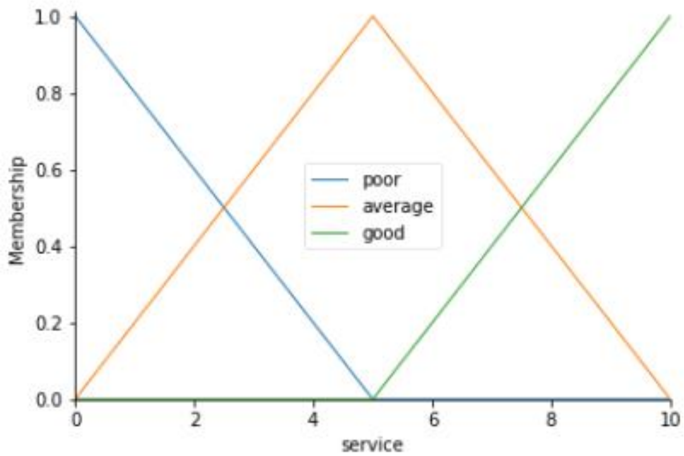
CODE:

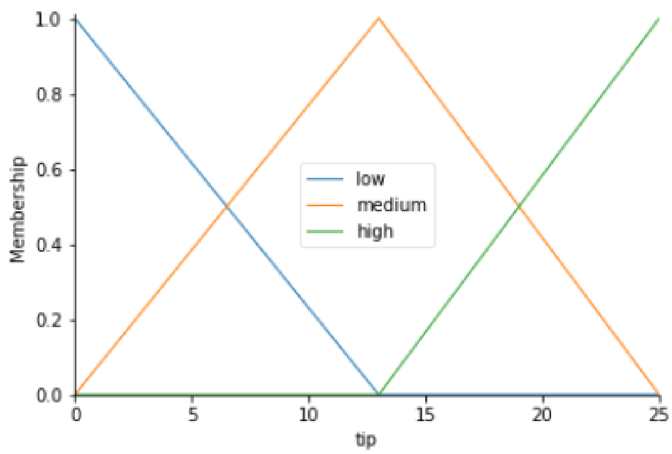
LAB-7:

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz

# New Antecedent/Consequent objects hold universe variables and membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-membership function population is possible with .automf(3, 5, or 7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

# You can see how these look with .view()
quality['average'].view()
service.view()
tip.view()
```

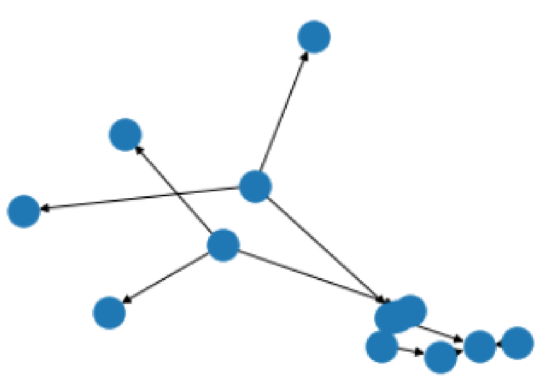






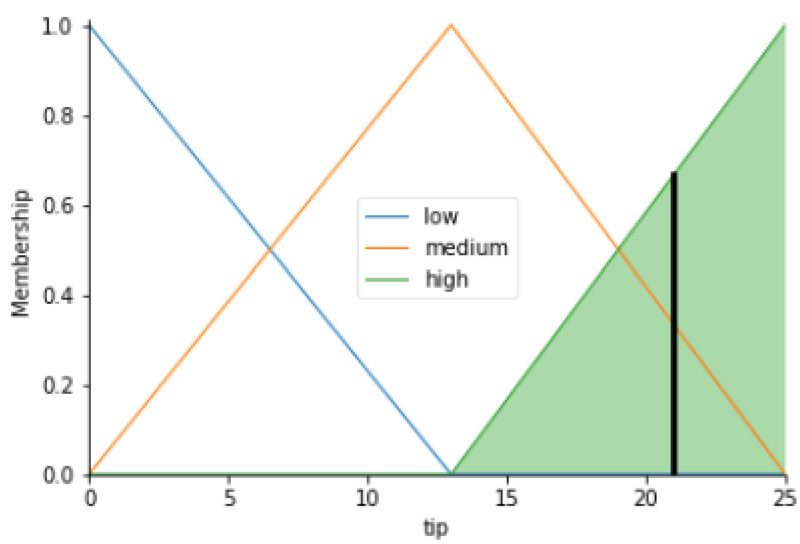

```
##
rule1 = ctrl.Rule(quality['poor'] & service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
rule1.view()
```

```
(<Figure size 432x288 with 1 Axes>, <AxesSubplot:>)
```



```
#
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)

# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
# Note: if you like passing many inputs all at once, use .inputs(dict_of_data)
tipping.input['quality'] = 10
tipping.input['service'] = 10
# Crunch the numbers
tipping.compute()
print (tipping.output['tip'])
tip.view(sim=tipping)
```

OUTPUT:



PROGRAM-8: To implement logic programming, list processing, variable, and constants in PROLOG.

Q1. Implement logic programming, variable, and constants in PROLOG.

logic_prog.pl [modified]

File Edit Browse Compile Prolog Pce Help

logic_prog.pl [modified]

```
apartmentpet(X):-pet(X),small(X).
pet(X):-cat(X).
pet(X):-dog(X).
dog(X):-poodle(X).
small(X):-poodle(X).
poodle(fluffy).
```

OUTPUT:

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/SANJAY/OneDrive/Documents/Prolog/logic_prog.pl compiled 0.00 sec, 6 clauses
?- poodle(X).
X = fluffy.

?- poodle(F).
F = fluffy.

?- poodle(cat).
false.

?- dog(fluffy).
true.

?- cat(fluffy).
ERROR: Unknown procedure: cat/1 (DWIM could not correct goal)
```

Q2. Check a number is in given list.

1 list_member.pl [modified]

File Edit Browse Compile Prolog Pce Help

1 list_member.pl [modified]

```
mem_list(X, [X|_]).
mem_list(X, [_|Y]):-mem_list(X,Y).
```

OUTPUT:

2 length_of_list.pl [modified]

File Edit Browse Compile Prolog Pce Help

1 list_member.pl [modified] | 2 length_of_list.pl [modified]

```
len_list([],0).
len_list([H|T],N):- len_list(T,N1), N is N1+1.
```

Q3. Find the length of list.

```
?- mem_list(10,[1,2,10,5])
|    .
true .

?- mem_list(11,[1,2,10,5]).
false.
```

OUTPUT:

```
?- len_list([1,2,3,4],N).
N = 4.

?- len_list([],N).
N = 0.

?- len_list(['a','b','c'],N).
N = 3.
```

Q4. Add member of list.

3 add elements of list.pl

File Edit Browse Compile Prolog Pce Help

3 add elements of list.pl

```
addlist([],0).
addlist([H|T],S):-addlist(T, S1), S is S1+H.
```

OUTPUT:

```
?- addlist([1,2,3,4,5],N).
N = 15.

?- addlist([1,-2,-3,4,5],N).
N = 5.
```

Q5. Add a number to list.

append a num.pl

File Edit Browse Compile Prolog Pce Help

append a num.pl

```prolog
addnum(X,[],[X]).
addnum(X, L, [X|L]).
```

OUTPUT:

```
?- addnum(5,[1,2,3],L).
L = [5, 1, 2, 3].

?- addnum('a',[1,2,3],L).
L = [a, 1, 2, 3].
```

Q6. Append to list.

4 concatenate.pl

File Edit Browse Compile Prolog Pce Help

4 concatenate.pl

```prolog
append1([],L,L).
append1([H1|T1],L2,[H1|T3]):-append1(T1,L2,T3).
```

OUTPUT:

```
?- append1([1,2],[3,4],L).
L = [1, 2, 3, 4].

?- append1([3,4],[3,4],L).
L = [3, 4, 3, 4].
```

Q7. Write a program to check Head and Tail in List.

7list1.pl [modified]

File Edit Browse Compile Prolog Pce Help

7list1.pl [modified]

```
p([H|T], H, T).
```

OUTPUT:

```
?- p([1,2,3,4],H,T).
H = 1,
T = [2, 3, 4].

?- p([4],H,T).
H = 4,
T = [].

?- p([],H,T).
false.

?- p(['a','b'],H,T).
H = a,
T = [b].
```

PROGRAM-9: Write a recursive program to compute factorial, Fibonacci, tower-of-Hanoi in PROLOG.

Q1. WAP to implement Factorial.

File Edit Browse Compile Prolog Pce Help

fact.pl [modified]

```
factorial(0,1).
factorial(N,R):-N1 is N-1, factorial(N1,R1), R is N*R1.
factorial(N):-factorial(N,R), write(R).
```

OUTPUT:

```
?- fib(5,F).
F = 5 .

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

Q2. WAP to implement Fibonacci Series number.

File Edit Browse Compile Prolog Pce Help

fact.pl [modified] fib.pl

```
fib(1,1).
fib(2,1).
fib(N,F):-N1 is N-1, N2 is N1-1, fib(N1,F1),fib(N2,F2), F is F1+F2.
```

```
?- factorial(5,N).
N = 120 .

?- factorial(10).
3628800
true .

?- factorial(5,120).
true .
```

Q3. WAP to implement tower of Hanoi.

```
File   Edit   Browse   Compile   Prolog   Pce   Help
fact.pl [modified] | fib.pl | tower.pl [modified]

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

```
?- move(3,'A','C','B').
Move top disk from A to C
Move top disk from A to B
Move top disk from C to B
Move top disk from A to C
Move top disk from B to A
Move top disk from B to C
Move top disk from A to C
true .

?- move(1,'A','C','B').
Move top disk from A to C
true .
```

PROGRAM-10: To Implement family tree program in PROLOG.

CODE:

family_tree.pl [modified]

File Edit Browse Compile Prolog Pce Help

family_tree.pl [modified]

```
parent(a,b).
parent(a,c).
parent(b,d).
uncle(X,Y):-parent(P,X),parent(P,Q),parent(Q,Y),X\=Q.
grandpar(X,Y):-parent(X,P),parent(P,Y).
grandchild(X,Y):-parent(P,X),parent(Y,P).
sibling(X,Y):-parent(P,X),parent(P,Y),X\=Y.
```

OUTPUT:

```
?- parent(a,b).
true.

?- parent(c,b).
false.

?- uncle(c,d).
true .

?- grandpar(a,d).
true .

?- grandchild(d,a).
true.

?- grandchild(b,a).
false.

?- sibling(b,c).
true.

?- sibling(a,c).
false.
```