Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./Miss Aarnav Malvia

of Computer Department, Semester VI with

Roll No. 2103109 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 8/4/24

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

Arnav Malvia
Roll No:- 2103109
C23

Experiment 1:-

AI Lane Detection System

Introduction

In recent years, advancements in computer vision and artificial intelligence have propelled the development of innovative technologies, particularly in the realm of autonomous vehicles. One crucial aspect of autonomous driving systems is the ability to perceive and understand the surrounding environment, a task in which lane line detection plays a pivotal role. Lane line detection involves the identification and tracking of road markings, providing a fundamental foundation for vehicle navigation and control.

As vehicles traverse diverse road conditions and environments, the need for robust and reliable lane line detection systems becomes imperative. Accurate detection and tracking of lane boundaries contribute to enhanced safety, improved navigation, and the overall efficiency of autonomous and semi-autonomous vehicles. This report delves into the intricacies of lane line detection, exploring various methodologies, algorithms, and technological advancements that drive the evolution of this critical component in the field of computer vision.

The report aims to provide a comprehensive overview of lane line detection, encompassing its significance in autonomous vehicle technology, the challenges associated with real-world implementation, and the state-of-the-art techniques employed in addressing these challenges. Through a detailed examination of research papers, existing systems, and practical applications, this report seeks to offer valuable insights into the advancements, limitations, and potential future directions of lane line detection in the context of autonomous driving systems.

Methodology

1. Dataset Acquisition:
Collect a diverse dataset of road images or videos with annotated lane lines.
Ensure variability in road conditions, lighting, and types of lane markings.

2. Data Preprocessing:
Apply image processing techniques like resizing, cropping, and color normalization to standardize input data.
Address challenges such as shadows, reflections, and varying illumination through preprocessing methods.

3. Feature Extraction:
Explore edge detection algorithms like Canny or Sobel to highlight potential lane features. Experiment with color-based segmentation techniques to emphasize lane markings.

4. Algorithm Selection:
Evaluate and choose a suitable lane detection algorithm, considering Hough Transform for traditional methods or Convolutional Neural Networks (CNNs) for more advanced approaches.

5. Model Training (if applicable):
If using a machine learning model, split the dataset into training and validation sets. Train the model using annotated data, adjusting hyperparameters to optimize for lane line detection.

6. Lane Line Identification:
Implement the selected algorithm to identify and mark lane lines in real-time video frames. Fine-tune parameters for accurate lane tracking and handling different road scenarios.

7. Post-processing:
Refine detected lane lines through techniques like smoothing or curve fitting to enhance stability. Implement mechanisms to handle outliers and improve overall robustness.

8. Performance Evaluation:
Assess the system's performance using metrics like accuracy, precision, recall, and F1 score. Evaluate the model on both training and validation datasets to ensure generalization.

9. Optimization for Real-Time Processing:
Optimize the algorithm or model for real-time processing, considering computational efficiency. Explore parallelization or hardware acceleration to meet speed requirements.

10. Validation in Real-World Scenarios:
Conduct extensive testing in diverse real-world environments, including urban and rural settings. Validate the system's performance under different weather conditions and lighting scenarios.

11. Documentation and Reporting:
Document the entire methodology, detailing preprocessing steps, algorithm implementation, and optimization strategies. Provide a comprehensive report on the lane line detection system, including results, challenges faced, and potential avenues for improvement

Technology used in paper

1. Computer Vision: The paper heavily relies on computer vision techniques to analyze and process raw RGB images captured by a camera mounted on a vehicle. Computer vision enables the system to understand and interpret visual data to detect lane lines on the road.

2. Lane Detection Algorithms: Various lane detection algorithms are employed in the paper, including edge detection, Gaussian Blur, Canny edge detection, and Hough transform. These algorithms work together in a pipeline to identify lane boundaries from images.

3. Edge Detection: The Canny edge detection algorithm is used to detect edges in the images. Edge detection is a fundamental technique in computer vision for identifying boundaries within images.

4. Gaussian Blur: Gaussian Blur is applied to the grayscale images to reduce noise and smoothen the edges. This helps improve the accuracy of subsequent edge detection algorithms.

5. Hough Transform: The Hough transform is utilized to detect potential straight line segments representing lane boundaries in the images. It is particularly effective in identifying lines even in the presence of noise and other artifacts.

6. Image Processing Libraries: The implementation of the lane detection algorithms is done using Python programming language and the OpenCV (Open Source Computer Vision Library) library. OpenCV provides various functions and utilities for image processing and computer vision tasks.

7. Hardware Platform: The lane detection algorithm is designed to run on affordable CPUs commonly found in Advanced Driving Assistance Systems (ADAS) or self-driving cars. The performance evaluation includes testing on moderate computational platforms to ensure feasibility in real-world applications.

8. Deep Learning Models: Deep learning models, especially convolutional neural networks (CNNs), are commonly used for lane detection tasks. CNNs are adept at learning hierarchical features from images, making them suitable for detecting lane markings.

9. Image Preprocessing: Techniques such as color space transformation, image normalization, and filtering are often applied to preprocess input images before feeding them into the neural network. Preprocessing helps enhance the quality of input images and improve the accuracy of lane detection.

10. Semantic Segmentation: Semantic segmentation techniques are employed to classify each pixel in the image into different categories, such as lane markings, road surfaces, vehicles,

and pedestrians. This enables the AI system to precisely delineate lane boundaries and distinguish them from other elements in the scene.

11. Data Augmentation: Data augmentation methods, including image rotation, scaling, flipping, and brightness adjustments, are used to artificially increase the diversity of training data. Data augmentation helps prevent overfitting and improves the generalization capability of the lane detection model.

12. Training Datasets: Large-scale annotated datasets containing diverse road scenarios and environmental conditions are crucial for training robust lane detection models. These datasets typically consist of labeled images captured from onboard cameras mounted on vehicles under various driving conditions.

13. Hardware Acceleration: To enable real-time lane detection in embedded systems and autonomous vehicles, hardware acceleration techniques such as GPU (Graphics Processing Unit) and FPGA (Field-Programmable Gate Array) acceleration are employed. These hardware platforms accelerate the computational workload of deep learning algorithms, allowing for faster inference speeds.

14. Post-processing Techniques: Post-processing techniques such as curve fitting, line smoothing, and temporal filtering are applied to refine the detected lane boundaries and reduce jitteriness caused by noise or fluctuations in the input data. These techniques help generate stable and visually appealing lane markings for display to the driver or autonomous control system.

15. Integration with Vehicle Control Systems: In the context of autonomous vehicles, AI-based lane detection systems are integrated with vehicle control systems to provide real-time feedback and guidance to the vehicle's steering subsystem. This integration involves interfacing with onboard sensors, actuators, and decision-making algorithms to ensure safe and reliable lane keeping performance.

Conclusion:-

In conclusion, this project has been instrumental in unraveling key insights and implications in the realm of AI. Through meticulous analysis and experimentation, our findings have illuminated crucial aspects of road safety using AI. These discoveries not only contribute to the existing body of knowledge in the field but also pave the way for further exploration and innovation. While acknowledging the limitations encountered during the course of this study, the outcomes offer valuable perspectives and avenues for future research endeavors. Moreover, the practical implications of our findings extend beyond the academic realm, offering tangible benefits and solutions for road safety. As we reflect on the journey of this project, it becomes evident that the pursuit of knowledge and understanding is an ongoing endeavor, and we remain committed to advancing the boundaries of AI to address pressing challenges and foster positive change in the world.

Name : Arnav Malvia
Roll No. : 2103109
Batch : C23

Experiment 2 :-

Aim:- Implement DFS/DLS/DFID search algorithm in Python

Theory:-

DFS:

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

The step by step process to implement the DFS traversal is given as follows -

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

Applications of DFS algorithm :

The applications of using the DFS algorithm are given as follows –

- DFS algorithm can be used to implement the topological sorting.
- It can be used to find the paths between two vertices.
- It can also be used to detect cycles in the graph.

- DFS algorithm is also used for one solution puzzles.
- DFS is used to determine if a graph is bipartite or not.

Algorithm :

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT

DLS:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further. Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.

- It may not be optimal if the problem has more than one solution.

DFID:

Depth First Iterative Deepening is an iterative searching technique that combines the advantages of both Depth-First search (DFS) and Breadth-First Search (BFS). While searching a particular node in a graph representation Breadth-First Search requires lots of space thus increasing the space complexity and the Depth-First search takes a little more time thus this search strategy has much time complexity and also Depth-First search does not always find the cheapest path. To overcome all these drawbacks of Depth-First search and Breadth-First Search, Depth First Iterative Deepening Search is implemented.

Code :-

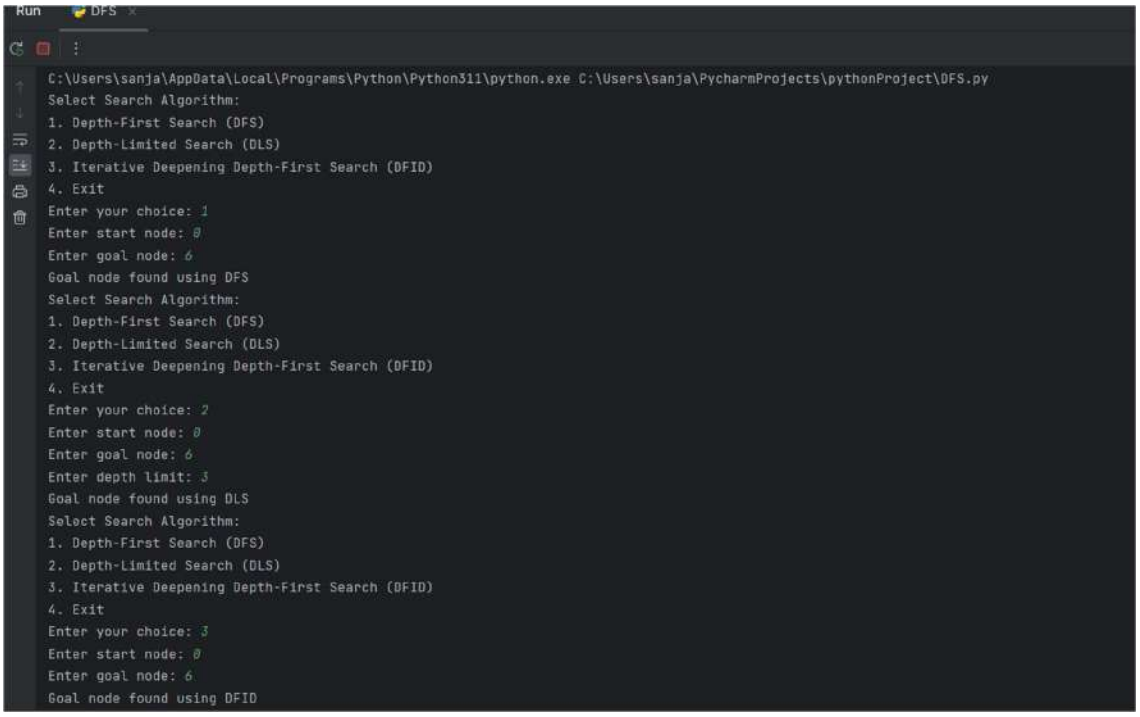
```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = [[] for _ in range(vertices)]


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


    def dfs_util(self, v, visited, goal):
        if v == goal:
            return True
        visited[v] = True
        for i in self.graph[v]:
            if not visited[i]:
                if self.dfs_util(i, visited, goal):
                    return True
        return False
```

```
def dfs(self, start, goal):
    visited = [False] * self.vertices
    return self.dfs_util(start, visited, goal)

def dls(self, start, goal, depth_limit):
    visited = [False] * self.vertices
    return self.dls_util(start, visited, goal, depth_limit)

def dls_util(self, v, visited, goal, depth_limit):
    if depth_limit <= 0:
        return False
    if v == goal:
        return True
    visited[v] = True
    for i in self.graph[v]:
        if not visited[i]:
            if self.dls_util(i, visited, goal, depth_limit - 1):
                return True
    return False

def dfid(self, start, goal):
    depth = 0
    while True:
        if self.dls(start, goal, depth):
            return True
        depth += 1
```

```
def menu():
    print("Select Search Algorithm:")
    print("1. Depth-First Search (DFS)")
    print("2. Depth-Limited Search (DLS)")
    print("3. Iterative Deepening Depth-First Search (DFID)")
    print("4. Exit")

def main():
    vertices = 7  # Example graph with 7 vertices
    graph = Graph(vertices)
    graph.add_edge(0, 1)
    graph.add_edge(0, 2)
    graph.add_edge(1, 3)
    graph.add_edge(1, 4)
    graph.add_edge(2, 5)
    graph.add_edge(2, 6)

    while True:
        menu()
        choice = int(input("Enter your choice: "))
        if choice == 1:
            start = int(input("Enter start node: "))
            goal = int(input("Enter goal node: "))
            if graph.dfs(start, goal):
                print("Goal node found using DFS")
            else:
                print("Goal node not found")
```

```
        elif choice == 2:
            start = int(input("Enter start node: "))
            goal = int(input("Enter goal node: "))
            depth_limit = int(input("Enter depth limit: "))
            if graph.dls(start, goal, depth_limit):
                print("Goal node found using DLS")
            else:
                print("Goal node not found within depth limit")
        elif choice == 3:
            start = int(input("Enter start node: "))
            goal = int(input("Enter goal node: "))
            if graph.dfid(start, goal):
                print("Goal node found using DFID")
            else:
                print("Goal node not found")
        elif choice == 4:
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please try again.")

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

Output:-

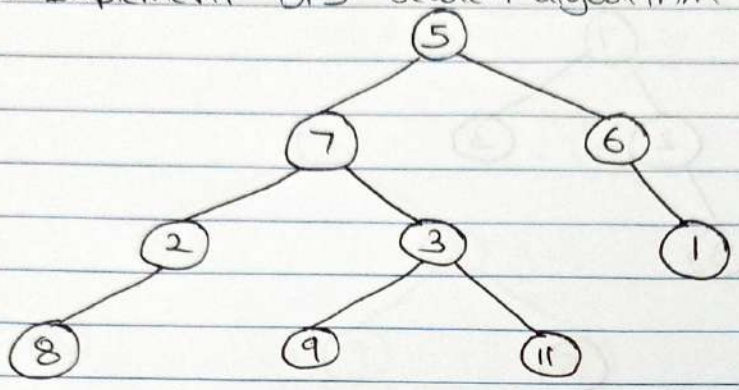

Run
DFS
C:\Users\sanja\AppData\Local\Programs\Python\Python311\python.exe C:\Users\sanja\PycharmProjects\pythonProject\DFS.py
Select Search Algorithm:
1. Depth-First Search (DFS)
2. Depth-Limited Search (DLS)
3. Iterative Deepening Depth-First Search (DFID)
4. Exit
Enter your choice: 1
Enter start node: 0
Enter goal node: 6
Goal node found using DFS
Select Search Algorithm:
1. Depth-First Search (DFS)
2. Depth-Limited Search (DLS)
3. Iterative Deepening Depth-First Search (DFID)
4. Exit
Enter your choice: 2
Enter start node: 0
Enter goal node: 6
Enter depth limit: 3
Goal node found using DLS
Select Search Algorithm:
1. Depth-First Search (DFS)
2. Depth-Limited Search (DLS)
3. Iterative Deepening Depth-First Search (DFID)
4. Exit
Enter your choice: 3
Enter start node: 0
Enter goal node: 6
Goal node found using DFID

Arnav Malvia
2103109
C23
AI

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment :- 2

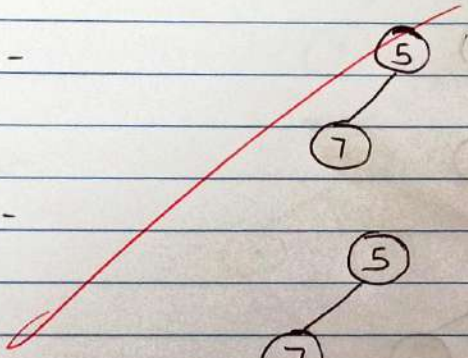
Aim:- Implement DFS search algorithm in Python.



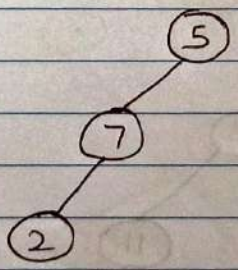
In the above graph find node (11) using DFS.

Root Node = (5)

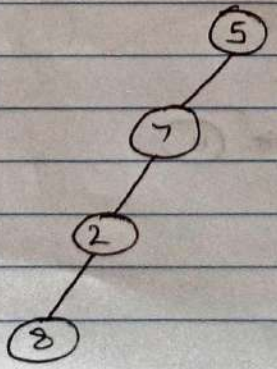
Step 1-



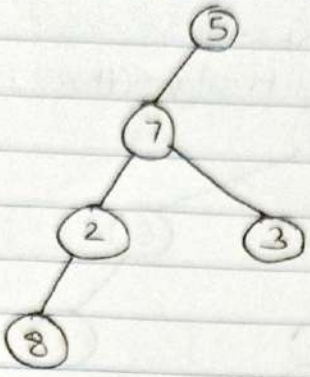
Step 2-



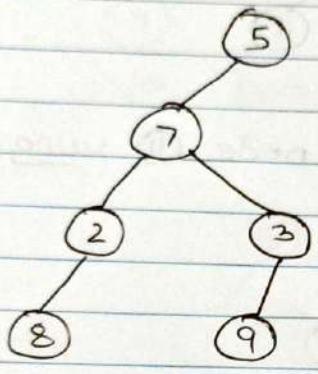
Step 3-



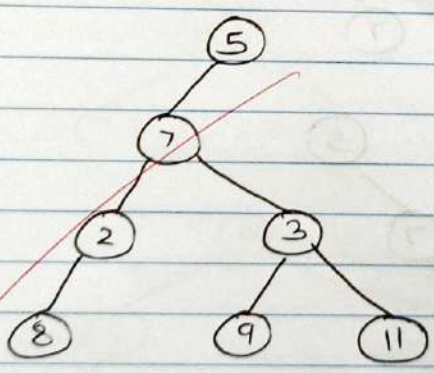
Step 4 -



Step 5 -



Step 6 -



Found node (11)

(5) → (7) → (3) → (11)

(BT) 8 / 8 1 4 12 4

Name : Arnav Malvia
Roll No. : 2103109
Batch : C23

Experiment 3 :-

Aim:- Implement BFS/UCS algorithm in Python

Theory:-

BFS:

In this article, we will discuss the BFS algorithm in the data structure. Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Applications of BFS algorithm :

The applications of breadth-first-algorithm are given as follows -

- BFS can be used to find the neighboring locations from a given source location.
- In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
- BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.
- BFS is used to determine the shortest path and minimum spanning tree.
- BFS is also used in Cheney's technique to duplicate the garbage collection.

- It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

Algorithm :

The steps involved in the BFS algorithm to explore a graph are given as follows-

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

Step 6: EXIT

UCS :

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Code :-

```
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."
```

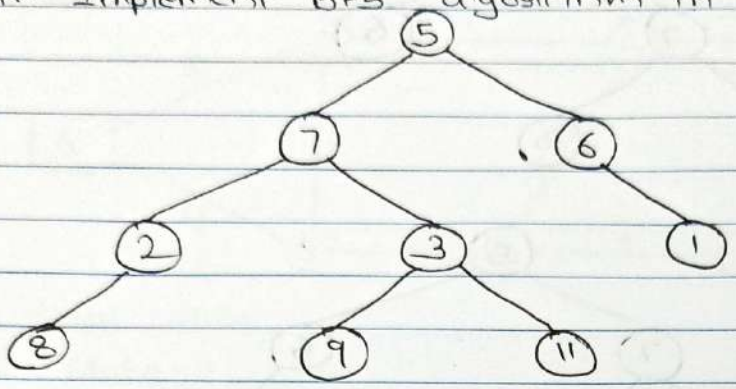
Output:-


Run
BFS
C:\Users\sanja\AppData\Local\Programs\Python\Python311\python.exe C:\Users\sanja\PycharmProjects\pythonProject\BFS.py
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:

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE
Abhay Malvia
2103109
C23
AI

Experiment :- 3

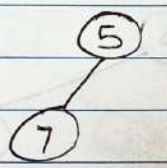
Aim :- Implement BFS algorithm in Python.



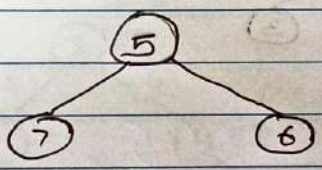
In the above graph find (8) using BFS.

Root Node = (5)

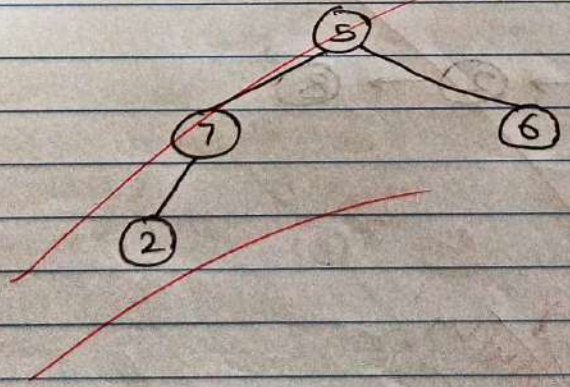
Step 1-



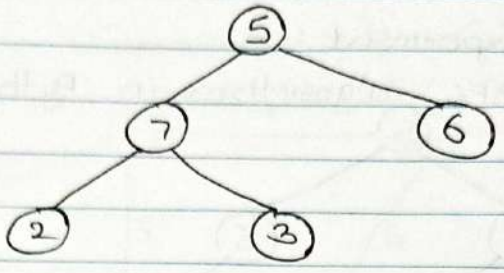
Step 2-



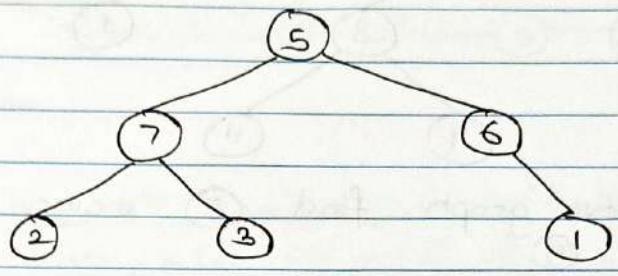
Step 3 -



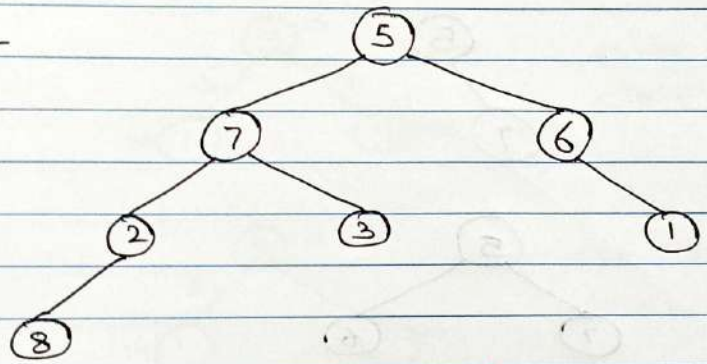
Step 4 -



Step 5 -



Step 6 -



Node (8) found

(5) → (7) → (2) → (8)

Name : Arnav Malvia
Roll No. : 2103109
Batch : C23

Experiment 4 :-

Aim:- Implement Greedy Best First Search/A* algorithm in Python

Theory:-

Greedy Search:

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function.

Algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.

- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

A*:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Code :-

```
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()
```

Tree Used:

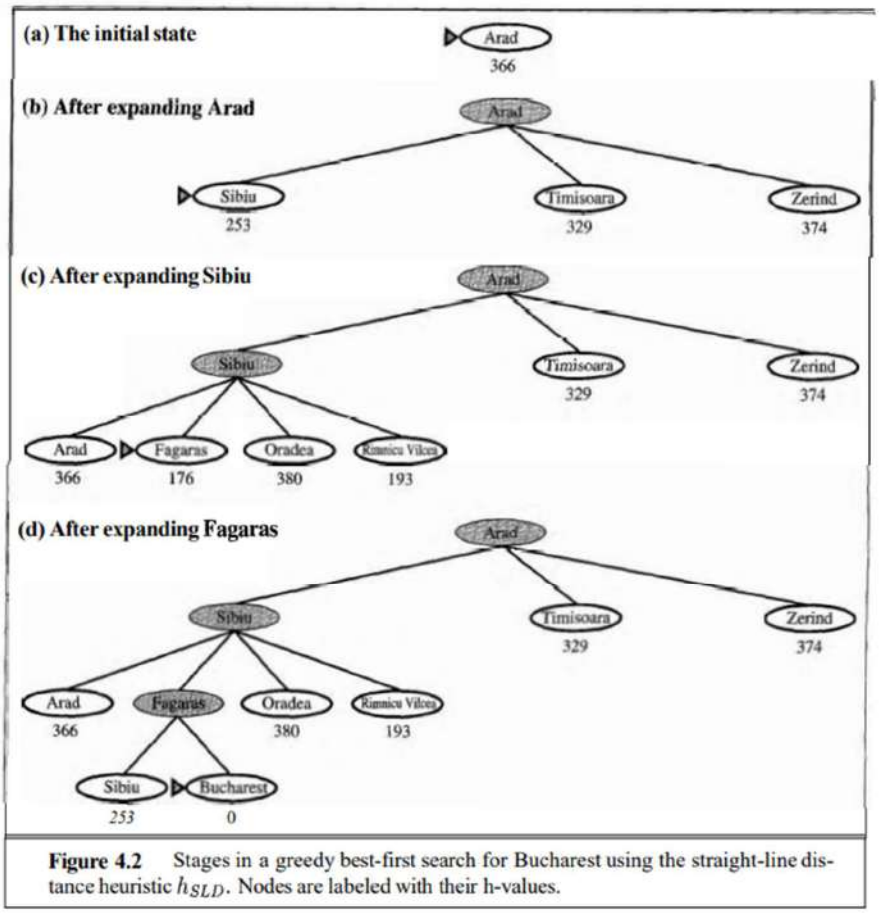
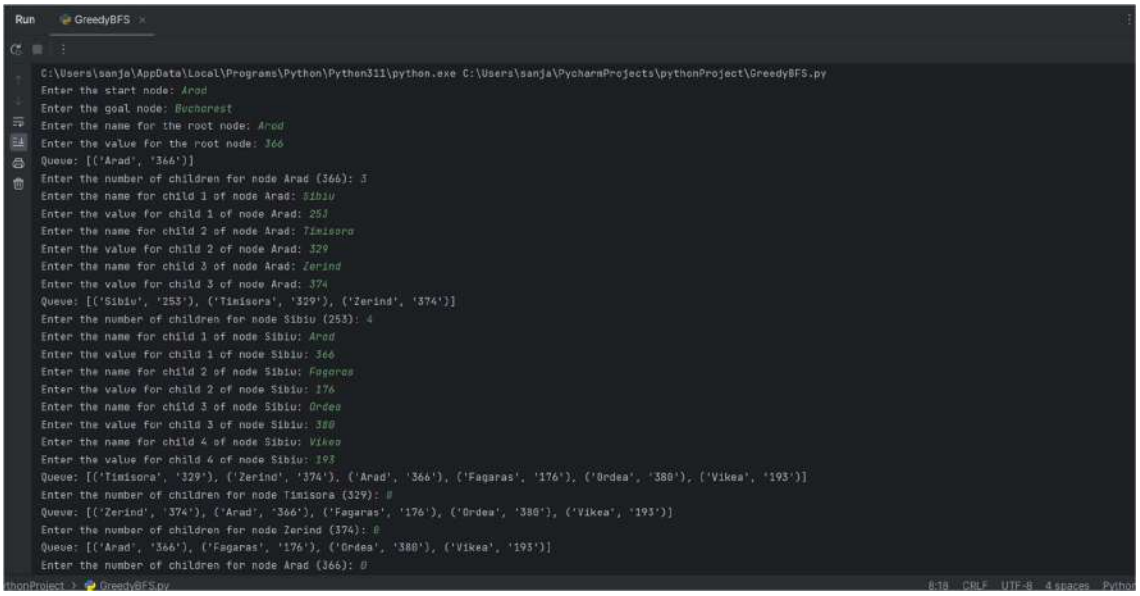


Figure 4.2 Stages in a greedy best-first search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

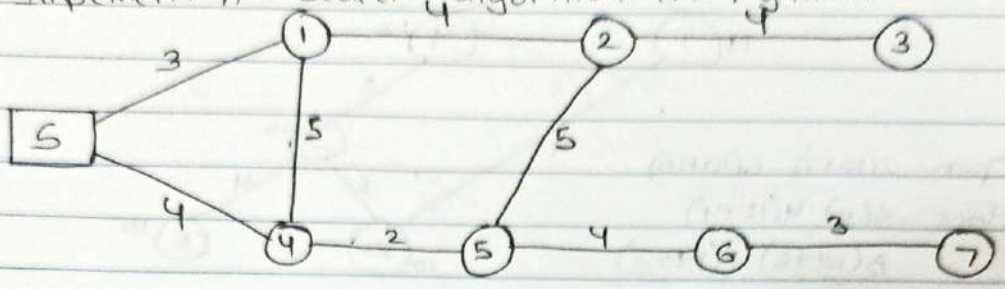
Output:-



Abhinav Malvia
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE
2103109
C23
AI

Experiment :- 4 (A*)

Aim:- Implement A* search algorithm in Python.

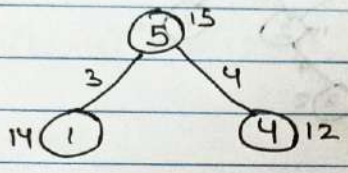


S → Start state

Goal State = 7

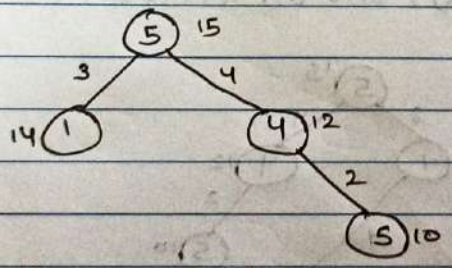
Find path using A* search, given $h(1)=14$, $h(2)=10$, $h(3)=8$, $h(4)=12$, $h(5)=10$, $h(6)=10$, $h(5)=15$

Step 1 -



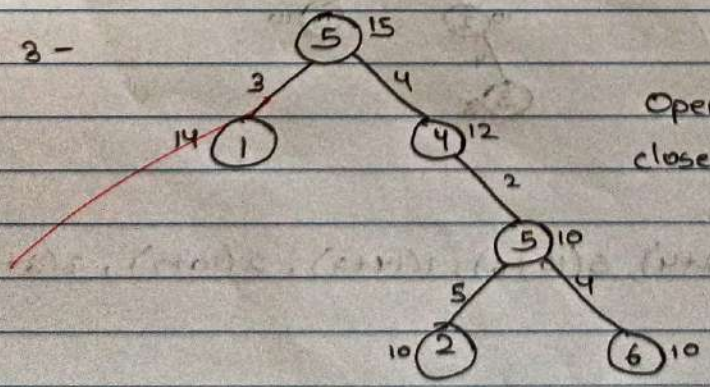
Open : 4(12+4), 1(14+3)
close : S(15)

Step 2 -



Open: 5(10+6), 1(14+3)
close: S(15), 4(12+4)

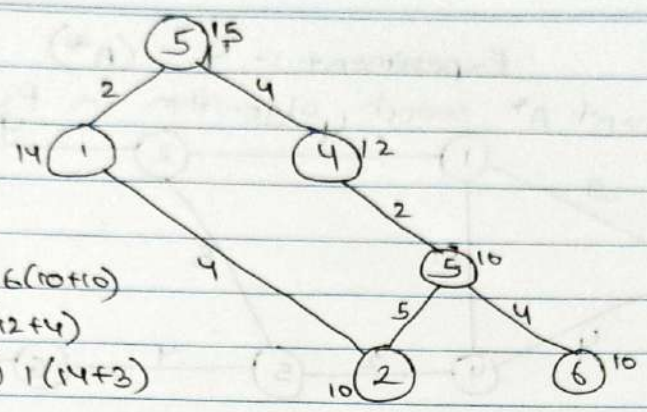
Step 3 -



Open: 1(14+3), 6(10+10), 2(10+11)
close: S(15), 4(12+4) 5(10+6)

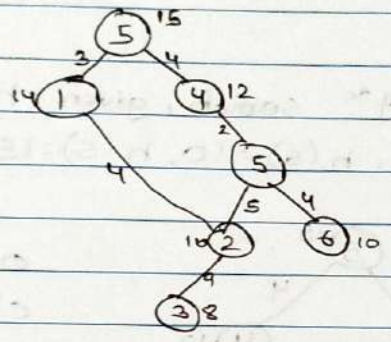
THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Step 4 -



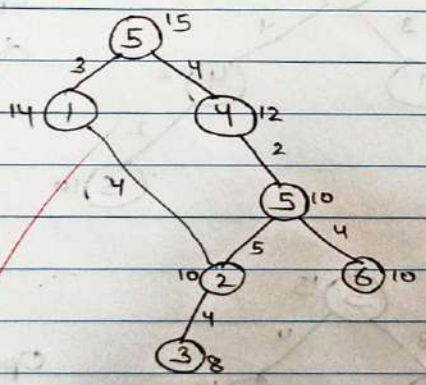
Open : 2(10+7) 6(10+10)
close : 5(5) 4(12+4)
5(10+6) 1(14+3)

Step 5 -



Open : 3(8+11), 6(10+10)
close : 5(15), 4(12+4), 5(10+6), 1(14+3), 2(10+7)

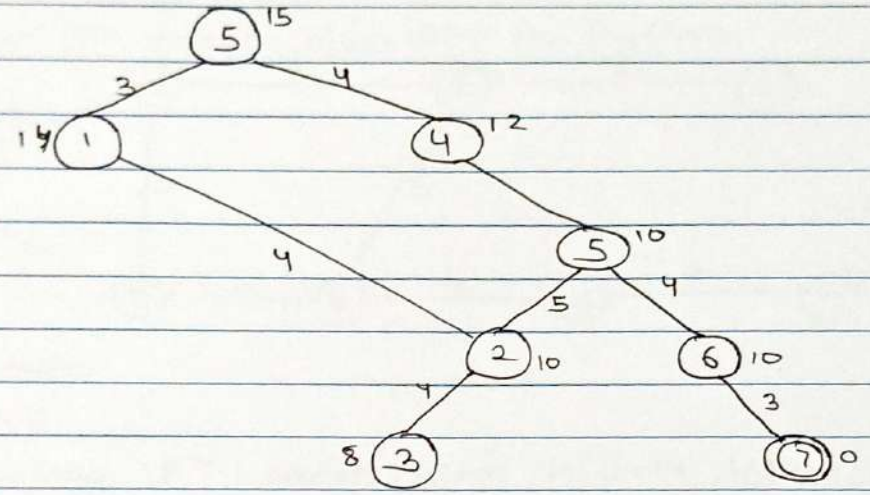
Step 6 -



Open : 6(10+10)
close : 5(15), 4(12+4), 5(10+6), 1(14+3), 2(10+7), 3(8+11)

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Step 7-



Open: 7(0+13)

close: 5(15), 4(12+4), 5(10+0), 1(14+3), 2(10+7), 5(8+4), 6(10+10)

BF & Solution

Name : Arnav Malvia
Roll No. : 2103109
Batch : C23

Experiment 5 :-

Aim:- Implement Genetics/Hill Climbing in Python

Theory:-

Genetics:

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.
Genetic algorithms simulate the process of natural selection which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate "survival of the fittest" among individuals of consecutive generations to solve a problem. 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 character/integer/float/bits. This string is analogous to the Chromosome.

Advantages of Genetic Algorithm

- The parallel capabilities of genetic algorithms are best.
- It helps in optimizing various problems such as discrete functions, multi-objective problems, and continuous functions.
- It provides a solution for a problem that improves over time.
- A genetic algorithm does not need derivative information.

Limitations of Genetic Algorithms

- Genetic algorithms are not efficient algorithms for solving simple problems.
- It does not guarantee the quality of the final solution to a problem.

- Repetitive calculation of fitness values may generate some computational challenges.

Hill Climbing:

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman. It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that. A node of hill climbing algorithm has two components which are state and value. Hill Climbing is mostly used when a good heuristic is available. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Code :-

```
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()

    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():
    # Initialize population with specific chromosomes
    population = [
        Chromosome(['1', '2', '3', '4']),
        Chromosome(['5', '6', '7', '8']),
        Chromosome(['9', '0', '1', '2']),
        Chromosome(['3', '4', '5', '6']),
        Chromosome(['7', '8', '9', '0']),
        Chromosome(['1', '2', '3', '4'])
    ]

    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(population, 1):
        print(f"Chromosome {i}: {chromosome.genes}")

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

    print(f"Best 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(int(gene) for gene in best_solution.genes)}")
    print(f"Number of Iterations: {iteration}")


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

Output:-

Aarnav Malvia
2103109
C23
AI

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment :- 5

Aim :- Implement Genetic algorithm in Python

Theory :-
Genetic algorithms are based on the ideas of natural selection and genetics.

Important Functions:-
1) Initialisation -
Each solution is typically represented as a chromosome or genotype, which encodes the candidate solution.

2) Fitness Function -
A fitness function evaluates how good each solution in population is assigning a numerical value, known as f, to each solution.

3) Selection -
In the selection phase individuals from the current population are chosen to be parents for the next generation. Common selection techniques include roulette wheel selection, tournament selection and rank-based selection.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

4) Crossover (Recombination)-
It is the process of combining genetic material from two parent solutions to produce off spring solutions.

5) Mutation-
Mutations are typically involved in randomly altering some components of the chromosome such as flipping its bits or swapping genes.

6) Termination-
The termination crieteria determines when to stop the algorithm such as reaching a maximum number of generations, achieving a satisfactory solution quality or running out of computational resources.

Conclusion:- Hence we were able to understand and successfully implement Generating Algorithm in Python.

BP 9/4/24.

Aaryan Malvia
2103109
C23
AI

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment :- 6

Aim :- Knowledge Representation and Knowledge Base for wumpus world.

Q1 Represent the following sentences in First order logic using consistent vocabulary.

Vocabulary:

Takes (x, c, s) : Student x takes course c in semester s.

Passes (x, c, s) : Student x passes course c in semester s.

Score (x, c, s) : Score obtained by student x in course c in semester s.

F and G : F stands for French course, G stands for Greek course.

Subject (c, f) : Subject of course c is field f.

Buys (x, y, z) : x ~~bys~~ ~~by~~ buys y from z

Sells (x, y, z) : x sells y to z

Shaves (x, y) : Person x shaves person y

Born (x, c) : Person x is born in country c.

Parent (x, y) : x is a parent of y

Citizen (x, c, r) : x is a citizen of country c for reason r

Resident (x, c) : x is a resident of country c.

Fools (x, y, t) : person x fools person y at time t.

Predicates:

Student (x), Person (x), Man (x), Barber (x), Expensive (x), Agent (x), Insured (x), Smart (x), politician (x).

a. Some students took French in Spring 2001
$\exists x$ student$(x) \wedge$ Takes$(x, F,$ Spring 2001$)$

b. Every student who takes French passes it
$\forall x, s$ student$(x) \wedge$ Takes$(x, F, s) \rightarrow$ Passes(x, F, s)

c. Only one student took greek in spring 2001
$\exists x$ student$(x) \wedge$ takes$(x, G,$ Spring 2001$) \wedge \forall y$
$y \neq x \rightarrow \sim$Takes$(y, G,$ spring 2001$)$

d. The best score in greek is always higher than the best score in French.
$\forall s \; \exists x \; \forall y$ score$(x, G, s) >$ score(y, F, s)

e. Every person who buys a policy is smart.
$\forall x$ Person$(x) \wedge (\exists y, z$ Policy$(y)) \wedge$ Buys$(x, y, z) \rightarrow$ Smart(x)

f. No person buys an expensive policy
$\forall x, y, z$ Person$(x) \wedge$ Policy$(y) \wedge$ Expensive$(y) \rightarrow \sim$Buys(x, y, z)

g. There is an agent who sells policies only to people who ~~are~~ are not insured.
$\exists x$ Agent$(x) \wedge \forall y; z$ Policy$(y) \wedge$ sells$(x, y, z) \rightarrow$ (Person$(z) \wedge \sim$Insured(z))

h. There is a barber who shaves all men who do not shave themselves.
$\exists x$ Barber$(x) \wedge \forall y$ Man$(y) \wedge \sim$shaves$(y, y) \rightarrow$ shaves(x, y)

i. A person born in UK each of whose ~~patients~~ parents is a UK citizen or a UK resident, is a UK citizen by birth.
$\forall x$ Person$(x) \wedge$ Born$(x,$ UK$) \wedge (\forall y,$ Parents$(y, x) \rightarrow$
$((\exists s$ (citizen$(y,$ UK$, s)) \vee$
Resident$(y,$ UK$))) \rightarrow$ citizen$(x,$ UK, Birth$)$

j. A person born outside UK, one of those parents is a UK ~~one~~ citizen by birth, is a UK citizen by descent.
$\forall x$ Person$(x) \wedge \sim$Born$(x,$ UK$) \wedge (\exists y$ Parent$(y, x) \wedge$ citizen$(y,$ UK, British$)) \rightarrow$ (citizen$(x,$ UK, descent)

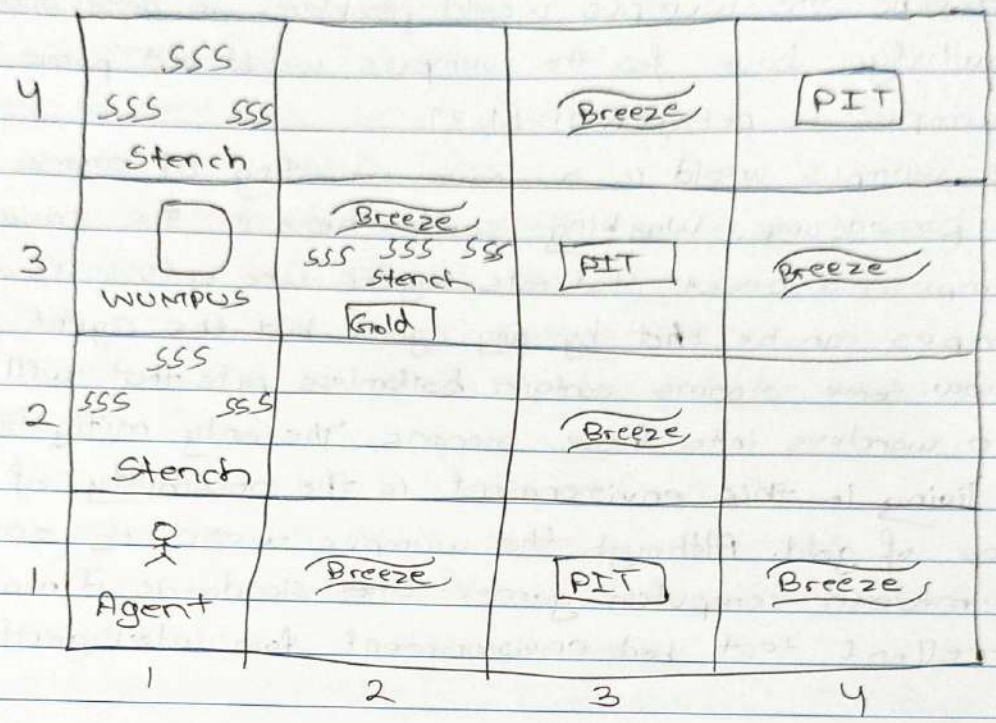
THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Q2 Describe the wumpus world problem in detail and create a knowledge base for the wumpus world and prove that the wumpus is present in [1, 3].

→ The wumpus world is a cave consisting of rooms connected by passageways. Working somewhere in the cave is the wumpus, a beast that eats anyone who enters its room. The wumpus can be shot by any agent, but the agent has only one arrow. Some rooms contain bottomless pits that will trap anyone who wanders into these rooms. The only mitigating feature of living in this environment is the possibility of finding a heap of gold. Although the wumpus world is rather tame by modern computer game ~~star~~ standards it makes an excellent test bed environment for intelligent agents.

It is a classic AI problem where an agent operates in a grid based environment containing hazards (such as pits) and a dangerous creature called the wumpus. In this problem the agent (player) starts at a certain location and has to navigate the world to reach a goal. The world is populated with various elements having pits, gold, wumpus, and walls each with its own characteristics and behaviours.

The goal of the agent is to find, the gold while avoiding the wumpus and pits, which can kill the agent. The agent has limited actions.



Propositional Variables:-

P_{ij} → there is a pit at (i,j)

B_{ij} → there is a breeze at (i,j)

W_{ij} → there is a wumpus in the square (i,j)

S_{ij} → there is a stench at (i,j)

V_{ij} → square (i,j) is visited

G_{ij} → there is gold at (i,j)

OK_{ij} → Room is safe

Propositional Rules for the wumpus word:-

$(R_1) \rightarrow S_{11} \rightarrow \sim W_{11} \wedge \sim W_{12} \wedge \sim W_{21}$

$(R_2) \rightarrow S_{21} \rightarrow \sim W_{11} \wedge \sim W_{21} \wedge W_{22} \wedge \sim W_{31}$

$(R_3) \rightarrow S_{12} \rightarrow \sim W_{11} \wedge \rightarrow W_{12} \wedge \rightarrow W_{23} \wedge \sim W_{13}$

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

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

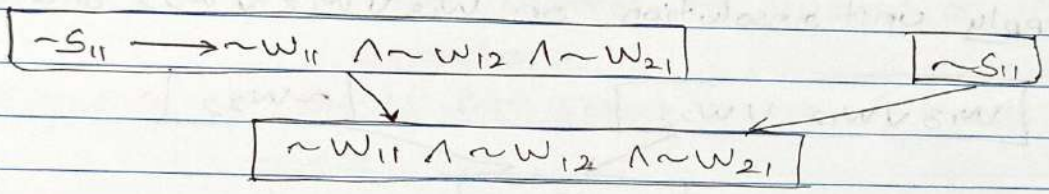
Representation of knowledge base for wumpus world when agent moves from room [1,1] to room [2,1]

$\sim W_{11}$	$\sim S_{11}$	$\sim P_{11}$	$\sim B_{11}$	$-\ q_{11}$	$\sim V_{11}$	OK_{11}
$\sim W_{12}$	...	$\sim P_{12}$	...	...	$-\ V_{12}$	OK_{22}
$\sim W_{21}$	$\sim S_{21}$	$\sim P_{21}$	B_{21}	$\sim G_{21}$	V_{21}	OK_{21}

Proof that wumpus is in the room (1,3)

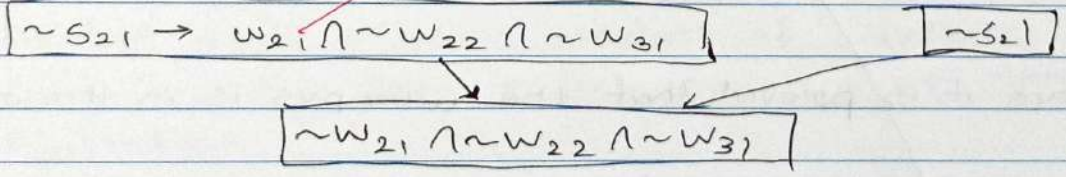
Using propositional and inference rules,

1. Apply Modus Ponens with $\sim S_{11}$ and R_1



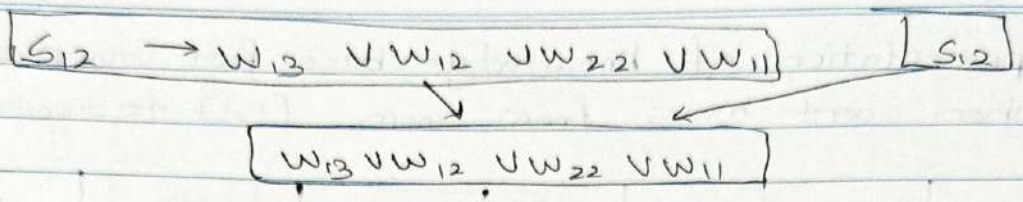
2. Apply and - elimination Rule
we get
$\sim W_{11}$, $\sim W_{12}$ and $\sim W_{21}$

3. Apply Modus Ponens to $\sim S_{21}$ and R_2

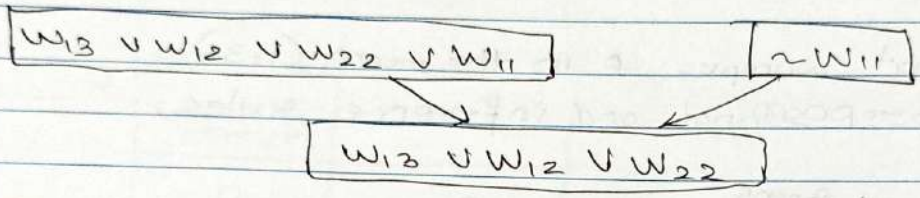


4. Apply and ~ Elimination rule
we get $\sim W_{21}$, $\sim W_{22}$ and $\sim W_{31}$

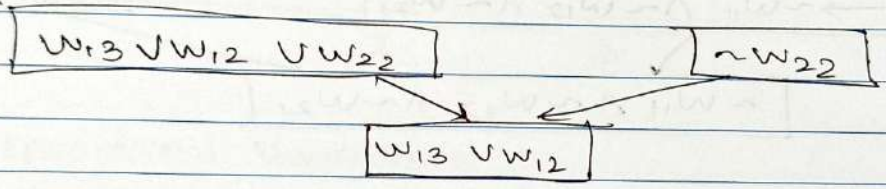
5. Apply modus Ponens to S_{12} and R_4



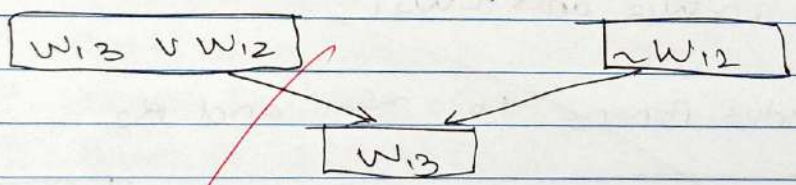
6. Apply unit resolution on $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ and $\sim W_{11}$:



7. Apply unit resolution on $W_{13} \vee W_{12} \vee W_{22}$ and $\sim W_{22}$



8. Apply unit Resolution on $W_{13} \vee W_{12}$ and $\sim W_{12}$



Hence it is proved that the wumpus is in the room (1,3)

Conclusion:- Hence we have understood knowledge representation and knowledge base for wumpus world.

8/4/24

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Aarnav Malvia
2103109
C23
AI

Experiment :- 7

Aim :- Planning for Blocks World problem

Theory :-

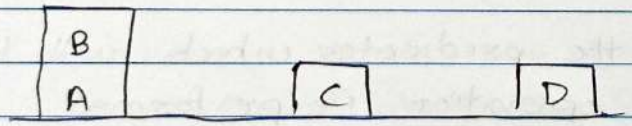
The Blocks World (BW) consists of a finite number of blocks stalled into towers on a table large enough to hold them all at the ~~poor~~ posioning of the towers on the table is irrelevant.

The BW planning problem is to form an initial state of the blocks into a goal state by moving one block at a time from the top of a tower onto another tower or to the table. The optimal BW planning problem is to do so in a minmal number of moves.

Example :-

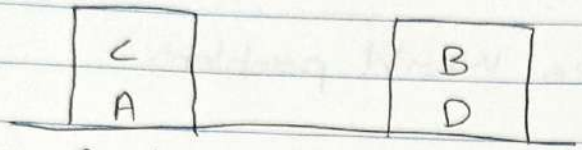
List of predicates :- ON(A,B) : Block A is on B
ONTABLE(A) : A is on table
CLEAR(A) : Nothing is on top of A
HOLDING(A) : Arm is holding A
ARM EMPTY : Arm is holding nothing

Inital state:



⇒ ON(B,A) ∧ ONTABLE(A) ∧ ONTABLE(C) ∧ ONTABLE(D) ∧ CLEAR(B) ∧ CLEAR(C) ∧ CLEAR(D) ∧ ARM EMPTY

Goal State:



⇒ ON(C,A) ∧ ON(B,D) ∧ ONTABLE(A) ∧ ONTABLE(D) ∧ CLEAR(B) ∧ CLEAR(C) ∧ ARMEMPTY

- The Robot arm can perform 4 operations:
1) STACK(X,Y) = Stacking Block X on Block Y
2) UNSTACK(X,Y) = Picking up block X which is on top of block Y.
3) PICKUP(X) = Picking up block X which is on top of the table.
4) PUTDOWN(X) = Put Block X on the table.

All the 4 conditions/operations have certain preconditions which need to be satisfied to perform the same.
The effect of these operations is represented using two lists: ADD and DELETE

ADD List - Contains the predicates which will become true once the operation is performed.

DELETE List - Contains the predicates which will leave to be true once the operation is performed.

Operators	Preconditions	DELETE	Add
STACK(x,Y)	CLEAR(Y) ∧ HOLDING(x)	CLEAR(Y) HOLDING(X)	ARMEMPTY ON(x,Y)
ONSTACK(x,Y)	ARMEMPTY ∧ ON(x,Y) ∧ CLEAR(x)	ARMEMPTY∧ ON(x,Y)	HOLDING(x) ∧ CLEAR(Y)
PICKUP(x)	CLEAR(x) ∧ ONTABLE(x)∧ ARMEMPTY	ONTABLE(x) ∧ ARMEMPTY	HOLDING(x)
PUTDOWN(x)	HOLDING(x)	HOLDING(x)	ONTABLE(x) ∧ ARMEMPTY

Once the operation is performed these predicates will cease to be true, thus they are individual in DELETE List as well.

On the other hand, once the operation is performed, the robot ~~wi~~ arm will be free (ARMEMPTY) and the block x will be on top of Y (ON(x,Y)).

Conclusion:- Hence we have understood the planning for blocks world problem.

SP Gurav

Name : Arnav Malvia
Roll No. : 2103109
Batch : C23

Experiment 8 :-

Aim:- Implement Family tree using prolog

Theory:-

Prolog stands for programming in logic. In the logic programming paradigm, prolog language is most widely available. Prolog is a declarative language, which means that a program consists of data based on the facts and rules (Logical relationship) rather than computing how to find a solution. A logical relationship describes the relationships which hold for the given application.

To obtain the solution, the user asks a question rather than running a program. When a user asks a question, then to determine the answer, the run time system searches through the database of facts and rules.

The first Prolog was 'Marseille Prolog', which is based on work by Colmerauer. The major example of fourth-generation programming language was prolog. It supports the declarative programming paradigm. In 1981, a Japanese computer Project of 5^{th} generation was announced. After that, it was adopted Prolog as a development language. In this tutorial, the program was written in the 'Standard' Edinburgh Prolog. Prologs of PrologII family are the other kind of prologs which are descendants of Marseille Prolog.

Prolog features are 'Logical variable', which means that they behave like uniform data structure, a backtracking strategy to search for proofs, a pattern-matching facility, mathematical variable, and input and out are interchangeable. To deduce the answer, there will be more than one way. In such case, the run time system will be asked to find another solution. To generate another solution, use the backtracking strategy. Prolog is a weakly typed language with static scope rules and dynamic type checking. Prolog is a declarative language that means we can specify what problem we want to solve rather than how to solve it.

Prolog is used in some areas like database, natural language processing, artificial intelligence, but it is pretty useless in some areas like a numerical algorithm or instance graphics. In artificial intelligence applications, prolog is used. The

artificial intelligence applications can be automated reasoning systems, natural language interfaces, and expert systems. The expert system consists of an interface engine and a database of facts. The prolog's run time system provides the service of an interface engine.

A basic logic programming environment has no literal values. An identifier with upper case letters and other identifiers denote variables. Identifiers that start with lower-case letters denote data values. The basic Prolog elements are typeless. The most implementations of prolog have been enhanced to include integer value, characters, and operations. The Mechanism of prolog describes the tuples and lists.Functional programming language and prolog have some similarities like Hugs. A logic program is used to consist of relation definition. A functional programming language is used to consist of a sequence of function definitions. Both the logical programming and functional programming rely heavily on recursive definitions.

The applications of PROLOG are as follows:

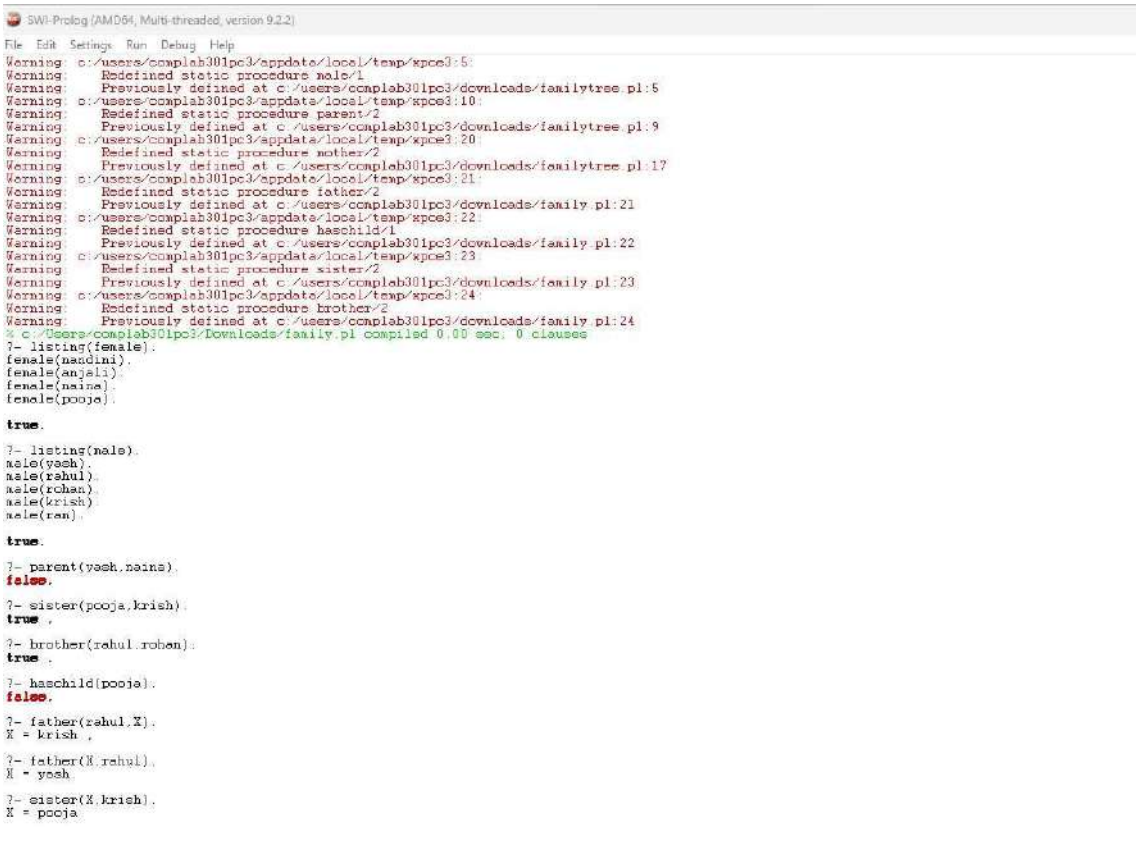
- Specification Language
- Robot Planning
- Natural language understanding
- Machine Learning
- Problem Solving
- Intelligent Database retrieval
- Expert System
- Automated Reasoning

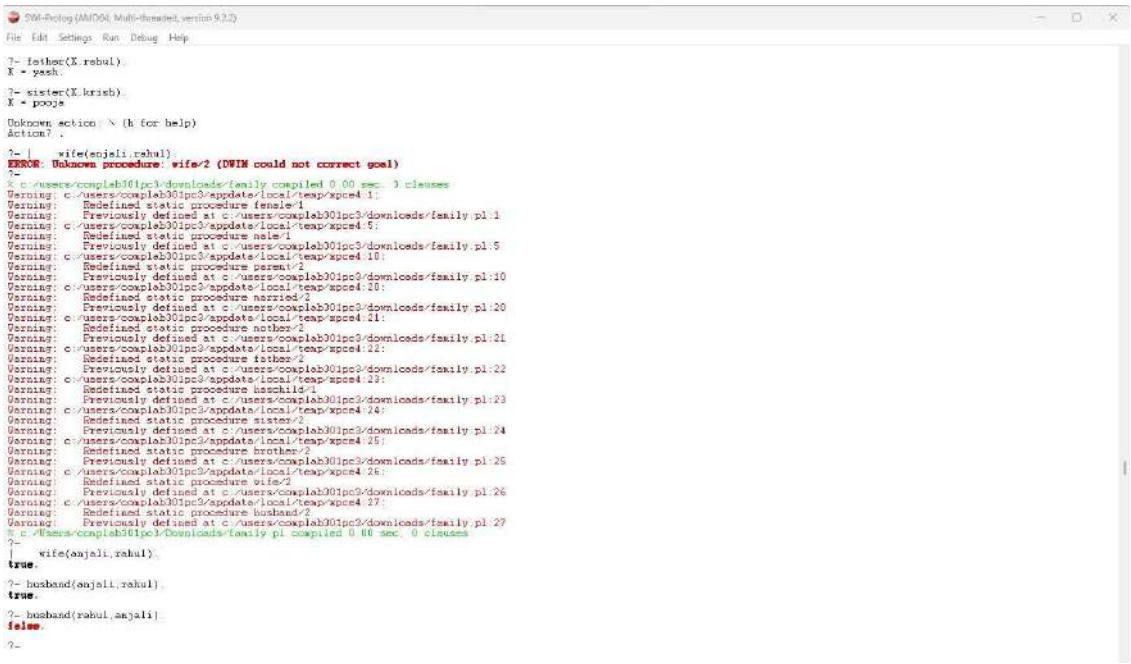
Code :-

```
female(nandini) .

female(anjali) .

female(naina) .

female(pooja) .

male(yash) .
```

```
male(rahul) .
male(rohan) .
male(krish) .
male(ram) .
parent(nandini,rahul) .
parent(nandini,rohan) .
parent(yash,rahul) .
parent(yash,rohan) .
parent(anjali,krish) .
parent(anjali,pooja) .
parent(rahul,krish) .
parent(rahul,pooja) .
parent(rohan,ram) .
parent(naina,ram) .
married(anjali,rahul).
mother(X,Y) :- parent(X,Y),female(X).
father(X,Y) :- parent(X,Y),male(X).
haschild(X) :- parent(X,_) .
sister(X,Y) :- parent(Z,X),parent(Z,X),female(X),X\==Y .
brother(X,Y) :- parent(Z,X),parent(Z,Y),male(X),X\==y.
wife(X,Y):- married(X,Y),female(X).
husband(X,Y):- married(X,Y),male(Y).
```

Output:-


SWI-Prolog (AMD64, Multi-threaded, version 9.2.2)
File Edit Settings Run Debug Help
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce3:5:
Warning: Redefined static procedure male/1
Warning: Previously defined at c:/users/comlab301pc3/downloads/familytree.pl:5
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce3:10:
Warning: Redefined static procedure parent/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/familytree.pl:9
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce3:20:
Warning: Redefined static procedure mother/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/familytree.pl:17
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce3:21:
Warning: Redefined static procedure father/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:21
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce3:22:
Warning: Redefined static procedure haschild/1
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:22
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce3:23:
Warning: Redefined static procedure sister/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:23
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce3:24:
Warning: Redefined static procedure brother/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:24
% c:/Users/comlab301pc3/Downloads/family.pl compiled 0.00 sec, 0 clauses
?- listing(female).
female(nandini).
female(anjali).
female(naina).
female(pooja).
true.
?- listing(male).
male(yash).
male(rahul).
male(rohan).
male(krish).
male(ran).
true.
?- parent(yash,naina).
false.
?- sister(pooja,krish).
true .
?- brother(rahul,rohan).
true .
?- haschild(pooja).
false.
?- father(rahul,X).
X = krish ;
?- father(X,rahul).
X = yash
?- sister(X,krish).
X = pooja


SWI-Prolog (AMD64, Multi-threaded, version 9.2.2)
File Edit Settings Run Debug Help
?- father(X,rahul).
X = yash
?- sister(X,krish).
X = pooja
Unknown action: ^ (h for help)
Action? .
?- | wife(anjali,rahul).
ERROR: Unknown procedure: wife/2 (DWIM could not correct goal)
?-
% c:/users/comlab301pc3/downloads/family compiled 0.00 sec, 3 clauses
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:1:
Warning: Redefined static procedure female/1
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:1
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:5:
Warning: Redefined static procedure male/1
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:5
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:10:
Warning: Redefined static procedure parent/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:10
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:20:
Warning: Redefined static procedure married/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:20
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:21:
Warning: Redefined static procedure mother/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:21
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:22:
Warning: Redefined static procedure father/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:22
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:23:
Warning: Redefined static procedure haschild/1
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:23
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:24:
Warning: Redefined static procedure sister/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:24
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:25:
Warning: Redefined static procedure brother/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:25
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:26:
Warning: Redefined static procedure wife/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:26
Warning: c:/users/comlab301pc3/appdata/local/temp/xpce4:27:
Warning: Redefined static procedure husband/2
Warning: Previously defined at c:/users/comlab301pc3/downloads/family.pl:27
% c:/Users/comlab301pc3/Downloads/family.pl compiled 0.00 sec, 0 clauses
?-
| wife(anjali,rahul).
true.
?- husband(anjali,rahul).
true.
?- husband(rahul,anjali).
false.
?-

Arnav Malvi
2103109
C23
AI

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment :- 1

Introduction to AI, Problem formulation and PEAS properties.

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

i) Acting humanly
ii) Thinking humanly
iii) Acting rationally
iv) Thinking rationally

→ i) Acting humanly:

The study of how to make computers do things at which, at the moment people are better.

Example of this ~~is~~ definition is the Turing Test : During the test, one of the humans functions as the questioner, while the second human and computer act as respondants.

ii) Thinking humanly:

The automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning, etc.

iii) Acting rationally:

Computational intelligence is the study of design of intelligent agents."

Example of this definition is a rational agent. They operate autonomously and perceive their environment, persist over a prolonged time period adapt to change and create and pursue goals.

iv) Thinking rationally:
The study of the computations that make it possible to perceive, reason and act.

Q2 Explain the components of AI system in detail.

→ AI is a vast digital field for research and it has got applications in almost all possible domains. Research in AI has focused mainly on the following components of AI:-

i) Perception:-
In order to work in the environment, intelligent agents need to scan the environment ~~of the~~ and the various objects in it by means of different sense organs real or artificial. Agent scans the environment using sense organs like camera, temperature sensor, etc. This is called perception.

ii) Knowledge representation:-
The information obtained from the environment through sensors may not be in the format required by the system. Hence, it needs to be represented in standards format for further processing like learning, various pattern, deducing inference, comparing with past objects etc. There are various knowledge representation techniques like propositional logic and first order logic.

iii) Learning:-
Learning is a very ~~esse~~essential part of AI of it happens is a number of different forms. The simplest form of learning is by trial and error. In this form the program remembers the action that has given

the desired output and discards the other trial actions and learns by itself.
It is also called unsupervised learning.

iv) Reasoning :-
Reasoning is also called as logic or generating judgements from the given set of facts. The reasoning is carried out based on a strict rule of validity to perform a specific task. Reasoning can be of two types, deductive or inductive. The deductive reasoning is in which the truth of the premissed guarantees the truth of the conclusion while in case of inductive reasoning, the truth of the premises supports the conclusion but it cannot be fully dependent on the premises.

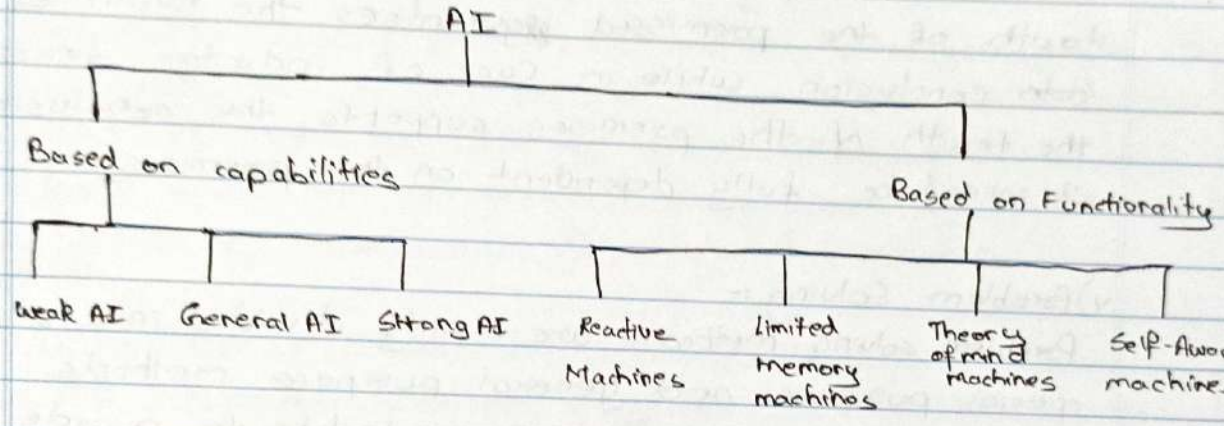
v) Problem Solving :-
Problem solving methods are mainly divided into 2 types: special purpose and general purpose methods.
General purpose methods are applicable to a wide range of problems, one used in AI in means-end analysis, which invests the step-by step reduction of the difference between the current state and the goal state. Special purpose methods are customised to solve a particular type of problem.

vi) Natural Language Processing :-
NLP involves machines or robots to understand and process the language that human speaks and infer knowledge from the speech input. It also involves the active participation from a machine in the form of dialog.

NLP aims at the text or verbal output from the machine or robot. The input and output of an NLP system can be speech and written text respectively.

Q3 Write a short note on categorization of AI

→ Artificial Intelligence can be divided in various types on two types of main classification - based on capabilities. and based on functionalities.



• BASED ON CAPABILITIES:-

i) Weak AI or Narrow AI:

Narrow AI is a type of AI which is able to perform a dedicated task with intelligence. The most common and currently available AI is narrow AI can fail in the world of Artificial Intelligence.

Narrow AI cannot perform beyond its field or limitations, as it is only trained for one specific task. Hence it is also termed as weak AI. Narrow AI can fail in unpredictable ways if it goes beyond its limits.

ii) General AI :-

General AI is a type of intelligence which could perform any intellectual task with efficiency like a human. The idea behind the general the general AI is to make such a system which could be smarter and think more a human by its own.

iii) Strong AI or Super AI :-

Super AI is a level of intelligence of systems at which machines could surpass human intelligence and can perform any task better than human with cognitive properties. It is an outcome of general AI.

- BASED ON FUNCTIONALITY :-

i) Reactive machines:

Purely reactive machines are the most basic types of Artificial Intelligence. Such AI systems do not store memories or past experiences for future actions. These machines only focuses on current scenarios and react on it as per possible best action.

ii) Limited memory machines:

Limited memory machines can store past experiences or some data for a short period of time. These machines can use stored data for a limited time period only.

iii) Theory of Mind machines:

Theory of Mind AI should understand human emotions, people, beliefs and be able to interact socially like humans. This type of AI machines are not yet developed.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q4 Explain problem formulation with the help of example.

→ Problem formulation is the step in problem definition that is used to understand and decide a course of action that needs to be considered to achieve a goal. Problem formulation in artificial intelligence organises several steps to formulate a target / goals which require a specific action to achieve a goal.
The problem can be defined formally in 5 components-
i) Initial State
ii) Actions
iii) Transition Model
iv) Goal Test
v) Path Cost.

Q5 Explain PEAS properties in detail.

→ PEAS stands for performance measure, environment, actuators and sensors. PEAS defines AI models and helps determine the task environment for an intelligent agent.

- Performance measure - It defines the success of an agent it evaluates the criteria that determines whether the system performs well.

- Environment - It refers to the external context in which an AI system operations. It encapsulates the physical and virtual surroundings, including other agents, objects and conditions.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

- Actuators - They are responsible for executing actions based on the directions made. They interact with the environment to bring about desired changes.
- Sensors - An agent observes and perceives its environment through sensors, sensors provide input data to the system, enabling it to make informed decisions.

Examples:

Agent	Performance measure	Environment	Actuators	Sensors
Vaccum cleaner	cleanliness, security, battery	Room, table, carpet, floors	wheels, brushes	Camera, sensors.

Q6 Describe different types of environments with suitable examples.

→ An environment in artificial intelligence is the surrounding of the agent.

There are several types of environments:-

i) Fully observable vs Partially observable:

When an agent sensor is capable to sense or access the complete state of an agent at each point in time, it is said to be a fully observable environment else it is partially observable. Maintaining a fully observable environment is easy as there is no need to keep track of the history of the surrounding.

eg: Chess - the board is fully observable, so are the opponents moves.

Driving - the environment is called unobservable, since no sensor in all environments.

ii) Deterministic vs Stochastic:

When a uniqueness in the agents current state completely determines the next state of the agent, the environment is said to be deterministic. The stochastic environment is random in nature which is not unique and cannot completely determine by the agent.

eg: Chess - there are only a few possible moves for a coin at the current state and these moves can be determined.

iii) Competitive vs Collaborative:

An agent is said to be in a competitive environment when it competes against another agent to optimise the output. Game of chess is competitive since agents compete with each other to win the game.

The agent is said to be collaborative when multiple agents cooperate to produce the desired output. eg: self driving cars.

iv) Single agent vs Multi agent:

When an environment has only one agent it is said to be single agent environment. Environment involving more than one agent is a multi agent environment. eg:- game of football is multi agent since there are 11 players.

v) Dynamic vs Static:

An environment that keeps constantly changing itself when the agent is up with some action is said to be dynamic. A roller coaster ride is dynamic. An idle environment with no change is static. eg: An empty house.

vi) Discrete vs Continuous:

If an environment consists of finite number of actions that can be deliberated in the environment to obtain the output, it is said to be a discrete environment.
The environment in which the actions are performed and cannot be numbered is said to be continuous.
eg:- chess is discrete since finite moves, self driving cars are continuous since their actions can be driving, parking, etc.

vii) Episodic vs Sequential:

In an episodic environment each of the agents actions is divided into atomic incidents or episodes. There is no dependency between current and previous incidents. In each incident agent receives input from the environment and then performs the corresponding action. eg: pick and place bot used to detect defective parts from conveyor belts.
In a sequential task environment, the previous decisions can affect all future decisions. The next action of the agent depends on what action he has taken previously.
eg: checkers game where the previous move can affect all the following moves.

AT 8/8/24

Arnav Malvia
2103109
C23
AI

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment :- 2

Q1 Discuss forward chaining and backward chaining algorithms.

→ Forward chaining :-

Foreward chaining employs a data driven approach to problem solving. It involves a logical reasoning, process similar to medical diagnosis.

Process :

Initial knowledge base - The process begins with a set of established facts or observations.

Rule application - The system posses a collection of inferance rules often structured as logical statements in the form of if-then conditions.

Iterative inference - The system iteratively applies these rules to the existing knowledge base. When the conditions watch the known facts, the conclusion (the then part) is added to the knowledge base, expanding the system's understanding. The cycle continues until a specific goal is achieved.

Common Applications :

i) Planning and scheduling
ii) System monitoring
iii) Medical diagnosis

→ Backward chaining :-

Backward chaining adopts a goal driven approach to problem solving. It mirrors the thought process employed during software debugging.

Process:
Defining the goal- The process starts with a clearly defined objective.
Rule Matching- The system consults its knowledge base; which comprises of rules that govern the interactions between the various components within the system analysed.
Subgoal Decomposition- The premises of identified rules transform into new subgoals. These symbols represent the conditions that must be satisfied for the overall goal to manifest.
Recursive goal exploration- System iteratively searches for additional rules where new symbols correspond to their conclusion. This process repeats till all subgoals can be matched with facts.

Common Applications:-
i) Configuration Tasks
ii) System Debugging
iii) Decision Support Systems

The law says that it is a crime for a American to sell weapons to hostile nations. The country Nono, The enemy of America has some missiles and all the missiles were sold to it by Colnel West, a America. West is a criminal using forwarded and backward chaining.

i) It is a crime for an American to sell weapons to hostile nations.
ii) Nono is an enemy of America.
iii) Enemy of America is hostile.

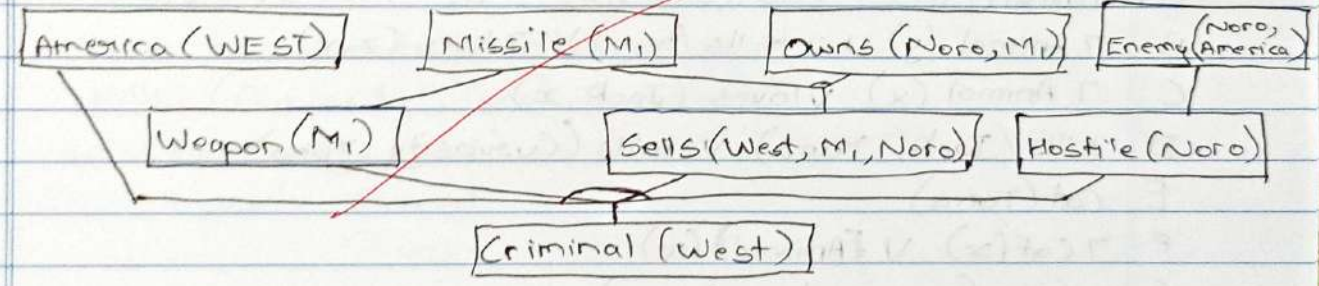
iv) Nono has some missiles.
v) Missile is a weapon.
vi) All missiles Nono has been sold by west
vii) West is an American.

Propositional Logic :-

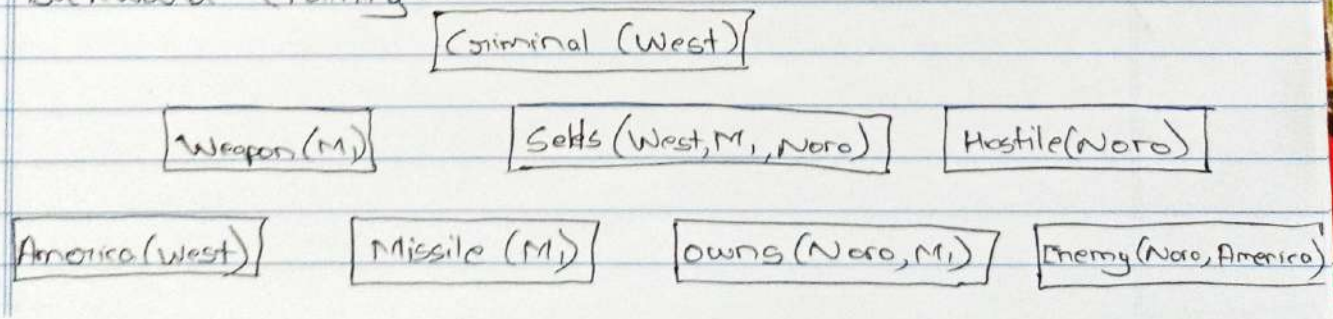
American (x) ^ Weapon (y) ^ Sells (x, y, z) ^ Hostile (z) → Criminal (x)

Owns (Nono, M_1)
Missile (M_1)
Missile (x) ^ Owns (Nono, x) → Sells (West, x, Nono)
Missile (x) → Weapon (x)
Enemy (x, America) → Hostile (x)
American (West)
Enemy (Nono, American)

- Forward Chaining :-



- Backward Chaining :-



THADOMAL SHA
TSEC
ENGINEERING COLLEGE

Q2 Consider the following examples and prove using resolution. Curiosity killed the cat.

i) Everyone who loves all animals is loved by someone.
ii) Anyone who kills an animal is loved by no one.
iii) Jack loves all animals.
iv) Either Jack or curiosity killed the cat named tuna.

Converting to first order logic:

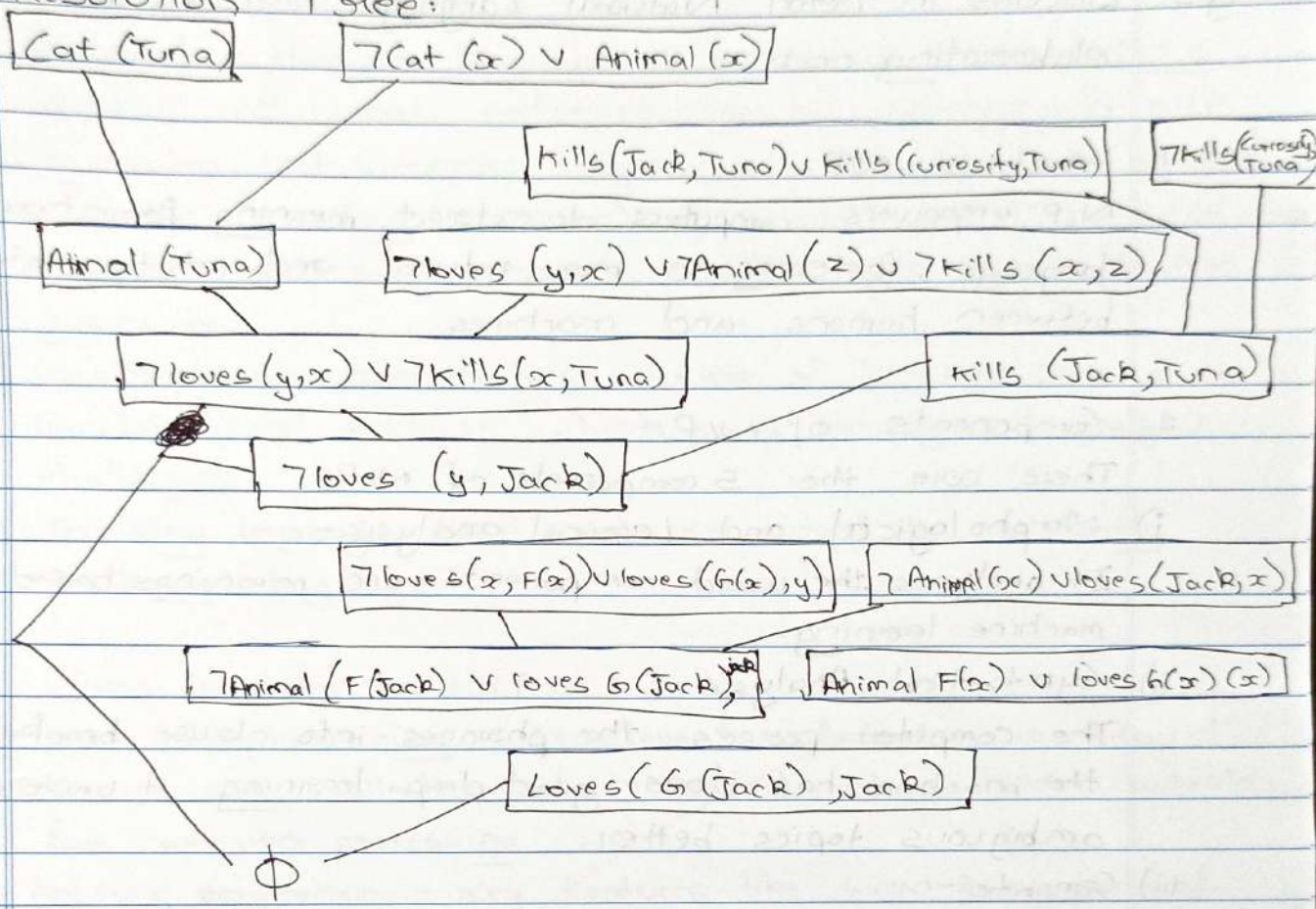
A. $\forall x : \forall y : Animal(y) \rightarrow loves(x,y) \rightarrow \exists\ loves(y,x)$
B. $\forall x : \exists y : Animal(y) \wedge Kills(x,y) \rightarrow \forall z \neg loves(z,x)$
C. $\forall x : Animal(x) \rightarrow loves(Jack,x)$
D. $Kills(Jack, Tuna) \vee Kills(curiosity, Tuna)$
E. $Cat(Tuna)$
F. $\forall x : Cat(x) \rightarrow Animal(x)$
G. $\neg Kills(curiosity, Tuna)$ (neglected goal)

Convert to CNF:

A1. $Animal(F(x)) \vee loves(G(x), x)$ A2. $\neg loves(x, E(x)) \vee loves(G(x), y)$
B $\neg Animal(y) \vee \neg Kills(x,y) \vee \neg loves(z,x)$
C $\neg Animal(x) \vee loves(Jack,x)$
D $Kills(Jack, Tuna) \vee Kills(curiosity, Tuna)$
E $Cat(Tuna)$
F $\neg Cat(x) \vee Animal(x)$
G $\neg Kills(curiosity, Tuna)$

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Resolution Tree:



Tree has reached null state

∴ ¬Kills (curiosity, Tuna) is False

∴ Kills (curiosity, Tuna) is True

Hence Proved.

THADOMAL SH
TSEC
ENGINEERING COLLEGE

Q3 Discuss in detail Natural Language Processing (NLP) elaborating on :-

1. What is NLP :-
NLP empowers computers to extract meaning from human language, fostering a more natural and intuitive interaction between humans and machines.

2. Components of NLP :-
These are the 5 components of NLP :-

i) Morphological and Lexical analysis -
It analyses the word components and meanings based on machine learning.

ii) Syntactical Analysis -
The computer parses the phrases into clause brackets, if the machine has done good deep learning it understands ambiguous topics better.

iii) Semantic -
It looks at the meaning of the sentence, instead of single components. It assigns structure to any phrase.

iv) Discourse Integration -
It makes an effort to look at more that one statement. It is particularly good with anaphora resolution.

v) Pragmatic -
It deals with real world meanings of communication. They can decipher the purpose of phrases and are advanced in programming.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

3. Difficulties in NLP:-

i) Ambiguity - Human language is ambiguous. Understanding sarcasm of human sentences can be challenging for NLP.

ii) Syntactic and Semantic Complexity - The structure (syntax) and the meaning (semantic) of a sentence is crucial for comprehension. NLP requires intricate logic to untangle the grammar.

iii) Unstructured data - A large portion of language data is in unstructured formats which poses parsing and analysis challenges.

iv) Evolving language - NLP needs to adapt to constantly evolving language.

4. Steps involved in NLP:-

a) Text Preprocessing - Irrelevant information is removed, errors are collected and converted into a format suitable for computer processing.

b) Feature engineering - Key features like word forms, parts of speech and relationships between words are extracted.

c) Model Building - ML algorithms are used to train NLP models on labelled data, to perform tasks like classification or machine translation.

d) Evaluation and Refinement - The trained NLP model's performance is meticulously evaluated and model is further refined to evaluate its accuracy and effectiveness.

5. Role of NLP in AI:-

a) Smart Assistants - NLP empowers virtual assistants like siri and alexa to understand and respond to spoken requests.

b) Machine Translation- NLP breaks down language barriers by translating text and speech with increasing accuracy, facilitating communication across cultures.

c) Chatbots- NLP allows chatbots to comprehend user enquires and provide relevant information or customer servire, enhancing user experience and streamlining business operations.

d) Sentiment Analysis- NLP helps analyze the emotions and opinions expressed in text data, offering valuable insights for market research, social media monitoring and brand reputation management.

e) Text summarisation- NLP can automatically summarise large values of text, saving time and effort in information retrieval.

9] What is robotics? Discuss the roles of AI in robotics. Brief the application of robotics in healthcare and agriculture.

→ Robotics is the field of engineering and science that deals with the design, construction, operation and application of robotics. Robots are machines that can sense their environment and take actions based on that information.

Role of AI in robotics-

i) Learning and Adaptability- AI techniques like machine learning enable robots to learn from experience. By analysing

data from sensors and cameras, robots can improve over time.
ii) Computer Vision - AI equips robots with computer vision, allowing them to interpret their surroundings.
iii) Decision making - AI algorithms give robots the ability to make decisions based on sensor data.
iv) Increased efficiency and productivity - AI powered robots can optimise processes, minimise errors leading to significant goals in efficiency.

Robotics in Healthcare:-
Robotics are transforming healthcare by offering
a) Surgical precision : robotic arms aid surgeons in minimal invasive procedures resulting in smaller incisions, faster recovery times and potentially better outcomes.
b) Remote Surgery: Robots can be operated remotely, enabling care for patients in distant locations.
c) Drug delivery and lab automation.

Robotics in agriculture:-
a) Precision planning and seeding : Robots can precisely plant seeds at optimal depths and distancing, maximising yields.
b) Automated Harvesting: Robots can harvest crops more efficiently than human labour, reducing waste and ensuring consistent quality.
c) Livestock management: Robots can automate tasks like milking and feeding livestock, improving farm efficiency and animal well being.

A+ SD 8/4/24