ASSIGNMENT-1

(1) Write a python program to check whether the number is even or odd.

→
```
num = int(input("enter the number : "))
if num%2==0 :
    print(num, " is an even number")
else:
    print(num, " is an odd number")
```

Output :-

```
enter the number : 45
45 is an odd number
```

(2) Write a python program to print the largest of the three numbers.

→
```
a = int(input("Enter a : "))
b = int(input("Enter b : "))
c = int(input("Enter c : "))
if a>b and a>c:
    print(a, " is the greatest")
elif b>c and b>a :
    print(b, " is the greatest")
else:
    print(c, " is the greatest")
```

Output :-

```
Enter a : 23
Enter b : 12
Enter c : 34
34 is the greatest
```

Write a python program to check your score is in grade O, E, A, B, C, D or F.

→
```
marks = int (input ("Enter the marks : "))
if marks > 90 and marks <= 100 :
    print ("Congrats ! You scored grade O ")
elif marks > 80 and marks <= 90 :
    print ("Congrats ! You scored grade E ")
elif marks > 70 and marks <= 80 :
    print ("Congrats ! You scored grade A ")
elif marks > 60 and marks <= 70 :
    print ("Congrats ! You scored grade B ")
elif marks > 50 and marks <= 60 :
    print ("Congrats ! You scored grade C ")
elif marks > 40 and marks <= 50 :
    print ("Congrats ! You scored grade D ")
else :
    print ("Sorry ! You've failed !")
```

Output :-

```
Enter the marks : 88
Congrats ! You scored grade E
```

4) Write a program to print a multiplication table.

→
```
num = int(input("Enter a number : "))
counter = 1
print("The multiplication table of: ", num)
while counter <= 10:
    ans = num * counter
    print(num, 'x', counter, '=', ans)
    counter += 1
```

Output :-

```
Enter a number : 12
The multiplication table of : 12
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
12 x 7 = 84
12 x 8 = 96
12 x 9 = 108
12 x 10 = 120
```

5) Write a python program to check the number is Armstrong number or not.

→
```
~~a = int(input("enter a :~~
~~b = int(input("enter b :~~
~~c = int(input("enter c :~~
num = int(input("Enter a number : "))
sum = 0;
temp = num
while temp > 0:
    digit = temp % 10
    sum = sum + digit**3
    temp = temp // 10
if num == sum:
    print("Armstrong number")
else:
    print("Not an armstrong number")
```

Bonalu 12/8/24

Output :-

```
Enter a number : 153
Armstrong number
```

ASSIGNMENT-2

① Write a python program to take input into list and add all the items and print sum.

→
```
items = []
n = int(input("Enter no. of elements : "))
for i in range(0, n):
    e = int(input()).
    items.append(e)
print(items)
def sum_list(items):
    sum = 0
    for x in items:
        sum += x
    return sum
print("the sum of elements is :: ", sum_list(items))
```

Output :-

```
Enter no. of elements : 3
1
2
-8
[1, 2, -8]
the sum of elements is :: -5
```

Write a python program to remove item from a given list.

→
```
num_set = set([0, 1, 3, 4, 5])
print("Original set : ", num_set)
num_set.pop()
print("\n After removing the first element from the
    said set : ")
print(num_set)
```

Output :-

```
Original set : {0, 1, 3, 4, 5}
After removing the first element from
  the said set :
{1, 3, 4, 5}
```

5) Write a python program to concat 3 dictionary taking input from user.

→
```
user_dict1 = {}
user_dict2 = {}
user_dict3 = {}
num_entries = int(input("Enter the no. of entries you
                  want to add in first dictionary : "))
for i in range(num_entries):
    key = input("Enter key : ")
    value = input("Enter value : ")
    user_dict1[key] = value
print("Dictionary after adding user input : ", user_dict1)
num_entries = int(input("Enter the no. of entries you
                  want to add in second dictionary : "))
for i in range(num_entries):
    key = input("Enter key : ")
    value = input("Enter value : ")
    user_dict2[key] = value
print("Dictionary after adding user input : ", user_dict2)
num_entries = int(input("Enter the no. of entries you
                  want to add in third dictionary : "))
for i in range(num_entries):
    key = input("Enter key : ")
    value = input("Enter value : ")
    user_dict3[key] = value
print("Dictionary after adding user input : ", user_dict3)
user_dic4 = {}
for d in (user_dict1, user_dict2, user_dict3):
    user_dic4.update(d)
print(user_dic4)
```

Output :-

```
Enter the no. of entries you want to add in
  first dictionary : 2
Enter key : name
Enter value : Rishika Majumdar
Enter key : roll
Enter value : 403
Dictionary after adding user input :
{'name': 'Rishika Majumdar', 'roll': '403'}
Enter the no. of entries you want to add in
second dictionary : 2
Enter key : area
Enter value : Garia
```

```
Enter key: city
Enter value : Kolkata
Enter key: state
Enter value : West Bengal
Dictionary after adding user input : {'area': 'garia',
    'city': 'Kolkata', 'state': 'West Bengal'}
Enter the no. of entries you want to add in the
third dictionary: 2
 Enter key : status
 Enter value : unmarried
 Enter key : university
 Enter value : Brainware
Dictionary after adding user input : {'status':
    'unmarried', 'university': 'Brainware'}
{'name': 'Aishika Majumdar', 'roll': '403', 'area': 'garia',
    'city': 'Kolkata', 'state': 'West Bengal', 'status':
    'unmarried', 'university': 'Brainware'}
```

1) Write a python program to convert a list into tuple.

→

```
items = []
n = int(input("Enter no. of elements: "))
for i in range(0, n):
    e = int(input())
    items.append(e)
print(items)
tuplex = tuple(items)
print(tuplex)
```

Output:

```
Enter no. of elements: 4
1
2
3
4
[1, 2, 3, 4]
(1, 2, 3, 4)
```

(5) Write a python program to count number of vowels in a string, taking input from a user.

→
```
def vowel_count (str):
    count = 0
    vowel = set ("aeiouAEIOU")
    for i in str:
        if i in vowel:
            count = count+1
    print ("No. of vowels :", count)
str = input ("Enter a sentence :: ")
vowel_count (str)
```

Output :

```
Enter a sentence :: My name is Aishika
No. of vowels : 7
```

Basak
19/8/24

ASSIGNMENT-3

① Write a program in Python for BFS implementation

```
from collections import deque
def bfs (graph, start_node):
    queue = deque ([start_node])
    visited = set ([start_node])
    while queue:
        current_node = queue.popleft ()
        print (current_node, end = ' ')
        for neighbour in graph [current_node]:
            if neighbour not in visited:
                queue.append (neighbour)
                visited.add (neighbour)

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B']
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
bfs (graph, 'A')
```

Output :-

A B C D E F

Write a python program for DFS implementati

②
```
def dfs (graph, start_node, visited = None):
    if visited is None:
        visited = set ()
    print (start_node, end = ' ')
    visited.add (start_node)
    for neighbours in graph [start_node]:
        if neighbour not in visited:
            dfs (graph, neighbour, visited)

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
    }
dfs (graph, 'A')
```

Output :-

A B D E F C

ASSIGNMENT-4

rite a python program to implement A* searching algorithm.

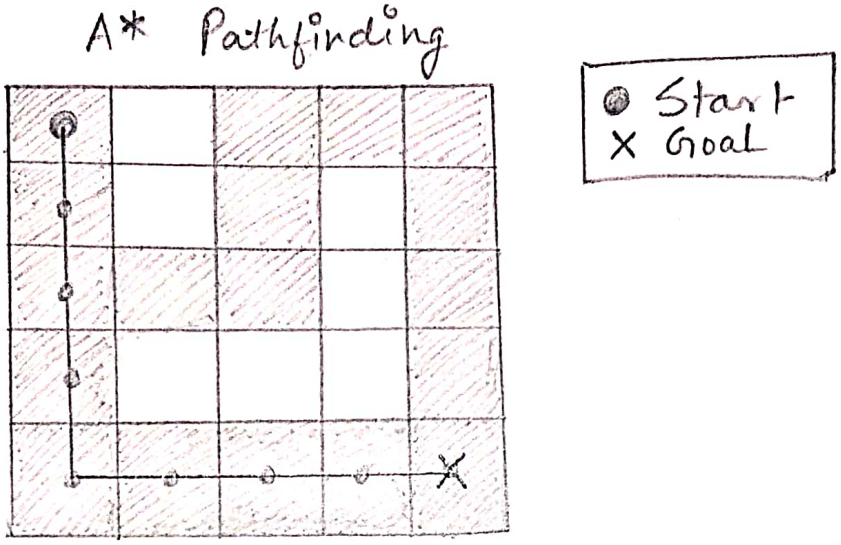
```
pont heapq
mport matplotlib.pyplot as plt
nport numpy as np
f heuristic(a, b):
    return abs(a[0] - b[0] + abs(a[1] - b[1])
lef astar_search(grid, start, goal):
    open_set = []
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {start : 0}
    f_score = {start : heuristic(start, goal)}
    while open_set :
        current = heapq.heappop(open_set)[1]
        if current == goal :
            path = []
            while current in came_from :
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]
        neighbours = [(current[0] + x, current[1] + y)
                      for x, y in [(0,1), (1,0), (0,-1),
                                   (-1,0)]]
        for neighbour in neighbours:
            if 0 <= neighbour[0] < len(grid) and
                    0 <= neighbour[1] < len(grid[0]):
                if grid[neighbour[0]][neighbour[1]]
                                        == 1:
                    continue
                tentative_g_score = gscore[current]+1
                if neighbour not in g_score or
                        tentative_g_score < g_score[neighbour]:
                    came_from[neighbour] = current
                    g_score[neighbour] =
                                tentative_g_score
```

```
            f_score[neighbour] = tentative_g_score + heuristic
                                 (neighbour, goal)
            heapq.heappush(open_set, (f_score[neighbour],
                                 neighbour))
    return None

grid = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]
start = (0, 0)
goal = (4, 4)
path = astar_search(grid, start, goal)
def visualize_path(grid, path, start, goal):
    grid_array = np.array(grid)
    fig, ax = plt.subplots()
    ax.imshow(grid_array, cmap = plt.cm.Pastel1)
    if path:
        path_x, path_y = zip(*path)
        ax.plot(path_y, path_x, color='red',
                linewidth = 2, marker = '0')
    ax.scatter(start[1], start[0], marker = "0",
               color = "green", s = 100, label = "start")
    ax.scatter(goal[1], goal[0], marker = "x",
               color = "blue", s = 100, label = "goal")
    ax.set_xticks(np.arrange(-0.5, len(grid[0], 1), minor=
                  True)
    ax.set_yticks(np.arrange(-0.5, len(grid), 1), minor=
                  True)
    ax.grid(which = "minor", color = "black", linestyle = '-',
            linewidth = 2)
    ax.tick_para(which = "both", bottom = False, left=
                 False, labelbottom = False, labelleft=
                 False)
```

```
plt.title("A* Pathfinding")
plt.legend()
plt.show()
visualize_path(grid, path, start, goal)
```

Output :-



Passed
9/9/24

ASSIGNMENT-5

1) Write a Python program to implement AO* Searching Algorithm.

```
import networkx as nx
import matplotlib.pyplot as plt
class Graph:
    def __init__(self):
        self.graph = {}
        self.heuristics = {}
    def add_node(self, node, h=0):
        self.graph[node] = []
        self.heuristics[node] = h
    def add_edge(self, parent, child, cost):
        self.graph[parent].append((child, cost))

def AO_star(graph, start):
    open_list = {start: graph.heuristics[start]}
    closed_list = 0
    solution_graph = {start: []}
    def expand_node(node):
        children = graph.graph[node]
        min_cost = float('inf')
        best_path = []
        for (child, cost) in children:
            total_cost = cost + graph.heuristics[child]
            if total_cost < min_cost:
                min_cost = total_cost
                best_path = [child]
        closed_list[node] = min_cost
        solution_graph[node] = best_path
        for n in best_path:
            open_list[n] = graph.heuristics[n]
    while open_list:
        node = min(open_list, key = open_list.get)
        expand_node(node)
        open_list.pop(node)
    return solution_graph, closed_list
def plot_graph(graph, solution_graph):
    G = nx.DiGraph()
```

```
for node in graph.graph:
    for (child, cost) in graph.graph [node]:
        G.add_edge (node, child, weight = cost)
pos = nx.spring_layout (G)
labels = nx.get_edge_attributes (G, 'weight')
plt.figure (figsize = (10, 7))
nx.draw (G, pos, with_labels = True, node_colour =
        'lightblue', node_size = 2000, font_size = 15,
        font_weight = 'bold', edge_color = 'gray')
nx.draw_networkx_edge_labels (G, pos, edge_labels =
                              labels)
solution_edges = [(k, v[0]) for k, v in solution_graph.
                                items () if v]
nx.draw_networkx_edges (G, pos, edgelist = solution_
    edges, edge_color = 'red', width = 2)
plt.title ("AO * Algorithm Solution Path")
plt.show ()
graph = Graph ()
graph.add_node ('A', 10)
graph.add_node ('B', 8)
graph.add_node ('C', 5)
graph.add_node ('D', 7)
graph.add_node ('E', 3)
graph.add_node ('F', 0)
graph.add_edge ('A', 'B', 1)
graph.add_edge ('A', 'C', 2)
graph.add_edge ('B', 'D', 5)
graph.add_edge ('B', 'E', 3)
graph.add_edge ('C', 'F', 1)
solution_graph, costs = AO_star (graph, 'A')
print ("Solution Graph:", solution_graph).
```

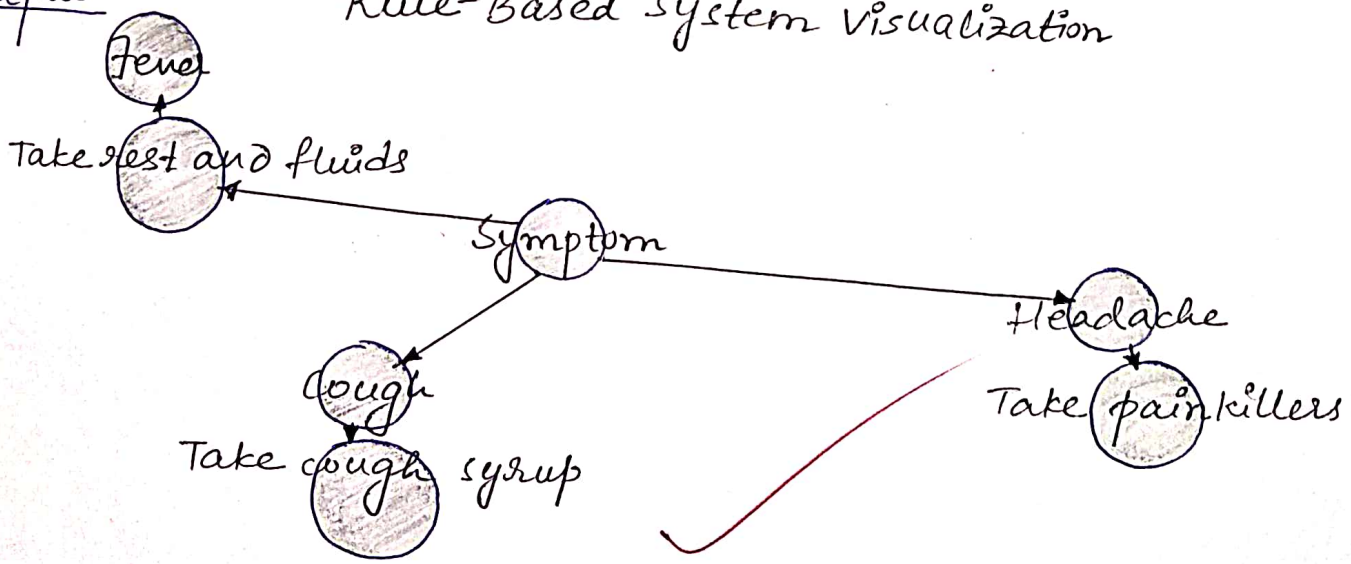
Bas
9/9/24

ASSIGNMENT - 6

1) Write a python program to implement rule-based system.

```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
rules = [
    ("Symptom", "Fever"),
    ("Symptom", "Cough"),
    ("Symptom", "Headache"),
    ("Fever", "Take rest and fluids"),
    ("Cough", "Take cough syrup"),
    ("Headache", "Take painkillers")
    ]
G.add_edges_from(rules)
plt.figure(figsize = (8, 6))
nx.draw(G, with_labels = True, node_color = "lightgreen",
        node_size = 3000, font_size = 10, font_weight = "bold")
plt.title("Rule-Based System Visualization")
plt.show()
```

Output :-

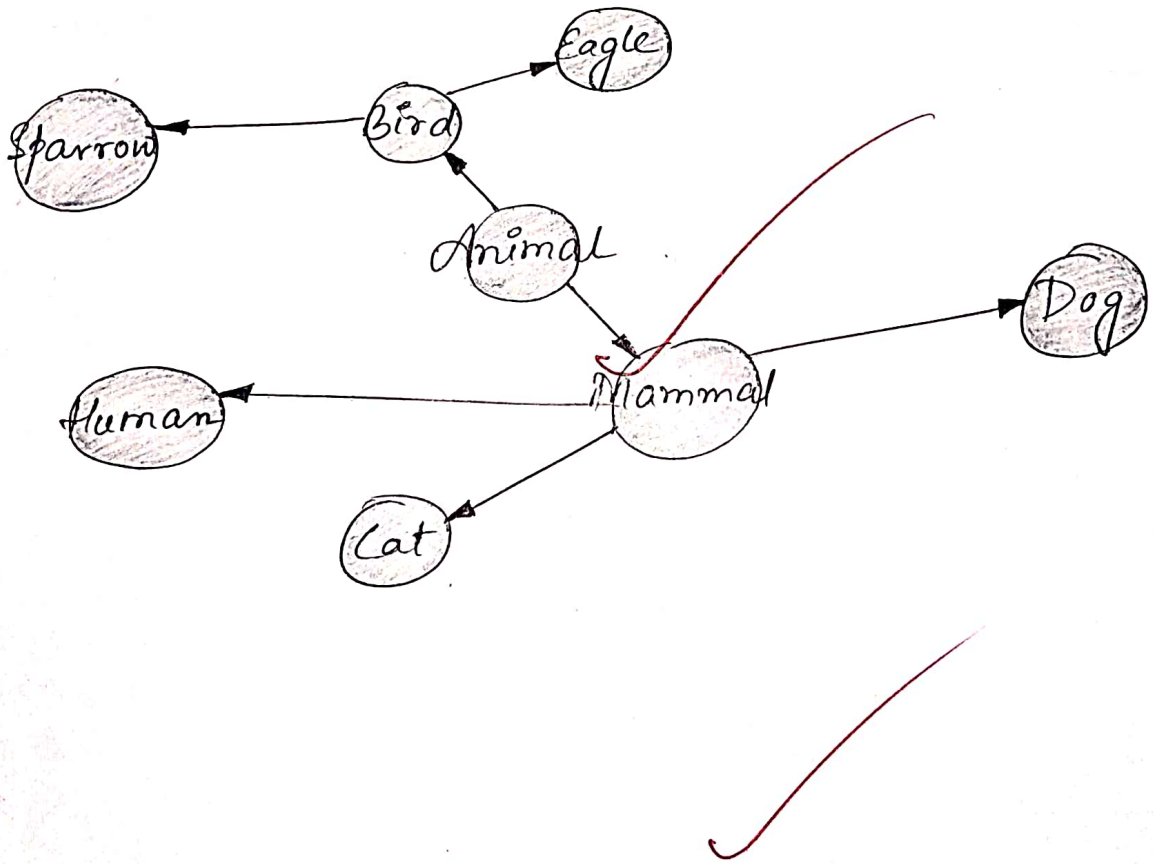


② Write a python program to visualize semantic network.

Ans:-

```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
G.add_edges_from([
    ('Animal', 'Mammal'), ('Mammal', 'cat'),
    ('Mammal', 'Dog'), ('Animal', 'Bird'), ('Bird',
    'Sparrow'), ('Bird', 'Eagle'), ('Mammal', 'Human')])
plt.figure(figsize=(8,6))
nx.draw(G, with_labels=True, node_color='lightblue',
        node_size=3000, font_size=10, font_weight="bold")
plt.title("Semantic Network Visualization")
plt.show()
```

Output :-

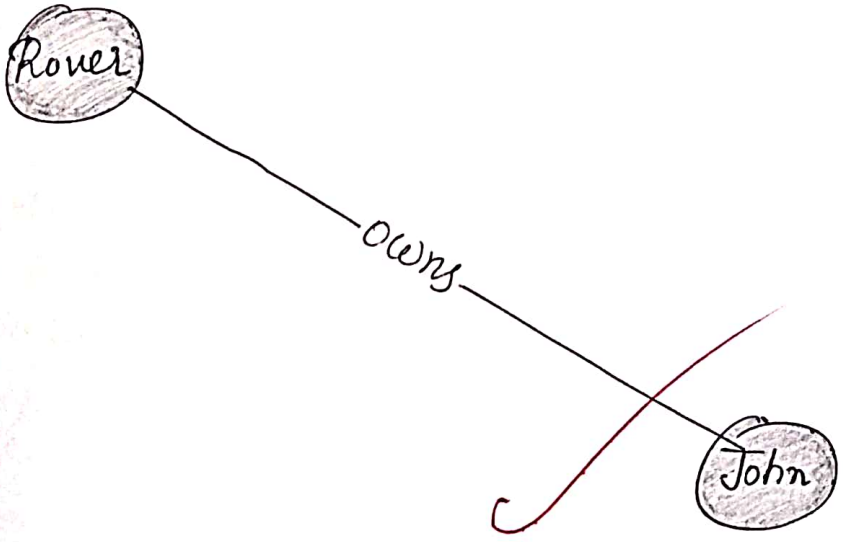


(3) Write a python program to visualize conceptual graphs.

Ans:-

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.Graph()
G.add_node("John", label="Person")
G.add_node("Rover", label="Dog")
G.add_edge("John", "Rover", relationship="owns")
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels = True, node_size=2000,
        node_color='lightblue', font_size=12,
        font_weight='bold')
edge_labels = nx.get_edge_attributes(G, 'relationship')
nx.draw_networkx_edge_labels(G, pos, edge_labels=
                             edge_labels)
plt.show()
```

Output:-

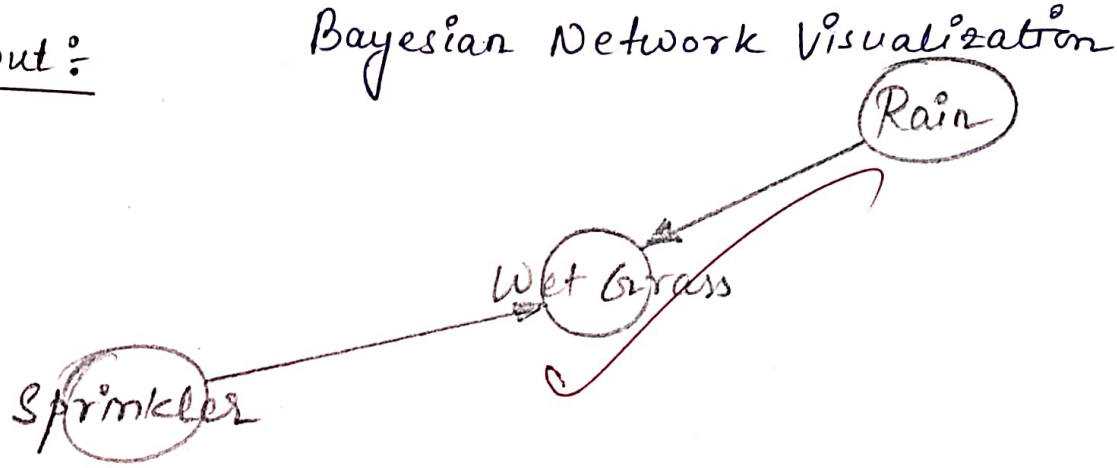


ASSIGNMENT-7

① Write a Python program to visualize Probabilistic Reasoning using networkx.

→
```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
G.add_edges_from([('Rain', 'Wet Grass'), ('Sprinkler',
                                          'Wet Grass')])
plt.figure(figsize = (8,6))
nx.draw(G, with_labels = True, node_color = 'lightblue',
        node_size = 3000, font_size = 10, font_weight = 'bold')
plt.title("Bayesian Network Visualization")
plt.show()
```

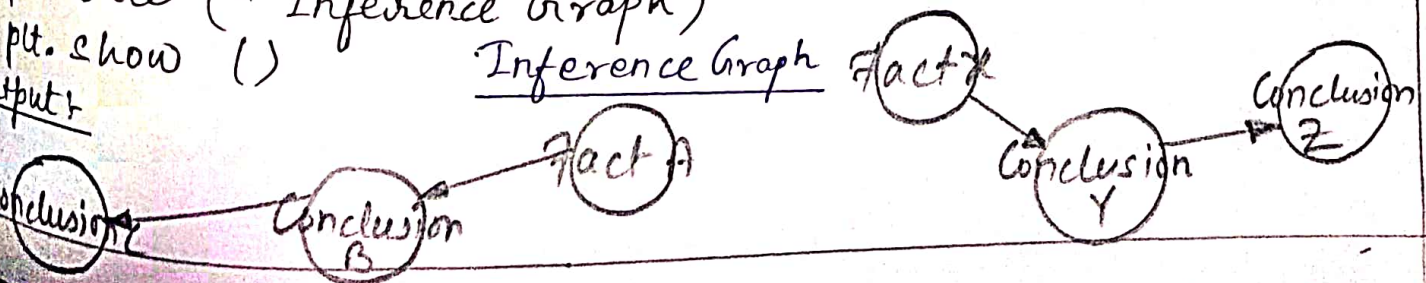
Output:



Write a Python program to visualize inference Engine using networkx

→
```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
G.add_edges_from([("Fact A", "Conclusion B"), ("Conclusion B",
                  "Conclusion C"), ("Fact X", "Conclusion Y"),
                  ("Conclusion Y", "Conclusion Z")])
plt.figure(figure = (8,6))
nx.draw(G, with_labels = True, node_color = 'lightcoral', node_size
        = 3000, font_size = 10, font_weight = 'bold')
plt.title("Inference Graph")
plt.show()
```

Output:

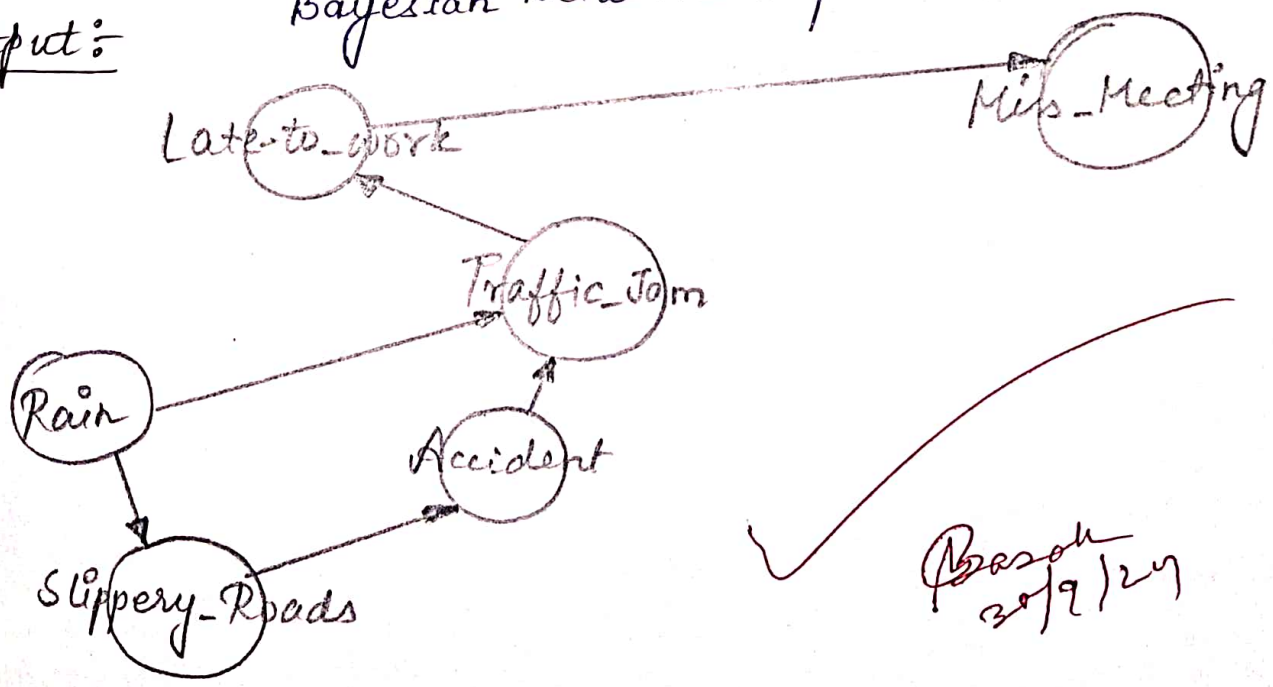


③ Write a Python program to represent probabilistic dependencies among variables.

→
```
!pip install networkx matplotlib pgmpy
import networkx as nx
import matplotlib.pyplot as plt
from pgmpy.models import BayesianNetwork

edges = [('Rain', 'Traffic_Jam'), ('Accident', 'Traffic_Jam'),
         ('Traffic_Jam', 'Late_to_work'), ('Late_to_Work',
         'Miss_Meeting'), ('Rain', 'Slippery_Roads'),
         ('Slippery_Roads', 'Accident')]
bayesian_net = BayesianNetwork(edges)
G = nx.DiGraph()
G.add_edges_from(edges)
plt.figure(figsize = (10, 6))
pos = nx.spring_layout(G, seed = 42)
nx.draw(G, pos, with_labels = True, node_color = 'lightgreen',
        node_size = 3000, font_size = 10, font_weight =
        'bold', arrows = True)
plt.title("Bayesian Network Graphical Visualization")
plt.show()
```

Output:



Basak
30/9/24