Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./Miss Parth Sandeep Dabholkar of Computer Engg. Department, Semester VI with Roll No. 2103032 has completed a course of the necessary experiments in the subject Artificial Intelligence under my supervision in the **Thadomal Shahani Engineering College** Laboratory in the year 2023 - 2024

Teacher In-Charge

Head of the Department

Date 12/04/24

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

Experiment No. 1

Title: Case study on AI applications published in IEEE/ACM/Springer/Elsevier or any prominent journal

Introduction

The case study mainly focus on the negative impact of human activities on the environment, particularly in relation to air pollution. It discusses the various forms of pollution, such as pollution of the seas, global warming, acid rain, and smog, and how these affect natural processes and environments. Furthermore, it emphasizes the invisible nature of air pollution and how it poses a significant risk to human health and ecological balance. The case study also touches on the sources of air pollution and the challenges in predicting and measuring pollutant distribution, highlighting the importance of accurate measurement through remote sensing.

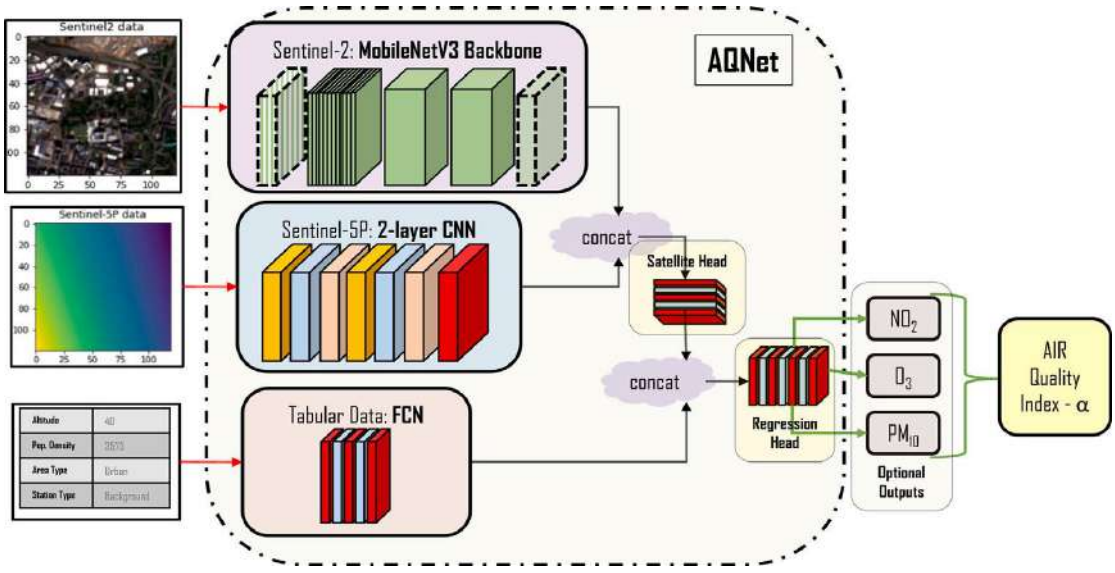
Domain

The domain for the case study is environmental science and technology, with a specific focus on air quality in urban environments and its impact on human health and the environment.

Problem identified

The case study identifies the negative impact of human activity on the environment through various forms of pollution, particularly air pollution, which affects natural processes and environments. It specifically mentions pollution of the seas, global warming, acid rain, and smog as key manifestations of human impact. The sources of air pollution, such as industrial processes, vehicle emissions, and other catalytic reactions, have been highlighted. Additionally, it discusses the challenge of accurately measuring air pollutant distribution, especially within cities, and emphasizes the role of remote sensing in providing accurate tools for measurement. The identified problems can be categorized as the impact of human activity on the environment through air pollution, the challenges in predicting and measuring air pollutant distribution, and the need for accurate measurement tools through remote sensing

Algorithmic Methodologies Applied



AQNet model Architecture. Named as AQNet-single if it returns only one of the three pollutants as output.

The algorithm mentioned in the case study is the AQNet, a multimodal AI architecture designed for air quality measurement using a combination of satellite imagery and tabular location information. The algorithm leverages machine learning (ML) and deep learning techniques to predict air quality metrics and pollutant concentrations. It utilizes different architectures to process and fuse from satellite imagery, tabular data, and ground concentration measurements.

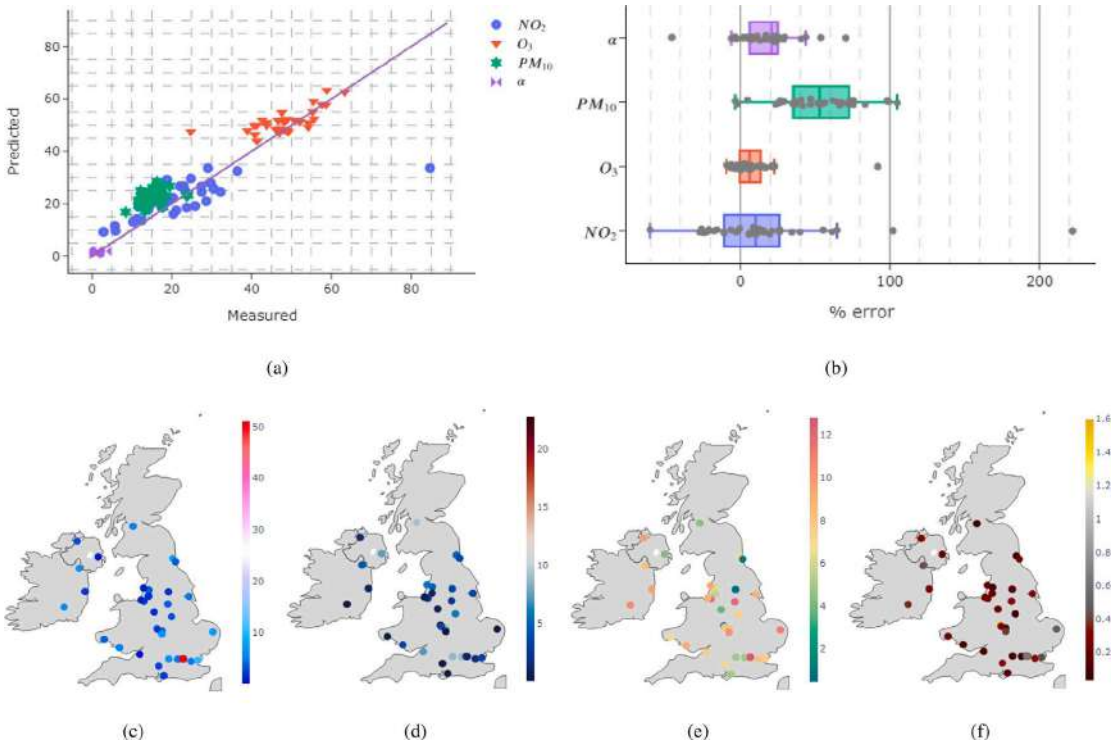
The key components of the algorithm include:

1. Satellite Data Processing: The algorithm preprocesses the satellite data by removing clouds and negative weather conditions. It then maps the data to a 10x10 km grid over Europe and matches the time of air quality measurements on the ground to the nearest time when the satellite passed over the monitoring station. This results in a high-resolution (10m) of the area.

2. Tabular Data Backbone: A 2-layer fully connected neural network (FCN) with ReLu activation functions processes the tabular data. This tabular data includes information such as altitude, population density, and categorical features indicating the type of monitoring (e.g., rural, suburban, urban; traffic, industrial, or background monitoring station). The network creates 32 features, which are then fused with the satellite data in the regression head.

3. Regression Head: This part of the algorithm is designed to create three outputs for the pollutants NO2, O3, and PM10. It allows for the simultaneous prediction of all three pollutants or individual predictions based on specific needs.

Solution



(a) Regression plots of true and predicted values of each pollutant and air quality - α. (b) Boxplots of % error between the measured and predicted values of each pollutant.(c)–(f) Absolute error scatter maps for NO2, O3, PM10 and air quality metric - α , respectively.

The case study discusses a model's predictions, noting an average overestimation of around 20% for all pollutants. It emphasizes that while O3 (ozone) predictions are closest to parity, there are significant errors in predicting NO2 and PM10 concentrations. The overestimation of PM10 predictions is suggested to be due to the small value range of PM10 measurements in the dataset, leading to a high percentage error.

Pros of the model mentioned in the case study include:

- Effective handling of large amounts of input data for predictive insight into environmental remote sensing problems.
- Highlighted potential for creating maps of PM2.5 distributions around Great Britain using random forests of decision tree algorithms.
- Discussion of using Sentinel 2 bands to create useful indices for air pollution prediction, indicating potential for improvement.

In summary, the model shows potential in providing predictive insights into air pollutant distributions, particularly through advanced statistical and machine learning techniques. It also demonstrates promise in leveraging satellite data for valuable information in environmental remote sensing problems.

Areas for Future Exploration

The case study outlines potential areas for future work and improvements in the field of environmental remote sensing and machine learning algorithms. It suggests several directions for research and development. Here are some of the key points from the case study:

1. Incorporating Meteorology: The case study highlights the importance of considering meteorological factors as inputs to the predictive models. The impact of weather conditions, such as rainfall, humidity, temperature, wind speed, and visibility on air pollutant transport, is significant. Therefore, future work could involve integrating meteorological data to enhance the predictivity of the models.

2. Geographical Information: The case study suggests that models could benefit from the inclusion of geographical information beyond the sovereign state where each pollutant monitoring station is located. This could involve encoding location features, such as latitude and longitude, to better predict pollution levels.

3. Expanding Feature Set: It is proposed that the models should include a broader range of features relating to land use and land cover (LULC) characteristics, potentially enhancing predictivity. This could involve explicit statements of land cover concentrations and the creation of useful indices for the purpose of air pollution prediction.

4. Utilization of Sentinel 2 Bands: The case study suggests considering the utilization of Sentinel 2 bands to create useful indices for air pollution prediction. Developed indices from the Sentinel 2 data might be incorporated into the tabular data to improve model performance.

5. Multi-Temporal Samples: Future research could address the challenge of sourcing suitable multi-temporal samples to run the model post-development, which is crucial for multimodal techniques in remote sensing.

Overall, the case study identifies the need to consider meteorological and geographical factors, expand the feature set, and explore the potential of satellite data for creating indices to improve the predictivity of machine learning models in environmental remote sensing. These directions for future work aim to enhance the accuracy and reliability of air quality prediction models.

Conclusion

The case study presents a comprehensive investigation of the AQNet algorithm for air quality measurement, utilizing satellite imagery and tabular location information. It introduces an air quality index and 3-pollutant dataset aimed at predicting air quality across different regions. The study delves into the influence of binary features, such as area and station type, on average pollutant concentrations, shedding light on the critical role of geographic and environmental factors in air quality. Moreover, the analysis acknowledges the potential impact of COVID-related lockdowns on air pollution reduction and highlights the need for further investigation into this area.

The case study also outlines potential areas for future work, focusing on the inclusion of meteorological and geographical data, expanding feature sets, and exploring the potential of satellite data for creating useful indices. The limitations and potential areas for improvement, such as the consideration of meteorological and geographical factors, suggest an ongoing need for refining air quality prediction models. Furthermore, the study critically examines the AQNet's use of Convolutional Neural Networks and raises the question of its suitability for the task. The case study concludes by emphasizing the ongoing pursuit of enhancements and the investigation of multi-task learning approaches for better performance in future releases of the AQNet.

In summary, the case study provides a detailed and nuanced exploration of the AQNet algorithm's development, its potential limitations, and future research directions, thus contributing valuable insights to the field of air quality prediction through remote sensing and machine learning.

EXPERIMENT NO: 2

Title: Implenting Depth First Search algorithm using python.

Problem statement: Consider a weighted connected graph with start node S and goal node G.

i) Apply DFS algorithm and show how statespace tree will be created while searching a path from initial node to goal node G.

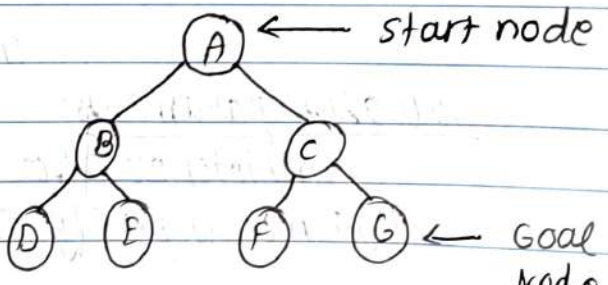
ii) Show the contents of Open and Closed List after each iteration.

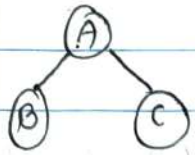
Theory: Uniformed search is a class of general-purpose search algorithms which opereates in brute force-way. Uniformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called a blind search.

Following are the types: i) BFS, ii) DFS, iii) DLS iv) DFDD v) UCS

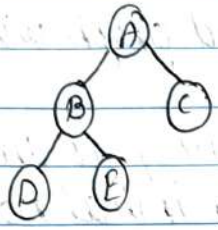
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is that, unlike trees, graphs may contain cycles (a node maybe visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

Let us take an example to understand DFS on graph

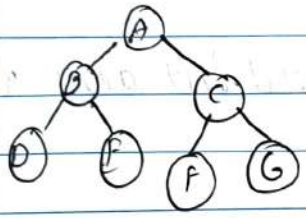




visited : [A] stack : [B, C] pop



visited [A | B] stack : [D, E, C] pop [E, C] pop [C] pop



visited: [A | B | D | E | C] stack [F, G] pop [G] pop

Finally : A → B → D → E → C → F → G

Algorithm : DFS (G, v) {
stack S = { } ;
for each vertex v, set visited [v] = false ;
push S, v ;
while (S is not empty) do
v = pop S ;
if (not visited [v]) then
visited [v] = True
for each unvisited neighbour w of v
push S, w ;
end if
end while } END DFS()

A+

21/3/24

Evaluation parameters:

(i) Completeness : Not complete
(ii) Time Complexity : $O(b^m)$
iii) Space : $O(b \times m)$
iv) Optimality : Non optimal.

DFS:

```
visited = set()  # Set to keep track of visited nodes of graph.

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



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

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

graph2 = {
    '0': ['1', '9'],
    '1': ['2'],
    '2': ['0', '3'],
    '9': ['3'],
    '3': ['3']
}
```

```
print("\nFollowing is the Depth-First Search for graph 2")
dfs(visited, graph2, '0')
```

DLS:

```
class Graph:
    def __init__(self):
        self.adjDict = dict()
        self.start_node = "S"
        self.goal_node = "G"
        self.path = list()

    def add_edge(self, u, v):
        if u not in self.adjDict:
            self.adjDict[u] = list()
        self.adjDict[u].append(v)

    def print(self):
        for key in self.adjDict:
            print(key)
            print(self.adjDict[key])

    def set_start_node(self, node):
```

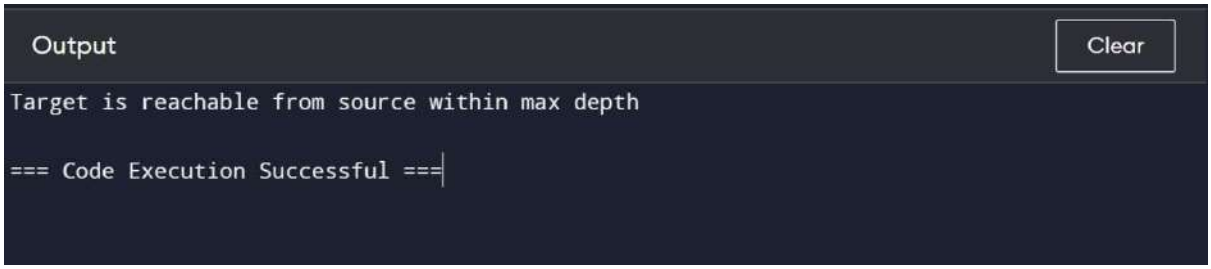
```
        self.start_node = node

    def set_goal_node(self, node):
        self.goal_node = node

    def dls(self, depth_limit):
        visited = list()
        path = list()
        found = self.dls_helper(visited, path, self.start_node, 0, depth_limit)
        return found

    def dls_helper(self, visited, path, node, current_depth, depth_limit):
        if node in visited:
            return False
        if current_depth > depth_limit:
            return False
        visited.append(node)
        path.append(node)
        if node == self.goal_node:
            self.path = path.copy()
            return True

        children = []
        if node in self.adjDict:
            children = self.adjDict[node]
        for child in children:
            found = self.dls_helper(
                visited, path, child, current_depth+1, depth_limit)
            if found:
                return True
```

```
        path.pop()
        return False


def main():
    print("enter input")
    graph = Graph()
    while True:
        tokens = input().split()
        parent = tokens[0]
        if parent == "-1":
            break
        children = tokens[1:]
        for child in children:
            graph.add_edge(parent, child)
    # graph.print()
    start_node = input("enter start node: ")
    goal_node = input("enter goal node: ")
    depth_limit = int(input("enter depth limit: "))
    graph.set_start_node(start_node)
    graph.set_goal_node(goal_node)
    found = graph.dls(depth_limit)
    if found:
        print("found!!")
        print(graph.path)
    else:
        print("not found")


main()
```

DFID:

```
from collections import defaultdict

# list representation
class Graph:

    def __init__(self,vertices):
        self.V = vertices
        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def DLS(self,src,target,maxDepth):
        if src == target : return True

        if maxDepth <= 0 : return False

        for i in self.graph[src]:
            if(self.DLS(i,target,maxDepth-1)):
```

```
                                    return True
            return False

    def IDDFS(self,src, target, maxDepth):

            for i in range(maxDepth):
                    if (self.DLS(src, target, i)):
                            return True
            return False

# Create a graph given in the above diagram
g = Graph (7)
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(1, 4)
g.addEdge(2, 5)
g.addEdge(2, 6)

target = 6; maxDepth = 3; src = 0

if g.IDDFS(src, target, maxDepth) == True:
        print ("Target is reachable from source " +
                "within max depth")
else :
        print ("Target is NOT reachable from source " +
                "within max depth")
```


Output
Clear
Target is reachable from source within max depth
=== Code Execution Successful ===

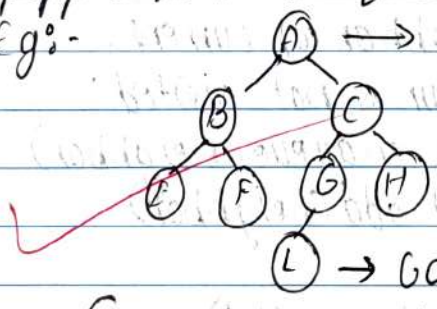
EXPERIMENT NO: 3

Title: Implementing Breadth First Search Algorithm using python.

Problem statement: Consider a weighted connected graph with start node S and goal node G.
i) Apply BFS and show how state space tree will be created while searching a path from initial to goal node.
ii) Show the contents of OpenList and ClosedList after each iteration.

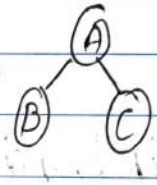
Theory: The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at next depth level.
Starting from the root, all nodes at a particular level are visited List and then the nodes of the next level are traversed till all nodes are visited. To do this a queue is used. All unvisited nodes are pushed in queue and nodes of current level are marked visited and popped from the queue.

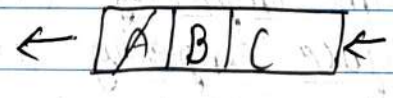
Eg:-



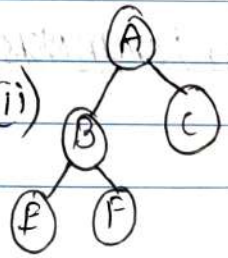
source: A
Goal: L

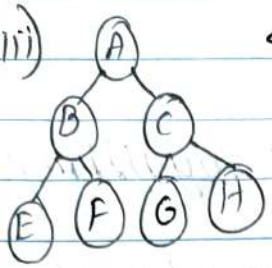
(i)



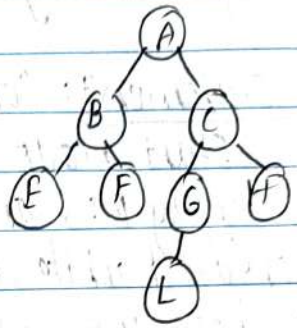


(ii)



iii) 

(iv) 

(v) 

Final Path : A → B → C → E → F → G → H → L

Algorithm :
```
1. BFS(Graph, start):
2.     queue = Queue()  // keep the track of visited
3.     visited = set()  nodes and initialize empty queue
       // To start enqueue start node and mark visited.
4.     queue.enqueue(start)
5.     visited.add(start)
6.     while queue is not empty:
           // Dequeue a node from the queue
7.         current = queue.dequeue()
8.         process(current)
9.         for each neighbor of current:
10.            if neighbour is not visited:
11.                queue.enqueue(neighbor)
12.                visited.add(neighbor)
```

Evaluation Parameters:
1. Time Complexity : $O(b^d)$
2. Space Complexity : $O(b^d)$
3. Completeness : complete if branch factor is finite.
4. Optimality : optimal if each branch has same cost.

```
import heapq

class Graph:
    def __init__(self):
        self.vertices = {}

    def add_edge(self, u, v, weight):
        if u not in self.vertices:
            self.vertices[u] = []
        if v not in self.vertices:
            self.vertices[v] = []
        self.vertices[u].append((v, weight))
        self.vertices[v].append((u, weight))

    def bfs(self, start, goal):
        visited = set()
        queue = [[start]]
        if start == goal:
            return "Start is the goal!"
        while queue:
            path = queue.pop(0)
            node = path[-1]
            if node not in visited:
                neighbors = self.vertices[node]
                for neighbor, _ in neighbors:
                    new_path = list(path)
                    new_path.append(neighbor)
                    queue.append(new_path)
                    if neighbor == goal:
```

```
                    return new_path
                visited.add(node)
        return "No path found"

    def ucs(self, start, goal):
        visited = set()
        queue = [(0, start, [])]
        while queue:
            cost, node, path = heapq.heappop(queue)
            if node not in visited:
                path = path + [node]
                if node == goal:
                    return path
                visited.add(node)
                for neighbor, weight in self.vertices[node]:
                    if neighbor not in visited:
                        heapq.heappush(queue, (cost + weight, neighbor, path))
        return "No path found"


def menu():
    print("Choose Algorithm:")
    print("1. Breadth First Search (BFS)")
    print("2. Uniform Cost Search (UCS)")
    print("3. Exit")


if __name__ == "__main__":
    g = Graph()
```

```
g.add_edge('A', 'B', 4)
g.add_edge('A', 'C', 2)
g.add_edge('B', 'C', 5)
g.add_edge('B', 'D', 10)
g.add_edge('C', 'D', 3)

while True:
    menu()
    choice = int(input("Enter choice: "))
    if choice == 1:
        start = input("Enter start node: ")
        goal = input("Enter goal node: ")
        print("Path using BFS:", g.bfs(start, goal))
    elif choice == 2:
        start = input("Enter start node: ")
        goal = input("Enter goal node: ")
        print("Path using UCS:", g.ucs(start, goal))
    elif choice == 3:
        print("Exiting...")
        break
    else:
        print("Invalid choice. Please try again.")
```

Output Clear

```
Choose Algorithm:
1. Breadth First Search (BFS)
2. Uniform Cost Search (UCS)
3. Exit
Enter choice: 1
Enter start node: A
Enter goal node: D
Path using BFS: ['A', 'B', 'D']
Choose Algorithm:
1. Breadth First Search (BFS)
2. Uniform Cost Search (UCS)
3. Exit
Enter choice: 2
Enter start node: A
Enter goal node: D
Path using UCS: ['A', 'C', 'D']
Choose Algorithm:
1. Breadth First Search (BFS)
2. Uniform Cost Search (UCS)
3. Exit
Enter choice: 3
Exiting...

=== Code Execution Successful ===
```

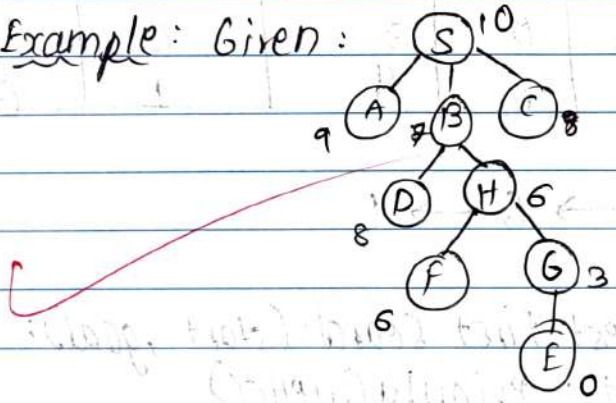
EXPERIMENT NO: 4

AIM: Implement Greedy Best First search Algorithm.

THEORY: Greedy Best First Search is an AI search algorithm that attempts to find the most promising path from a given starting point to the goal. It prioritizies the paths that appear to be most promising, regardless of whether or not they are actually the shortest path. The algorithm works by evaluating cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal node is reached.

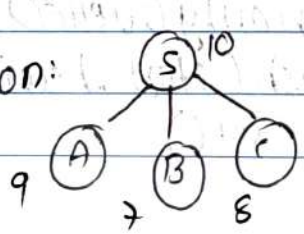
The algorithm works by heuristic function to determine which path is most promising. The heuristic function takes into count the cost of current path and the estimated cost of remaining paths. If the cost of the current path is lower than the estimated remaining paths, then the current path is chosen. This process is repeated until the goal node is reached.

Example: Given:



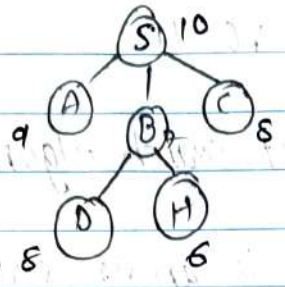
open		closed	
Node	H(n)	Node	Parent
S	10		

1st Iteration:



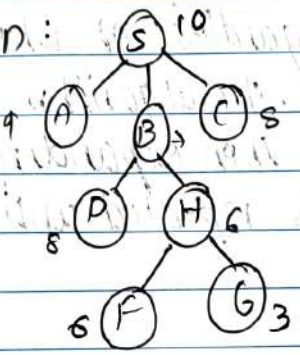
open		closed	
Node	H(n)	Node	Parent
B	7	S	S
C	8	B	S
A	9		

2nd iteration:



open		closed	
Node	H(n)	Node	Parent
H	6	S	
C	8	B	S
D	8	H	B
A	9		

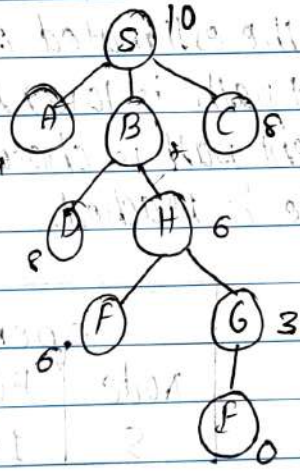
3rd iteration:



open		closed	
Node	H(n)	Node	Parent
G	3	S	
F	6	B	S
C	8	H	B
D	8	G	H
A	9		

4th iteration:

Goal Reached



open		closed	
Node	H(n)	Node	Parent
E	0	S	
F	6	B	S
C	8	H	B
D	8	G	H
A	9	F	G

Path: S → B → H → G → E

ALGORITHM:

```
Greedy-Best-First-Search(start, goal):
    front = PriorityQueue()
    front.push(start, heuristic(start, goal))
    explored = set()
```

```
while not frontier.isEmpty():
    current = frontier.pop()
    if current == goal:
        return "Goal found"
    explored.add(current)
    for neighbor in current.neighbors():
        if neighbor not in explored and neighor not in front:
            front.push(neighbor, heuristic(neighbour, goal))
        else if neighbor in front:
            if heuristic(neighbor, goal) < front.getPriority:
                front.updatePriority(neighbor, heuristic(neighbor, goal))
return "Goal Not Found"
```

Evaluation Parameters:

i) Time Complexity: • In worst case, the time complexity of Greedy Best-First search is exponential i.e. $O(b^d)$.
where:
- b is the branching factor
- d is the depth of shallowest goal state.

ii) Space Complexity: The space complexity of Greedy Best First Search can also be exponential in worst case i.e. $O(b^d)$
- b is the branching factor
- d is the depth of shallowest goal state.

iii) Optimality: It is not guaranteed to find the optimal solution because it makes decisions based solely on heuristic function without considering total cost from start node.

ii) Completeness: Greedy Best First search is not guaranteed to be complete, meaning it may fail to find a solution even if one exists.
If the heuristic function is admissible (never overestimates the cost to reach the goal), and the search space is finite, then Greedy Best-First search is complete.
However if heuristic function is not admissible or search space is infinite, Greedy Best-First search may fail to find a solution.

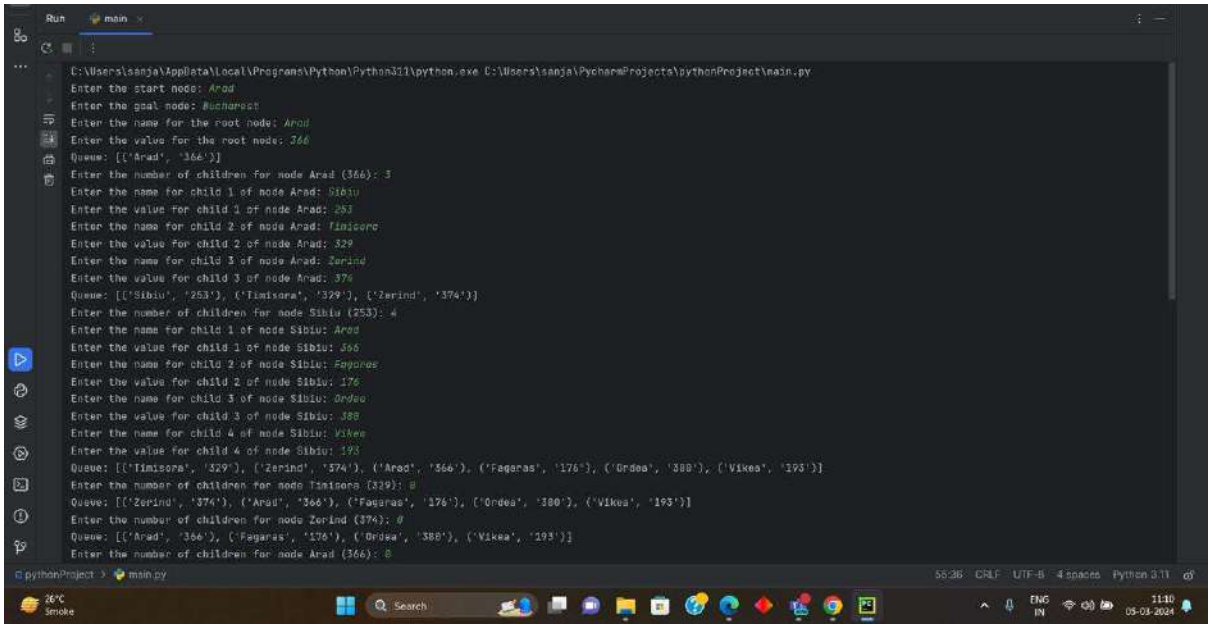
A

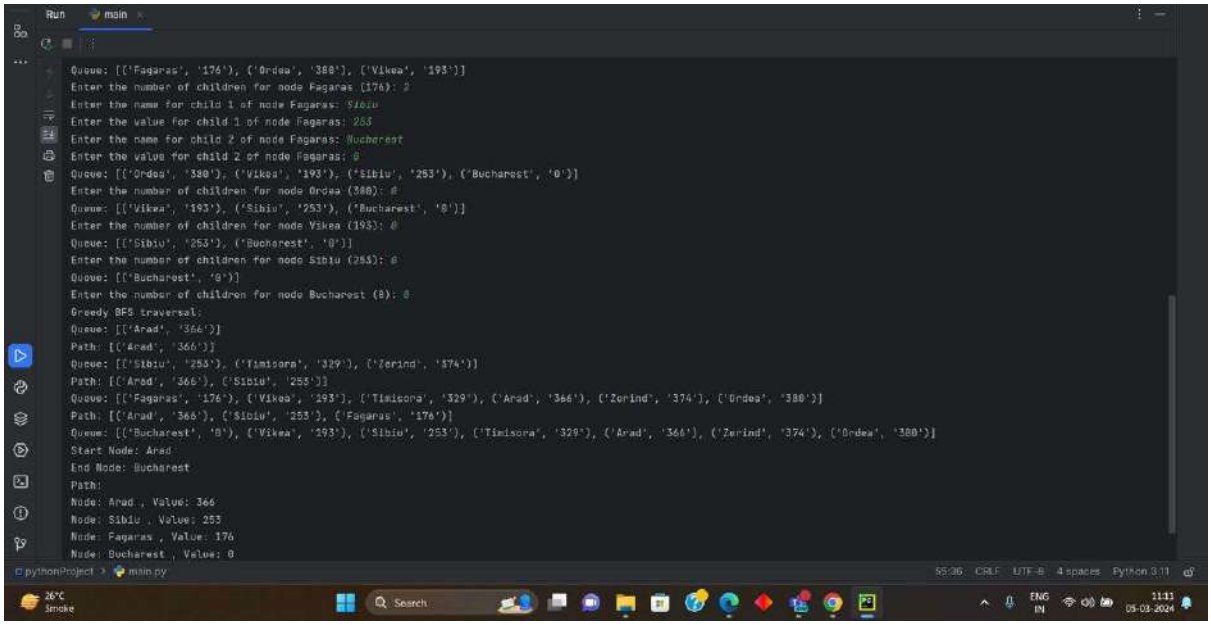
21/3/24

```
class Node:
  def __init__(self, name, value):
    self.name = name
    self.value = value
    self.children = []



def build_tree():
  root_name = input("Enter the name for the root node: ")
  root_value = input("Enter the value for the root node: ")
  root = Node(root_name, root_value)
  queue = [root]


  while queue:
    print("Queue:", [(node.name, node.value) for node in queue])
    current_node = queue.pop(0)
    num_children = int(input(f"Enter the number of children for node {current_node.name}
({current_node.value}): "))
    for i in range(num_children):
      child_name = input(f"Enter the name for child {i + 1} of node {current_node.name}: ")
      child_value = input(f"Enter the value for child {i + 1} of node {current_node.name}: ")
      child_node = Node(child_name, child_value)
      current_node.children.append(child_node)
```

```
            queue.append(child_node)


    return root



def greedy_bfs(root, goal_node):
    queue = [root]
    path = []

    while queue:
        print("Queue:", [(node.name, node.value) for node in queue])
        current_node = queue.pop(0)
        path.append((current_node.name, current_node.value))
        if current_node.name == goal_node:
            break
        if current_node.children:
            queue = sorted(queue + current_node.children, key=lambda x: x.value)
        print("Path:", path)

    return path



def main():
```

```
  start_node = input("Enter the start node: ")
  end_node = input("Enter the goal node: ")
  root = build_tree()
  print("Greedy BFS traversal:")
  path = greedy_bfs(root, end_node)
  print("Start Node:", start_node)
  print("End Node:", end_node)
  print("Path:")
  for node in path:
     print("Node:", node[0], ", Value:", node[1])
     if node[0] == end_node:
        break



if __name__ == "__main__":
  main()
```


Run
main
C:\Users\sanja\AppData\Local\Programs\Python\Python311\python.exe C:\Users\sanja\PycharmProjects\pythonProject\main.py
Enter the start node: Arad
Enter the goal node: Bucharest
Enter the name for the root node: Arad
Enter the value for the root node: 366
Queue: [['Arad', '366']]
Enter the number of children for node Arad (366): 3
Enter the name for child 1 of node Arad: Sibiu
Enter the value for child 1 of node Arad: 253
Enter the name for child 2 of node Arad: Timisora
Enter the value for child 2 of node Arad: 329
Enter the name for child 3 of node Arad: Zerind
Enter the value for child 3 of node Arad: 374
Queue: [['Sibiu', '253'], ['Timisora', '329'], ['Zerind', '374']]
Enter the number of children for node Sibiu (253): 4
Enter the name for child 1 of node Sibiu: Arad
Enter the value for child 1 of node Sibiu: 366
Enter the name for child 2 of node Sibiu: Fagaras
Enter the value for child 2 of node Sibiu: 176
Enter the name for child 3 of node Sibiu: Ordea
Enter the value for child 3 of node Sibiu: 380
Enter the name for child 4 of node Sibiu: Vikea
Enter the value for child 4 of node Sibiu: 193
Queue: [['Timisora', '329'], ['Zerind', '374'], ['Arad', '366'], ['Fagaras', '176'], ['Ordea', '380'], ['Vikea', '193']]
Enter the number of children for node Timisora (329): 0
Queue: [['Zerind', '374'], ['Arad', '366'], ['Fagaras', '176'], ['Ordea', '380'], ['Vikea', '193']]
Enter the number of children for node Zerind (374): 0
Queue: [['Arad', '366'], ['Fagaras', '176'], ['Ordea', '380'], ['Vikea', '193']]
Enter the number of children for node Arad (366): 0
pythonProject > main.py
55:36 CRLF UTF-8 4 spaces Python 3.11
26°C Smoke
Search
ENG IN
11:10 05-03-2024


Run
main
Queue: [['Fagaras', '176'], ['Ordea', '380'], ['Vikea', '193']]
Enter the number of children for node Fagaras (176): 2
Enter the name for child 1 of node Fagaras: Sibiu
Enter the value for child 1 of node Fagaras: 253
Enter the name for child 2 of node Fagaras: Bucharest
Enter the value for child 2 of node Fagaras: 0
Queue: [['Ordea', '380'], ['Vikea', '193'], ['Sibiu', '253'], ['Bucharest', '0']]
Enter the number of children for node Ordea (380): 0
Queue: [['Vikea', '193'], ['Sibiu', '253'], ['Bucharest', '0']]
Enter the number of children for node Vikea (193): 0
Queue: [['Sibiu', '253'], ['Bucharest', '0']]
Enter the number of children for node Sibiu (253): 0
Queue: [['Bucharest', '0']]
Enter the number of children for node Bucharest (0): 0
Greedy BFS traversal:
Queue: [['Arad', '366']]
Path: [['Arad', '366']]
Queue: [['Sibiu', '253'], ['Timisora', '329'], ['Zerind', '374']]
Path: [['Arad', '366'], ['Sibiu', '253']]
Queue: [['Fagaras', '176'], ['Vikea', '193'], ['Timisora', '329'], ['Arad', '366'], ['Zerind', '374'], ['Ordea', '380']]
Path: [['Arad', '366'], ['Sibiu', '253'], ['Fagaras', '176']]
Queue: [['Bucharest', '0'], ['Vikea', '193'], ['Sibiu', '253'], ['Timisora', '329'], ['Arad', '366'], ['Zerind', '374'], ['Ordea', '380']]
Start Node: Arad
End Node: Bucharest
Path:
Node: Arad , Value: 366
Node: Sibiu , Value: 253
Node: Fagaras , Value: 176
Node: Bucharest , Value: 0
pythonProject > main.py
55:36 CRLF UTF-8 4 spaces Python 3.11
26°C Smoke
Search
ENG IN
11:11 05-03-2024

EXPERIMENT NO:5

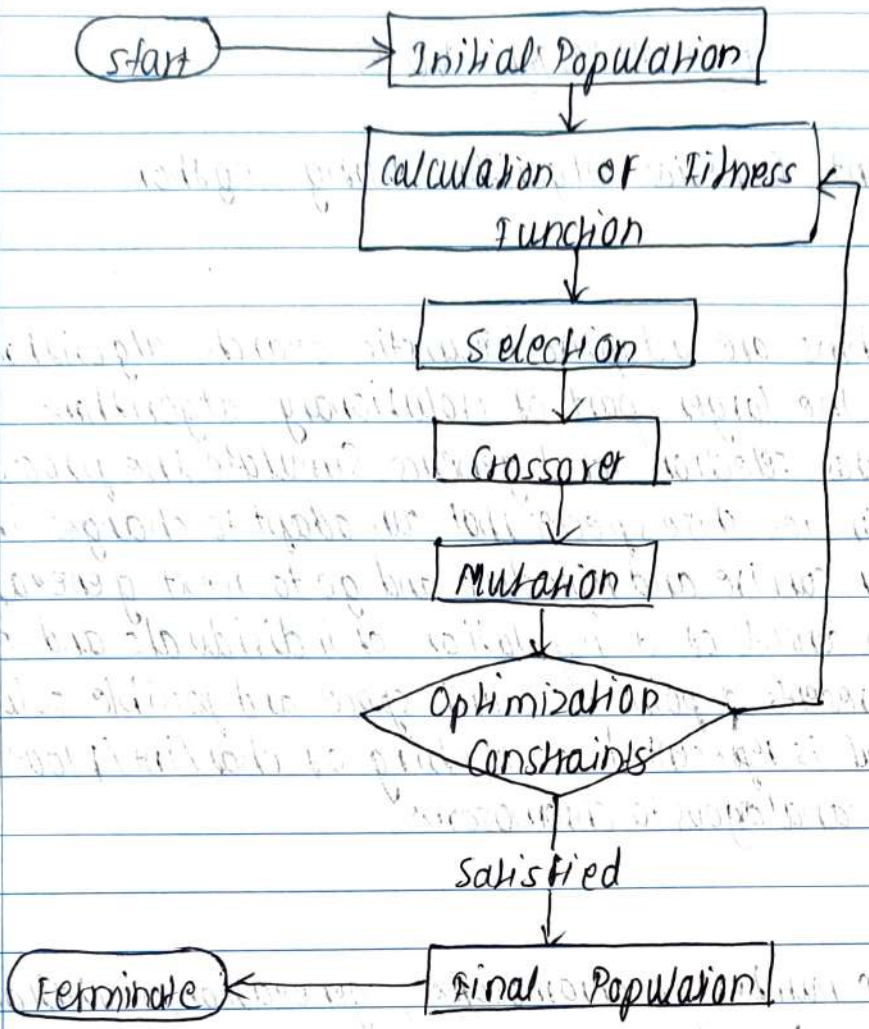
AIM: Implement Genetic Algorithm using Python.

THEORY:

Genetic algorithms are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms Based on idea of natural selection and genetics. Simulate the process of natural selection ie. those species that can adapt to changes in environment can survive and reproduce and go to next generation. Each generation consists of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of char/int/float/bits. This string is analogous to chromosome.

Algorithm:

1. Determine the number of chromosome, generation, mutation rate and crossover rate value.
2. Generate chromosome number of chromosome population and the initialization value of the genes chromosomes. Chromose with random value.
3. Process steps 4-7 repeated until no of generations
4. Evaluation of fitness value of chromosomes by calculating objective function.
5. Chromosome selection
6. Crossover
7. Mutation
8. Best Solution.



Operators: 1. Selection: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generation.

2. Crossover: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly.

A+

3. Mutation: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence.

21/3/24

```
import random

POPULATION_SIZE = 6
GENES = [str(i) for i in range(10)]
TARGET = 30
MAX_ITERATIONS = 50

class Chromosome:
  def __init__(self, genes):
    self.genes = genes
    self.fitness = self.calculate_fitness()

  @classmethod
  def create_chromosome(cls):
    return [random.choice(GENES) for _ in range(4)]

  def calculate_fitness(self):
    return abs(sum((i + 1) * int(gene) for i, gene in enumerate(self.genes)) -
TARGET)

def selection(population):
  sorted_population = sorted(population, key=lambda x: x.fitness)
  return sorted_population[:2]

def crossover(parent1, parent2):
  # Two-point crossover
  crossover_points = sorted([random.randint(1, len(parent1.genes) - 1) for _ in
range(2)])
  child_genes = (
    parent1.genes[:crossover_points[0]] +
    parent2.genes[crossover_points[0]:crossover_points[1]] +
    parent1.genes[crossover_points[1]:]
  )
  return Chromosome(child_genes)

def mutation(child):
  # Randomly mutate one or two genes
  mutated_gene_indices = random.sample(range(len(child.genes)),
random.randint(1, 2))
```

```
    for index in mutated_gene_indices:
        child.genes[index] = random.choice(GENES)
    return child

def main():
    # Given input
    initial_chromosomes = [
        [12, 5, 23, 8],
        [2, 21, 18, 3],
        [10, 4, 13, 14],
        [20, 1, 10, 6],
        [1, 4, 13, 19],
        [20, 5, 17, 1]
    ]

    population = [Chromosome(chromosome) for chromosome in
initial_chromosomes]
    iteration = 0
    best_solution = None

    while iteration < MAX_ITERATIONS:
        parent1, parent2 = selection(population)
        child = crossover(parent1, parent2)
        child = mutation(child)

        population.remove(max(population, key=lambda x: x.fitness))
        population.append(child)

        best_chromosome = min(population, key=lambda x: x.fitness)
        if best_solution is None or best_chromosome.fitness <
best_solution.fitness:
            best_solution = best_chromosome

        iteration += 1

    print("Initial Chromosomes:")
    for i, chromosome in enumerate(initial_chromosomes, 1):
        print(f"Chromosome {i}: {chromosome}")
```

```
    print("\nFinal Chromosomes:")
    for i, chromosome in enumerate(population, 1):
        print(f"Chromosome {i}: {chromosome.genes}")

    print(f"\nBest Chromosome: {best_solution.genes}")
    print(f"Best Solution: {sum((i + 1) * int(gene) for i, gene in
enumerate(best_solution.genes))}")
    print(f"Values of ['a', 'b', 'c', 'd']: {tuple(best_solution.genes)}")
    print(f"Number of Iterations: {iteration}")

if __name__ == "__main__":
    main()
```

```
Initial Chromosomes:
Chromosome 1: ['3', '1', '7', '1']
Chromosome 2: ['2', '1', '7', '1']
Chromosome 3: ['2', '1', '7', '1']
Chromosome 4: ['3', '1', '6', '1']
Chromosome 5: ['3', '9', '6', '1']

Final Chromosomes:
Chromosome 1: ['3', '1', '7', '1']
Chromosome 2: ['2', '1', '7', '1']
Chromosome 3: ['2', '1', '7', '1']
Chromosome 4: ['3', '1', '6', '1']
Chromosome 5: ['3', '9', '6', '1']

Best Chromosome: ['3', '1', '7', '1']
Best Solution: 30
Values of ['a', 'b', 'c', 'd']: (3, 1, 7, 1)
Number of Iterations: 3

Process finished with exit code 0
```

EXPERIMENT NO: 6

AIM: Knowledge representation and creating knowledge base for wumpus world.

Introduction: The wumpus world agent is an example of knowledge based agent that represent knowledge representation, reasoning and planning. knowledge based agent like general knowledge with current percepts to infer hidden characters of current state before selecting action. its necessary is vital in partially observable environment.

Q1. Represent the following sentence in First Order logic using consistent vocabulary.

a). Some student took French passes it in Spring 2001.
b). Every student who takes French passes it.
c). Only one student took greek in spring 2001.
d). The best score in Greek is always higher than best score in french.
e). Every person who buys policy is smart.
f) No person buys an expensive policy.
g) There is an agent who sells policies only to people who are not insured.
h) There is a barber who shaves all men who do not shave themselves.
i) A person born in UK each of whose parent is a UK citizen or UK resident, is a UK citizen by birth.
j) A person born outside UK, one of whose parent is a UK citizen by birth, is a UK citizen by descent.

Solution: ⇒ To represent the sentence in first Order Logic (FOL), we define a consistent vocabulary.

Let:-

1) took(student, course, semester) - represent that student took course in a given semester.
2) passes(student, course) - represent that student pass the course.
3) bestScore(course, score) - represent that best score in specific course.
4) Score(s, c, t) - The score of student s in course c in term t.
5) Expensive(y) - y is expensive.
   Agent(x) - x is an agent.
   SellsPolicy(x, y): x sells policy to y.
   Insured(x): x is insured.
   Barber(x): x is a barber.
   Shaves(x, y): x shaves y.
   Born(x, y): x was born in country y.
   Parent(x, y): x is y's parent.
   Resident(x, y): x is a resident of country y.
   citizenbybirth(x, y): x is citizen of county y by birth.
   citizenbydescent(x, y): x is citizen of country y by descent.
   French: constant, course teaching language French.
   Spring2021: Constant, the spring term in 2001 year.
   Person(x): x is a person.
   Policy(y): y is a policy.
   Buys(x, y): x buys y.
   Smart(x): x is smart.

a. ∃ s took(s, French, Spring 2001)
b. ∀ s,t took(s, French, t) → passes(s, French)

c. $\exists, s$ took $(s,$ greek, spring 2001$) \wedge (\forall z$ took $(s,$ Greek, spring 2001$) \rightarrow s = z)$

d) $\forall, t, s1, s2\ [(\forall s\ 8 > (score(s1, Greek, t), score(8s8, Greek, t))) \wedge (\forall s9 > (score(s2, Greek, t), score(s9, greek, t))] \rightarrow > (score(s1, greek, t), score(s_2, French, t))$.

e. $(\forall x)(\forall y)$ Person$(x) \wedge$ Policy$(y) \wedge$ Buys$(x, y) \Rightarrow$ smart(x).

f. $(\forall x)(\forall y)$ Person$(x) \wedge$ Policy$(y) \wedge$ Buys$(x, y) \Rightarrow \sim$ Expensive(y)

g. $(\exists x)$ (Agent$(x) \wedge (\forall y)$ sells Policy$(x, y) \Rightarrow \sim$ Insured(y))

h. $(\exists x)$ (Barber$(x) \wedge (\forall y)$ man$(y) \wedge$ shaves$(x, y) \Rightarrow \sim$ shave(y, y)).

i. $(\forall x)$ ((Born$(x,$ UK$) \wedge (\forall y)$ (Parent$(y, x) \wedge$ (citizen$(y,$ UK) $\vee$ Resident$(y,$ UK)))) $\Rightarrow$ citizen by Birth$(x,$ UK).

Q2) Introduction of wumpus world is explained earlier.

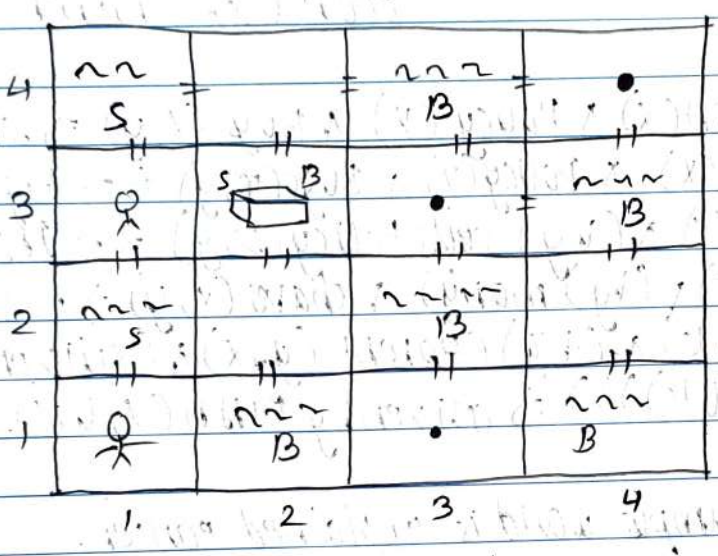
PEAS for wumpus-World Problem are:

1. Performance Measure: - Agent gets gold and return back safe = +1000 pts.
   - Agent dies = -1000 pts.
   - Each move of agent = -1 pt.
   - Agent use the arrow = -10 pts.

2. Environment: - A cave with 16 rooms.
   - Room adjacent to wumpus are stinking
   - Room adjacent to pit are busy.
   - Room with golden glitter.
   - Agent initial position - Room[1, 1] at RHS.
   - Location of wumpus, gold and pit can be anywhere except Room[1, 1].

3. Actuators: - move forward
- Turn right
- Turn left
- Release

4. Sensors: - Breeze
- Stench
- Glitter
- Scream
- Bump



= → Path
• → Pit
→ Treasure
→ agent
B → Breeze
S → Stench
→ Wumpus

The agent start visiting from first square [1,1], we already, it is safe for agent. To build knowledge base for it, we will use some rule and atomic preposition, we need symbol i,j for each location in wumpus world where i is row, j is colomn.

Same preposition rules:

$R_1 \rightarrow S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

$R_2 \rightarrow S_{21} \rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

$R_3 \rightarrow S_{12} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}$

$R_4 \quad S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

Knowledge Base Representation

$\neg W_{11}$	$\neg S_{11}$	$\neg P_{11}$	$\neg B_{11}$	$\neg G_{11}$	V_{11}	OK_{11}
$\neg W_{12}$	---	$\neg P_{12}$	---	---	$\neg V_{21}$	OK_{12}
$\neg W_{21}$	$\neg S_{21}$	$\neg P_{21}$	B_{21}	$\neg G_{21}$	V_{21}	OK_{21}

Now prove that wumpus in the room (1, 3)

- Apply Modus ponens with $\neg S_{11}$ and R_1

$\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$; $\neg S_{11}$

$\neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

- Apply Elimination Rule:
  After applying this we get 3rd statement
  $\neg W_{11}$, $\neg W_{12}$ and $\neg W_{21}$

- Apply modus ponens to $\neg S_{21}$ and R_2

$\neg S_{21} \rightarrow \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$; $\neg S_{21}$

$\neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

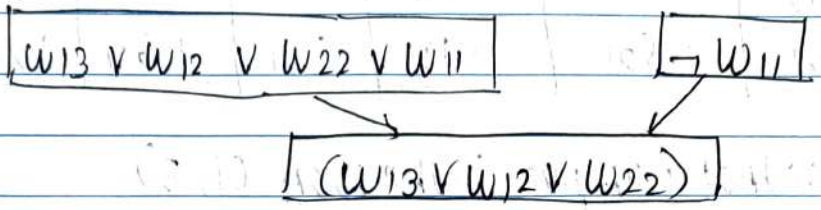
- Apply And 'Elimination Rule':-
  After that we get $\neg W_{21}$, $\neg W_{22}$ and $\neg W_{31}$

- Apply mp to S_{12} and R_4:

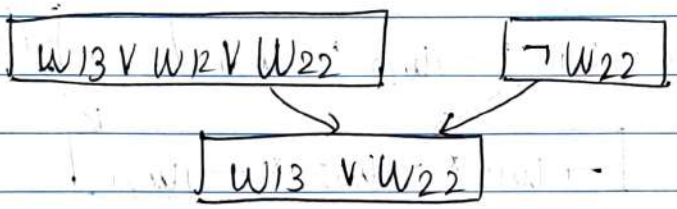
$S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$; S_{12}

$W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

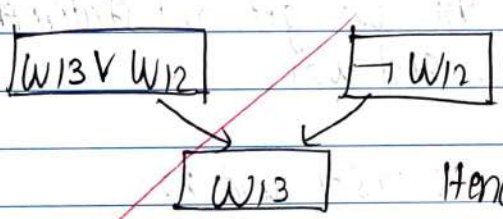
- Apply unit resolution $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ and $\neg W_{11}$:-



- Apply unit resolution on $W_{13} \vee W_{12} \vee W_{22}$ and $\neg W_{22}$



- Apply unit resolution on $W_{13} \vee W_{12}$ and $\neg W_{12}$



Hence proved

EXPERIMENT NO: 7

AIM: Planning for Blocks World Problem.

Q]- Explain the Block world problem and describe how it can be solved using classical planning method.

Planning problem in AI is about the descision making performed by intelligent creature like robot, human or program when trying to achieve goal.

It involves choosing a sequence of action that will transform this state of world and step by step so that it satisfies the goal:-

∴ Planning → set of actions
∴ Actions → Precondition → For actions to be performed there is some need of some pre condition.
∴ Effect → Effect is result of action performed.

Planning consists of:-
I) Choose best rule.
II) Apply result on problem to get new state.
III) Detect if solution is found.
IV) Detect dead end so that new directions will get explored.

what is Block World Problem?:

Classical problem in AI and planning which involves moving back and block from an initial state to goal state.

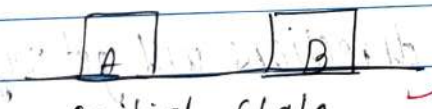
It consists of:
1. Block: square block of same size can be stack on one another.
2. Flat surface /table
3. Robotic Arm.
4. Stack

Initial state:
Consist of set of block placed on the table or each other forming a stack structur. Each block have label and thus position is determined from where they are placed.

Goal state:
It specifies desired configuration of blocks. It would be particular arrangement of block on table or on top of each other. The goal state define what kind configuration of

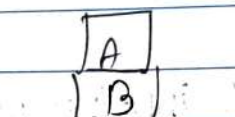
Step Cost:
Each action has step cost associated with it. It represents the cost occured to perform action and to find optimal solution.

Eg:



Initial state

Initial description
on (A, table)
on (B, table)



Goal state

goal description
on (A, B)

Rule	Precondition	Action
pickup(x) ↳ pick up block x	on(table, x) clear(x) hand empty	holding(x)
Putdown(y) ↳ put block y on table	Holding(x)	on(x, table) clear(x) hand empty.
stack(x, y) ↳ put block x on block y	Holding(x) clear(y)	on(x, y) clear(x)
Unstack(x, y) ↳ remove block x from block y	on(x, y) clear(x)	Holding(x) clear(y)

Solution for eg:

Rules:

I) Compare initial state with goal state if they match then stop, else go to step 2.

II) Pop goal state from stack and apply suitable rule to perform action.

III) If all precondition of rule satisfied then perform action.

IV) If any precondition of certain rule satisfied then just pop that from stack, don't perform any action.

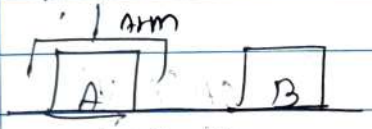
Now by using the above solution S1 ⇒

Action: stack(A,B)
Precondition holding(A)
clear(B)

→

Action: pickup(A)
Precondition: on(A,table)
clear(A)
hand empty.

Initial state

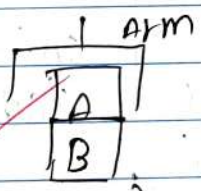


Initially,
on(A, table)
on(B, table)

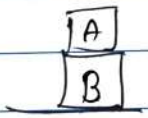
→

Initial state
on table (A, table)
on (B, table)

pickup(A)



on (A, B)



Goal description
on(A, B)

Goal State.

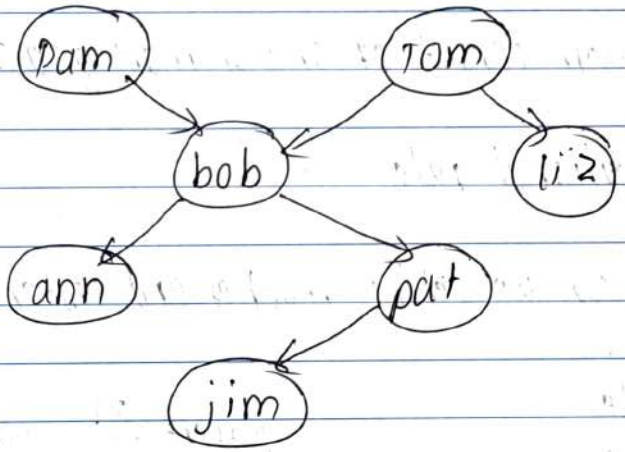
EXPERIMENT NO:8

AIM: Implementation of Family Tree using Prolog.

THEORY:

Prolog is a programming language for symbolic, non numeric computation. It is specially well suited for both solving problem that involve object and also the relation between those objects.

In prolog, we can write the family tree as:




```
parent(pam,bob).          sibling(bob,liz).
parent(tom,bob).          sibling(ann,pat).
parent(tom,liz).
parent(bob,ann).          grandparent(Pam,ann).
parent(bob,pat).          grandparent(Pam,pat).
parent(pat,jim).          grandparent(Bob,Jim).
```

These rules help illustrate important points.

we have defined parent relation by starting 'n' tuple of object based on given information in family tree.

The user can easily query the prolog system about relation defined in the program.

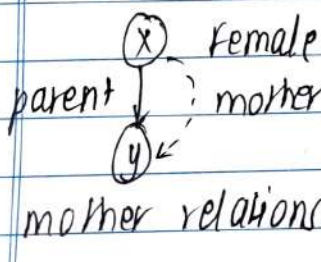
A prolog consists of classes terminated by fullstop.

The argument of relation can be concrete object, constant or general object such as X and Y object of first kind in our program are called atoms. Objects of second kind are called variables.

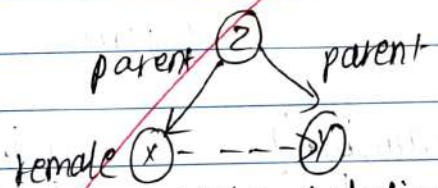
Question to system consists of one or more goals.

There are additional rules:-

Let's see how can we define mother and sister relationship



mother relationship



sister relationship

mother(x,y) :- parent(x,y), female(x).
sister(x,y) :- parent(z,x), parent(z,y), female(x), x \==y.

Similarly we can define,

father(x,y) :- parent(x,y), male(x).

hasChild (x) : parent (x, _)
brother (x, y): parent (z, x) , parent (z, y) , male (x) , x \= y

For eg:

* Facts *

male (John)
female (Lisa)
female (Sarah)
parent (John, Lisa)
parent (John, Sarah).

* Query *

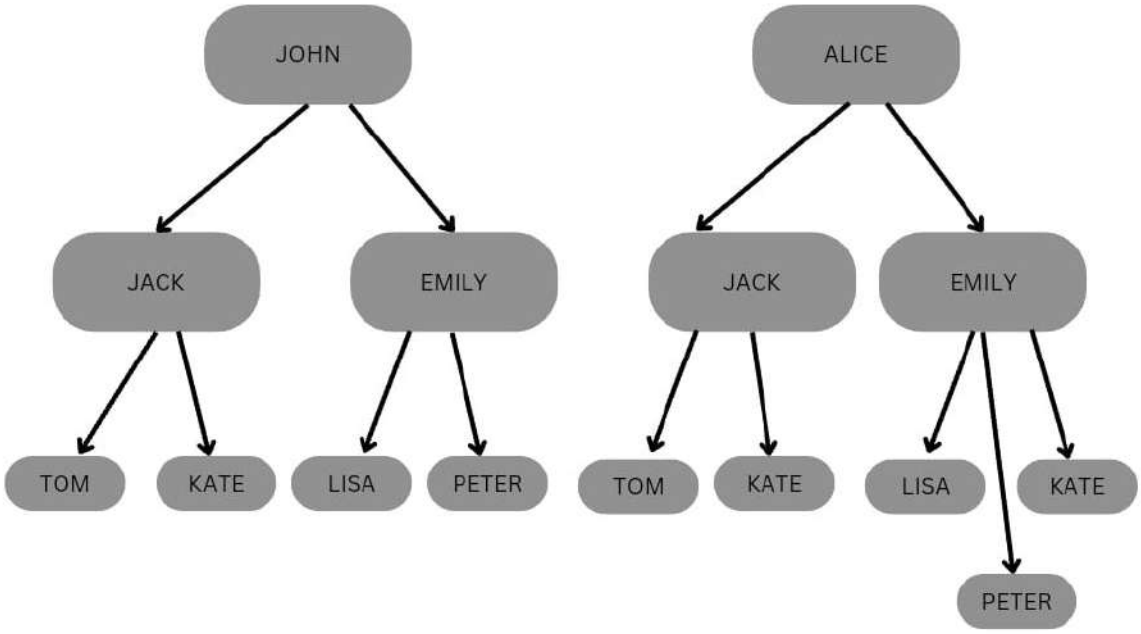
? - father (John, Lisa)
true

? - mother (Jane, Sarah)
false. % Jane is not defined in facts. Prolog cannot find her as mother of Sarah.

? - parent (John, Sarah)
true % John is parent of Sarah.

```
/* Facts */

parent(john, jack).
parent(john, emily).
parent(alice, jack).
parent(alice, emily).
parent(jack, tom).
parent(jack, kate).
parent(emily, lisa).
parent(emily, peter).

/* Rules */

father(X, Y) - parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
grandparent(X, Z) - parent(X, Y), parent(Y, Z).
sister(X, Y) :- parent(Z, X), parent(Z, Y),female (X), X \= Y.
brother(X, Y) :- parent(Z, X), parent(Z, Y), male(X), X \= Y.

/* Gender facts */

male(john).
male(jack).
male(tom).
male(peter).

female(alice).
female(emily).
female(kate).
female(lisa).
```

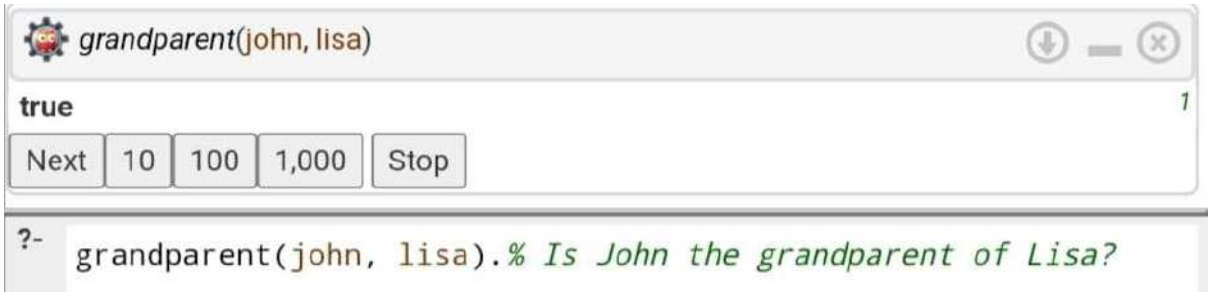
FAMILY TREE:

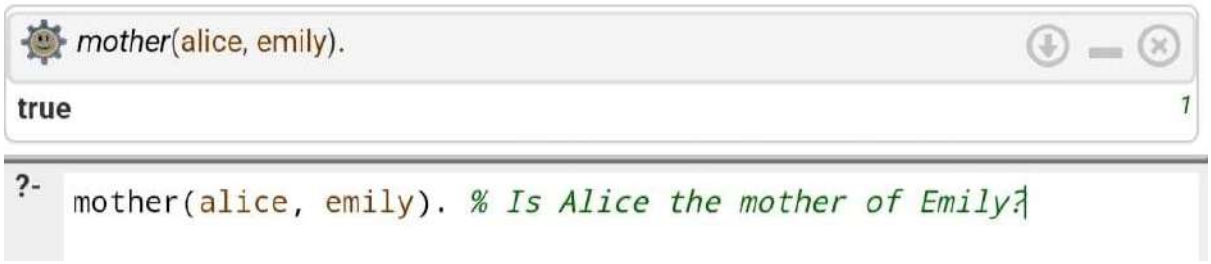

JOHN
JACK
EMILY
TOM
KATE
LISA
PETER
ALICE
JACK
EMILY
TOM
KATE
LISA
KATE
PETER

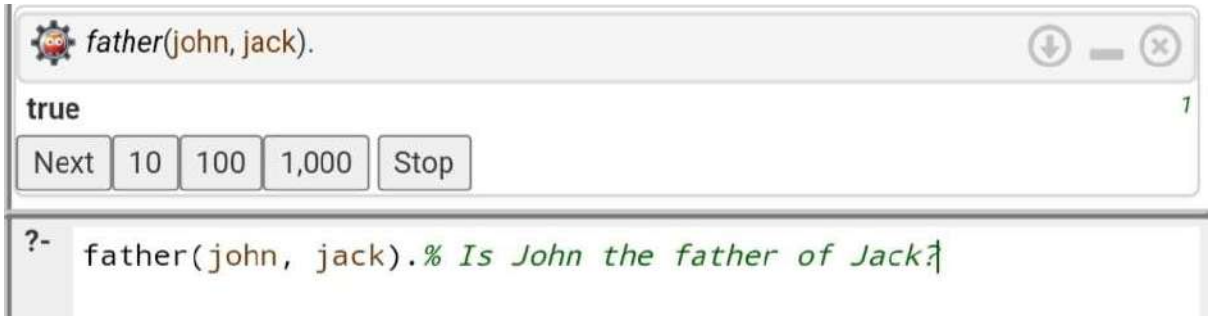
SWISH File Edit Examples Help 206 users online

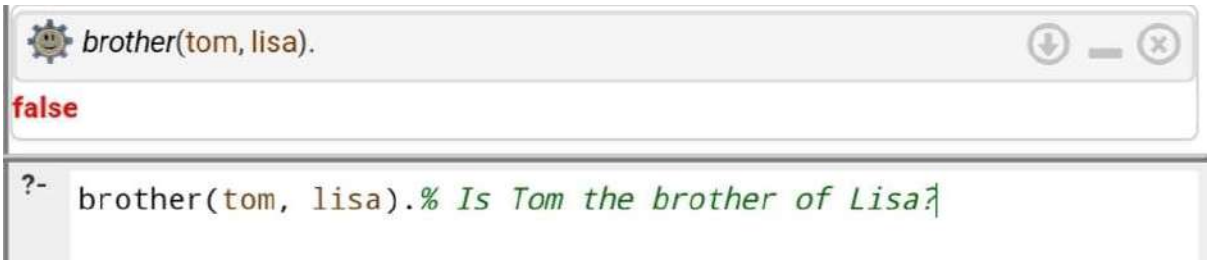
Program

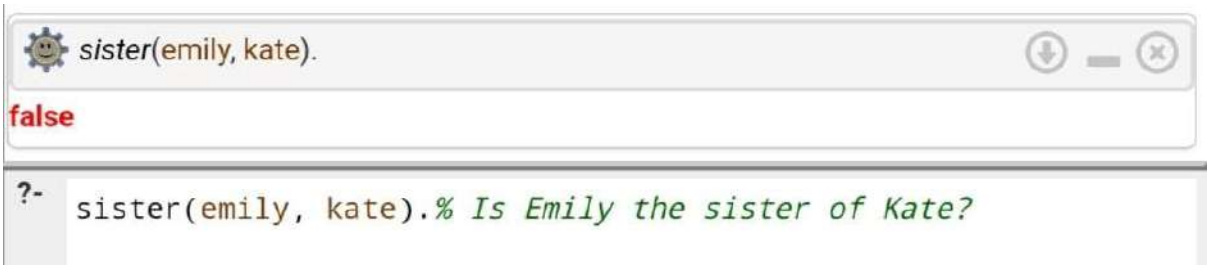
```
/* Parth D. C12 2103032
 /* Facts */
parent(john, jack).
parent(john, emily).
parent(alice, jack).
parent(alice, emily).
parent(jack, tom).
parent(jack, kate).
parent(emily, lisa).
parent(emily, peter).

/* Rules */
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
sister(X, Y) :- parent(Z, X), parent(Z, Y), female(X), X \= Y.
brother(X, Y) :- parent(Z, X), parent(Z, Y), male(X), X \= Y.

/* Gender facts */
male(john).
male(jack).
male(tom).
male(peter).

female(alice).
female(emily).
female(kate).
female(lisa).
```


grandparent(john, lisa)
true
1
Next
10
100
1,000
Stop
?-
grandparent(john, lisa).% Is John the grandparent of Lisa?


mother(alice, emily).
true
1
?-
mother(alice, emily). % Is Alice the mother of Emily?


father(john, jack).
true
1
Next
10
100
1,000
Stop
?-
father(john, jack).% Is John the father of Jack?


brother(tom, lisa).
false
?-
brother(tom, lisa).% Is Tom the brother of Lisa?


sister(emily, kate).
false
?-
sister(emily, kate).% Is Emily the sister of Kate?

Parth D.
C12
2103032

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment no : 1

Assignment on : a) Introduction to AI
b) Problem Foundation
c) PEAS Properties

Q1] Give one definition on AI for each of the following approaches:

i) Acting Humanly: This approach focusses on developing AI systems that mimic human behaviour or actions. A definition for this approach could be "AI acting humanly refers to the ability of artificial intelligence systems to perform tasks or exhibit behaviours in a manner that is indistinguishable from human actions."

ii) Thinking Humanly: This approach aims to understand and replicate human thought processes with AI systems. A definition for this approach could be "AI thinking humanly involves the emotion/ emulation of human thought processes, such as reasoning, problem solving, and decision making, to achieve intelligent behaviour." "The automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning..."

iii) Acting Rationally: This approach concentrates on designing AI systems that make decisions to achieve optimal outcomes, regardless of whether the decision-making process resembles human thinking. A definition for this approach could be "AI acting rationally refers to the ability of artificial systems to select actions or make decisions that lead to the best possible outcomes, based on information and goals".

iv) Thinking Rationally: This approach focusses on creating AI systems that follow the principles of logic and rationality in their decision making and problem-solving processes. A definition for this approach could be "AI thinking rationally involves the application of logical reasoning and rational principles to solve problems and make decisions, without necessarily imitating human thought patterns". "The study of mental faculties through the use of computational model." "The study of computations that make it possible to percive, reason and act."

Q2]- Explain the components of AI systems in detail:

Artificial intelligence is a field, which combines computer science and robust datasets, to enable problem solving. It also encompasses sub-fields of machine learning and deep learning, which are frequently mentioned in conjunction with artificial intelligence.

To understand AI systems thoroughly, here is an explanation of the components of AI system:

1. Perception: Perception is the ability of AI system to gather data from its environment through various sensors or input devices. This data can include text, image, sound, and other forms of sensory information. Perception allows AI system to understand and interpret the world around it, which is essential for tasks such as image recognition, speech recognition and natural language understanding.

2. Knowledge Representation: knowledge representation involves the process of storing and organizing information in a format that AI

systems can understand and manipulate. This can include representing knowledge usings symbols, graphs or other structured format.

3. Learning: Learning is the process by which AI system improves its performance overtime through experience. There are different types of learning approaches in AI, including supervised learning, unsupervised learning, and reinforcement learning. In supervise learning, AI is trained on a labelled data, while in unsupervised learning, it discovers patterns and relationships in data. Reinforcement learning involves learning by interacting with environment and recieving feedback in form of rewards or penalties.

4. Reasoning: Reasoning is the process of drawing conclusions or making inferences based on available information and knowledge. AI uses various reasoning techniques such as deductive reasoning, inductive reasoning, probabilistic reasoning to make logical deductions and predictions. Reasoning enables AI systems to make informed decisions, solve problems and perform tasks that require logical thinking.

5. Problem Solving: Problem solving involves finding solutions to complex problems or achieving goal in a given domain. AI systems use heuristics to search through problems and its possible solutions and evaluate the effectiveness based on pre-defined criteria. It includes search algorithms, constraint satisfaction, optimization algorithms and planning.

6. Natural Language Processing (NLP): NLP is the field of AI that focusses on enabling the computers to understand, interpret and generate human language. NLP encompasses tasks such as speech

recognition, language translation, sentiment analysis and text generation. NLP techniques involve processing and analyzing natural language data using machine learning algorithms, stastical models and linguistic rules.

03) Write a short note on categorization of AI.

The categorization of AI can be approached from two perspectives: based on Capabilities and based on Functionalities:

1. Based on Capabilities:

a. Weak AI: Also known as a Narrow AI, refers to AI systems designed and trained for specific tasks or domains. These systems demonstrate intelligence with only within narrow scope of program capabilities. Eg include virtual assistants, recommendation systems and image recognition algorithms.

b. General AI: Also refered to as AGI (Artificial General Intelligence), describes AI systems with human-like cognitive abilities capable of understanding, learning and reasoning across a wide range of tasks and domains. General AI possesses a level of versatality and adaptability that surpasses narrow AI systems.

c. Strong AI: Refered to as ASI (Artificial superintelligence) that exceeds human intelligence in every aspect, including, creativity, problem solving and emotional understanding. strong AI posseses the ability to autonomously learn and improve its indefinetly, leading to profound transformation in society.

2. Based on Functionalities:

a. Reactive AI: This AI operates purely in present moment, responding to inputs based on predefined rules and patterns. They lack memory and cannot form long-term strategies or learns from past experiences. Eg: Game playing AI agents like Deep Blue, which defeated Chess World Champion Garry Kasparov in 1997.

b. Limited Memory AI: It possesses the ability to learn from historical data and past experiences to improve future decision making. These systems retain a limited amount of information over-time, enabling them to adapt to changing environments and make more informed choices.

c. Theory of Mind Machines: Theory of Mind Machines are AI systems capable of understanding and attributing mental states, such as beliefs, intentions and emotions to themselves and to others. These systems can anticipate the behaviour of other agents and adjust their actions accordingly, leading to more sophisticated forms of interactions and collaboration.

d. Self-Aware AI: Represents a hypothetical scenario where AI-systems possess consciousness and subjective experience akin to humans. These systems would not only exhibit intelligence but also self-awareness, introspection and possible emotions.

Q4] Explain the problem formulation with the help of an example

Problem formulation is a crucial step in designing an artificial intelligence system, particularly in the context of problem solving agents. It involves defining the elements of a problem in a structured manner so that AI can effectively search for a solution.

1. State formulation: The problem space is represented in terms of states. A state describes a particular configuration or a system situation that the problem solving agent can be in. It encompasses all relevant aspects of the problem at a given time point. For example in a navigation problem, states could be represented differently on the map.

2. Initial state: Initial state is the starting point of the problem solving process. It represents the situation in which the problem solving agent begins to search for a solution. The initial state provides the context from which the agent will navigate towards the desired goal. In navigation problem, the initial state would be the current location of the user.

3. Goal State (Test): The goal state determines whether a given state is a goal state or not ie. whether the problem has been solved. It defines the condition that must be satisfied for the problem to be considered solve.

4. Action sequence: Action represents the possible transitions between states in the problem space. An action defines the effect of performing a particular operation or moving from one state to another. In the navigation problem, problem solving could be moving from one location of the map to the other.

5. Path Cost : The path cost assigns a numerical cost to each action or a sequence of actions in the problem space. It quantifies the resources or effort required to execute a particular action or reach a specific state. In navigation problem, the path cost could be measured in terms of distance or time associated with travelling between locations on the map.

Example: Consider a robot which needs to navigate from its initial location to its final location on the map.

- State Formulation: Each state represents a distinct location on the map.
- Initial State: The initial state is the current location on the map.
- Goal Test: The goal test checks whether robot current location matches the specified goal location
- Action sequence: Actions including moving the robot up, down, left or right on the map.
- Path Cost: Path cost is defined as the number of movements required to reach the goal location.

Q5] Explain PEAS properties in detail.

PEAS properties provide a structed framework for defining the characteristics of an intelligent agent. These properties are performance measures, Environment, Actuators and Sensors.

1. Performance Measures: The performance measure defines the criteria by which the success or effectiveness of the agent's behaviour is evaluated. It specifies what the agent is trying to achieve and how well it accomplishes its objectives.

2. Environment: The environment refers to the external context in which the agent operates and interacts. It encompasses all the entities, objects and conditions that the agent percieves and affec through its actions.

3. Actuators: Actuators are the mechanisms or the compononents through which the agent affects its environment. They enable the agent to perform actions and exert control over its surroundings. They are designed to perform, including motors, effectors, manipulators and communication devices.

4. Sensors: Sensors are perceptual mechanisms or devices through which the agent recieves information about its environment. They enable the agent to percieve and interpret sensory inputs, such as visuals, auditory, tachle or other forms of sensory data.

Example: Lets consider the PEAS properties for customer support chatbot.

Performance Measures: • Response Accuracy: Percentage of inquiries answered correctly.
• Respone Time: Preage time to respond to queries
• User satisfaction: Sentiment Analysis of user feedbacks and ratings.

Environment: • Online Platform: Web/app where chatbot is deployed.
• User Interaction: Text based queries, complaints, feedbacks.
• Customer data: User Profile, Purchase history, preferences.
• Service knowledge Base: Info about products, FAQ, services.

3. Actuators : • Text Output : Responding to queries, providing info.
• Interactive Buttons/Menus : Offering pre-defined options for users to chose from during interactions
• API calls : Accessing external databases to perform actions.

4. Sensors : • Natural Language Understanding (NLU) : Analyzing and understanding semantics and intent of user messages.
• User FeedBack Ratings
• Knowledge Base Queries

Q6]- Describe different types of environments with suitable examples.

Environment in the context of artificial intelligence and reinforcement learning can be characterized along various dimensions.

1. Fully and Partially Observable: Fully observable environments provide an agent with complete information about the current state while partially observable offer limited environment information.
Eg: Fully Observable → Chess Game.
Partially Observable → Poker Game.

2. Deterministic V/s Stochastic : Deterministic environments have outcomes entirely determined by their current state and actions taken. Stochastic environments involves some level of randomness.
Eg: Deterministic → Physics simulation
stochastic → Game involving dice rolls.

3. Dynamic V/s Static : Dynamic environments change over time, while static environments remain constant.

Dynamic → Traffic System
Static → Fixed Puzzle

4. Episodic v/s Sequential : Episodic environments consists of isolated episodes without influencing future states of tic-tac-toe. Sequential environment involves sequence of actions leading to cumulative effects, like a chess game.

5. Discrete v/s Continuous : Discrete environments have distinct, seperate states and actions like a chessboard.
Continuous environment have continuous state and action spaces, such as robotic arm's movement.

6. Single Agent v/s Multiagent : Single Agent environments include a solitary decision maker, while multiagent environments have multiple independent decision makers.
Single Agent scenario → Maze solving Robot
Multi Agent scenario → Multiple Robots co-operating or competing.

(A)

21/3/24

ASSIGNMENT NO: 2

Q1] Forward and Backward Chaining.

Forward and Backward Chaining are two environmental strategies in AI used by inference engine to make deductions based on available data and goal respectively.

* Forward Chaining:

Definition: Forward Chaining, also known as forward deducation or forward reasoning, start from known facts, trigger rules whose premises are satisfied and adds their conclusion to known facts until the problem is solved.

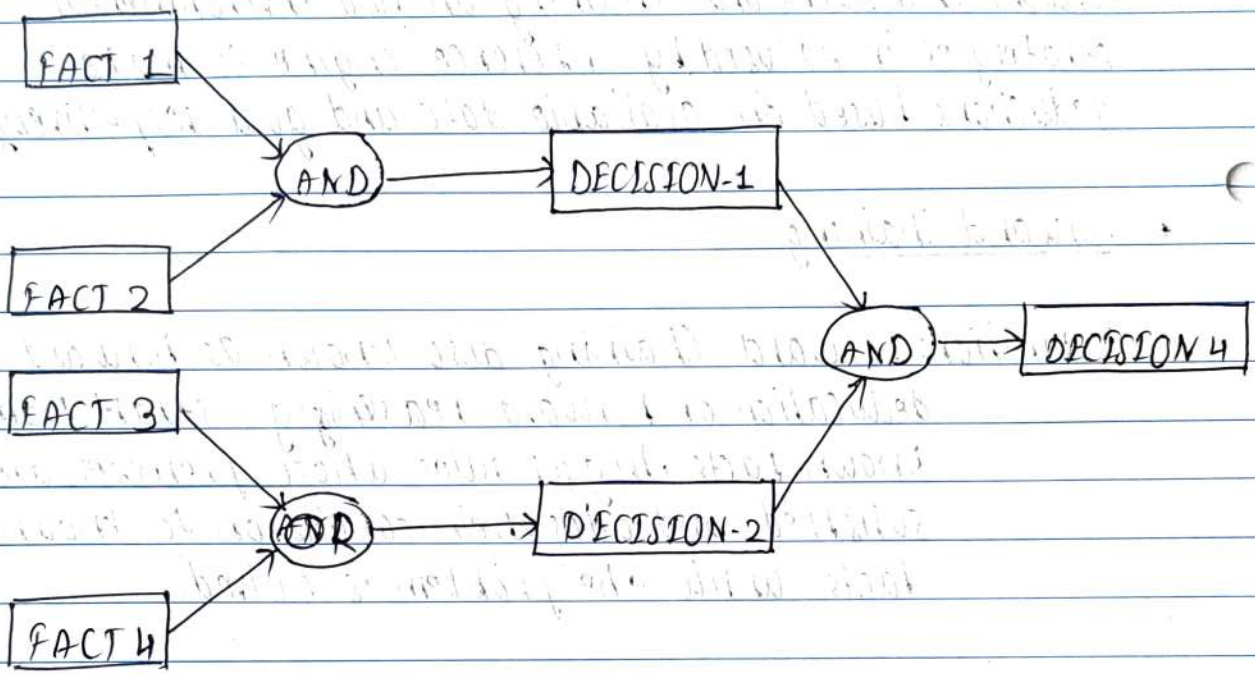
Process: It works from initial state forward toward the goal by applying inference rule to reach a conclusion.

Example: To prove that "Robert is criminal" forward chaining would start with known fact like "American (Robert)" and iteratively apply rule until reaching the conclusion that "Robert is criminal".

What can happen next approach:

Properties:

It is a data structure driven uses Breadth First search (BFS) strategy, aim to get conclusion, operates in forward direction and used in planning, monitoring, control and interpretation application.



* Backward Chaining:

Definition: Backward chaining, also known as Backward deduction or backward reasoning, start from goal and works backward through rules to find known facts that support the goal.

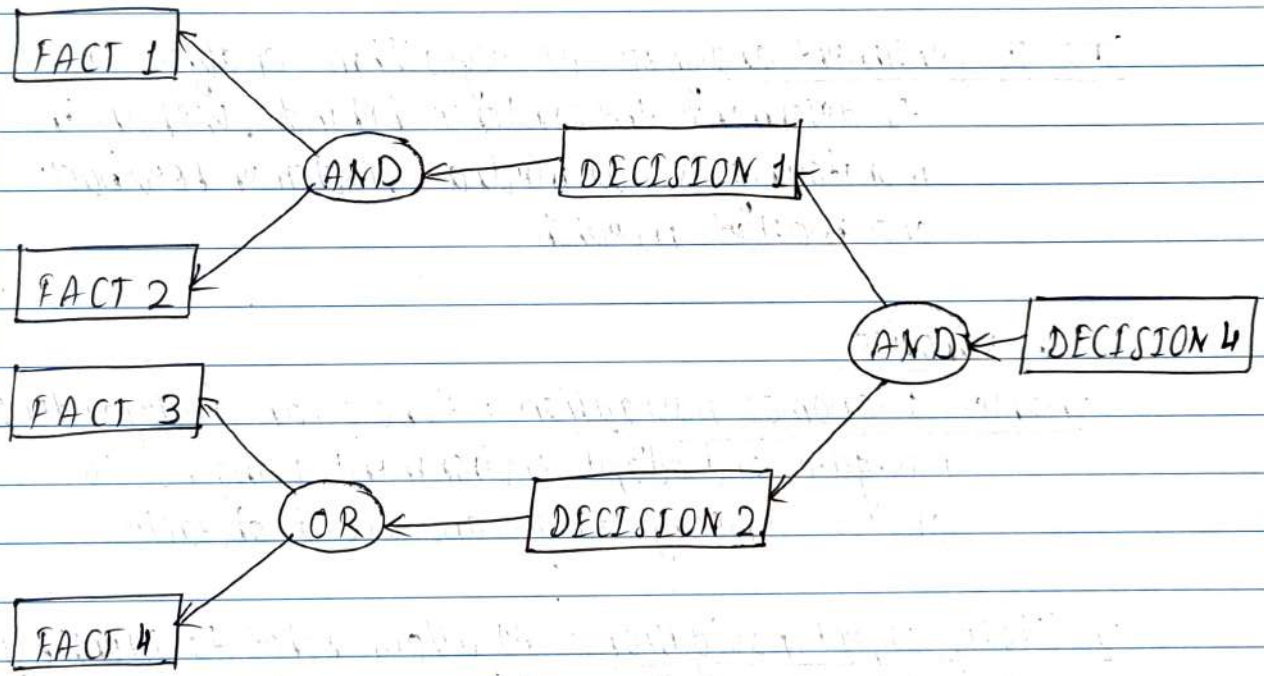
Process: It begin with goal and work toward the initial state by inferring facts that lead to signal.

Example: Starting with goal "Robert is a criminal", backward chaining would infer fact like "if he

is running, he sweats" to establish that "He is running".

Properties: It goal driven uses DFS strategy and aim to find possible fact or required data, operates in backward direction and used in automated inference engines and other AI application.

why this happened approach.



Q2). AI in NLP and Robotics.

AI in NLP:

Enhances language understanding:- AI model comprehend and generate human like text, enabling complex interactions and understanding of context.

Improves information extraction: AI can identify and extract specific information from large volume of text, making data processing more efficient.

Power chatbots and virtual assitant: Leveraging AI, these platform can understand and respond to human queries in natural manner, offering personalized assitant.

Conducts sentiment analysis: AI algorithm analyze text to determine the sentence behind, useful in monitoring social media, customer feedback and market research.

AI in Robotics:

Enables autonomos navigation: Robots can independently navigate and adapt environment, using AI to process sensory data and avoid obstacles.

Facilitates object navigation and manipulation: AI allow robot to identify, classify and interact object in their surrounding, crucial for tasks in manufacturing, logistics and service industries.

Improves human robot interaction: AI enhance ability of robot to understand and respond to human language and gesteres, making interaction more natural and intuitive.