Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ Rishab Mandal

of COMPUTER Department, Semester VI with

Roll No. 2103110 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 27/03/2024

Principal

THADOMAL SHAHANI TSEC ENGINEERING SOLLEGE

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1.	Case Study		25/01/24	
2.	To implement DFS/DLS/DFID algorithm using Python.		08/02/24	
3.	To implement BFS/UCS algorithm		01/02/24	
4.	To implement Genetic algorithm		26/02/24	
5.	To implement A* algorithm		07/03/24	
6.	To study knowledge representation and knowledge base for Wumpus world.		14/03/24	
7.	To solve Blocks world problem using Planning in AI.		21/03/24	
8.	To implement Family tree using prolog.		28/03/24	
9.	Assignment 1		18/01/24	
10.	Assignment 2		04/04/24	

Name: Rishab Mandal
Batch: C23
Roll_No.:2103110

EXPERIMENT NO. 1

CNN-Based Smart Waste Management System

Introduction : -

As urban populations surge, waste management in cities faces significant challenges, with the World Bank estimating an increase from 2.01 billion tons of waste in 2016 to 3.40 billion tons by 2050. Despite the European Union recycling 56% of its waste in 2016, challenges persist, with 24% ending up in landfills. Effective waste management involves multiple steps, from collection to regulation, and collaboration between authorities and users is crucial due to the high cost. However, government efforts often fall short, resulting in either underutilization of resources or environmental pollution.

Challenges arise in waste management when waste collection follows a schedule, leading to underfilled bins or overflow. Recycling initiatives face hurdles due to user ignorance in waste categorization. The advent of the Internet of Things (IoT) provides a transformative solution by integrating intelligence into technologies. The rapid growth of IoT, with 328 million devices connected monthly, underscores its crucial role in addressing waste management challenges. Additionally, machine learning, particularly deep learning, has emerged as a powerful technology, with the market size expected to increase from 1.4 billion dollars in 2016 to 59.8 billion dollars by 2025. Deep learning, exemplified by Convolutional Neural Networks (CNN), excels in image processing and object recognition. Integrating CNN into a smart waste management system holds promise for accurate waste classification, resource conservation, and waste reduction. This paper introduces a smart waste management system utilizing SSD MobileNetV2 for waste detection, integrated with IoT sensors and LoRa-GPS modules for location tracking and long-distance status transfer.

SECTION II. Literature Review/Related Work : -

Urban waste management faces challenges in monitoring and optimizing waste levels in bins. Various authors propose methods utilizing different sensors and communication technologies to address these challenges. Researchers have explored various waste monitoring systems:

Researchers have explored various waste monitoring systems, including cost-effective solutions like infrared sensors and IR wireless systems for real-time garbage status (Navghane et al.). Aasim et al. introduced a GSM-based monitoring system using ultrasonic sensors, alerting authorities when bins are full but with limitations. Efficient telecommunication protocols, incorporating LPWAN technologies like LoRa for extended data transmission, and Bhor et al.'s integration of ultrasonic sensors and GSM in smart bins, enhance waste collection efficiency but lack automated categorization. Norfadzlia et al.'s smart garbage alert system and Kumar et al.'s microcontroller and IoT-based waste alert system address specific aspects, highlighting the need for integrated approaches in urban waste management.

An IoT system with low-power LoRa sensor nodes addresses power consumption concerns in waste management but lacks automatic waste categorization. CNN-based models like WasteNet on Jetson Nano achieve high accuracy in waste classification. However, existing systems often lack comprehensive solutions such as real-time alerts, long-distance monitoring, and automated waste categorization, emphasizing the need for integrated approaches in urban waste management.

SECTION III. Methodology : -

The development of the smart waste management system involves several key components and stages, including the design of the smart bin, the implementation of a CNN-based waste classification and categorization system, and the integration of bin status monitoring with an RFID-based locker system. Each component contributes to creating an efficient and automated waste disposal system.

A. Smart Bin Design

1. Bin Structure and Compartments:

- Design the smart bin structure using acrylic plastic with dimensions of 0.50 m (length) × 0.50 m (width) × 1.20 m (height).

- Incorporate multiple compartments within the bin to facilitate waste sorting and categorization.

2. Electronic Component Compartment:

- Allocate a top compartment for storing electronic components, ensuring protection against external factors.

- Integrate an RFID-based locker system to secure the electronic component compartment.

3. Waste Categorization Compartments:

- Design compartments for specific waste categories (e.g., paper, cardboard, glass, plastic, and metal).

- Implement a system for automatic waste categorization using servo motors controlled by a CNN-based object detection model.

B. CNN-Based Object Detection Model

1. Choice of TensorFlow Lite:

- Opt for TensorFlow Lite over TensorFlow for compatibility with low-power mobile platforms like Raspberry Pi.

2. Object Detection Model Selection:

- Choose the SSD MobileNetV2 Quantized 300×300 model for its real-time detection capabilities and suitability for low-power devices.

3. Dataset Acquisition:

- Obtain the dataset through a combination of free sources and capturing images using a 12-megapixel camera module.

- Resize all images to 300×300 pixels using Batch Image Resize to match the model's requirements.

4. Data Labelling and Augmentation:

- Label waste images using Labelling software, categorizing them into paper, cardboard, glass, plastic, and metal.

- Apply data augmentation techniques (e.g., image shifts, flips, brightness adjustments) to enhance model robustness.

5. Training on Google Colab and Exporting to TensorFlow Lite:

- Utilize Google Colab for training the CNN model, leveraging the superior GPU capabilities for faster convergence.

- Implement hyperparameter tuning with Adam optimizer and cosine decay learning rate.

C. Waste Classification and Categorization System

1. Integration with Hardware:

- Integrate electronic components such as Raspberry Pi, camera module, ultrasonic sensor, and servo motors for waste classification.

- Connect servo motors to a servo driver HAT for effective control.

2. Waste Movement Mechanism:

- Utilize ultrasonic sensors to detect non-detectable waste in the placement area.

- Implement servo motors to move detected waste into the corresponding waste compartments.

D. Bin Status Monitoring and Locker System

1. Monitoring System with Arduino:

- Employ Arduino Uno to monitor bin status using ultrasonic sensors for waste fill levels and GPS module for location tracking.

- Use LoRa communication to transmit real-time information to a remote receiver.

2. RFID-Based Locker Integration:

- Employ Arduino Uno with RFID reader and solenoid locker to protect the electronic component compartment.

- Implement a 30-second RFID authorization window to unlock the locker when an authorized tag is detected.

E. Prototype Testing and Evaluation

1. Smart Bin Prototype Testing:

- Assemble and test the physical prototype for waste classification, bin status monitoring, and locker systems.

2. CNN Model Evaluation:

- Evaluate the CNN model's performance in terms of accuracy, precision, and inference time using a variety of waste objects.

3. System Integration Testing:

- Integrate all components and conduct thorough testing for seamless operation. Analyse results to identify optimization areas and implement iterative development cycles for refinement.

The methodology provides a comprehensive guide for developing and implementing a smart waste management system, incorporating hardware integration, CNN-based waste classification, and advanced monitoring capabilities.

Technology used in the Smart Waste Management System : -

The smart waste management system integrates a combination of hardware components and cutting-edge technologies to automate waste classification, optimize bin status monitoring, and enhance the efficiency of waste disposal. The key technologies employed in the system include:

1. Raspberry Pi:

- Role: Main processing unit for waste classification.

- Functionality: Executes the CNN-based object detection model and controls the movement of waste into respective compartments using servo motors.

- Significance: Enables real-time processing and decision-making for efficient waste sorting.

2. TensorFlow Lite:

- Role: Framework for developing and deploying machine learning models on low-power mobile platforms.

- Functionality: Hosts the SSD MobileNetV2 Quantized 300×300 model for object detection on Raspberry Pi.

- Significance: Facilitates the integration of a lightweight, yet powerful, object detection model suitable for resource-constrained devices.

3. Google Colab:

- Role: Cloud-based platform for machine learning model training.

- Functionality: Utilized for training the CNN-based object detection model with the GPU acceleration provided by Google Colab.

- Significance: Accelerates the training process, enhancing the model's accuracy and efficiency.

4. Arduino Uno:

- Role: Central microcontroller for bin status monitoring and locker system.

- Functionality: Interfaces with ultrasonic sensors, GPS module, LoRa module, RFID reader, and solenoid locker to monitor bin conditions and protect electronic components.

- Significance: Provides a flexible and versatile platform for integrating multiple sensors and communication modules.

5. LoRa (Long Range):

- Role: Wireless communication technology for long-range data transmission.

- Functionality: Facilitates the transmission of real-time bin information (e.g., location, waste fill percentage) to a remote receiver.

- Significance: Enables efficient and long-range communication between the smart bin and the monitoring system.

6. RFID (Radio-Frequency Identification):

- Role: Technology for secure access control to the electronic component compartment.

- Functionality: RFID reader authenticates registered tags to unlock the solenoid locker, providing access to the electronic components.

- Significance: Enhances security and prevents unauthorized access to sensitive electronic components.

7. GPS (Global Positioning System):

- Role: Satellite-based navigation system for accurate location tracking.

- Functionality: Tracks the latitude and longitude of the smart bin for precise location information.

- Significance: Enables real-time monitoring and tracking of the smart bin's geographical position.

8. Servo Motors:

- Role: Mechanical actuators for waste movement within the bin.

- Functionality: Control the movement of plastic boards to categorize waste into designated compartments.

- Significance: Facilitates the automated sorting of waste based on the output from the object detection model.

9. Ultrasonic Sensors:

- Role: Sensors for real-time monitoring of waste fill levels in each compartment.

- Functionality: Measure the distance to detect waste levels and optimize waste collection schedules.

- Significance: Provides accurate and efficient monitoring of waste levels within the smart bin.

10. Acrylic Plastic:

- Role: Material for constructing the physical prototype of the smart bin.

- Functionality: Provides a durable and transparent structure for accommodating electronic components and waste compartments.

- Significance: Ensures the robustness and visual clarity of the smart bin prototype.

The amalgamation of these technologies forms a comprehensive smart waste management system that combines machine learning, wireless communication, and sensor-based monitoring to create an intelligent and automated waste disposal solution.

Conclusion : -

In addressing the challenges of improper use of recycling bins and resource wastage in scheduled waste collection, our proposed system integrates a CNN model, SSD MobileNetV2 Quantized 300×300, and Pi Camera on Raspberry Pi to automate waste classification effectively. Successfully categorizing paper, cardboard, plastic, glass, and metal, the system, equipped with servo motors, swiftly moves waste into designated compartments. The bin status monitoring system, employing ultrasonic sensors and LoRa/GPS shield connected to Arduino Uno, provides accurate waste fill percentage readings and precise latitude and longitude. Utilizing the LoRa module, this information is transmitted to a connected laptop, enabling remote monitoring. While limitations include a small dataset, precision constraints without GPU support, and reliance on batteries, future improvements should focus on dataset expansion, model enhancements, and exploring renewable energy sources for sustained system longevity.

Experiment No. 2

Implementing Depth-First Search (DFS), Depth-Limited Search (DLS), and Depth-First Iterative Deepening (DFID) algorithms in AI involves understanding their basic principles and then translating those into code. Here is a theoretical overview of each algorithm and how you might implement them:

1. **Depth-First Search (DFS):**

- DFS is a fundamental graph traversal algorithm.
- It explores as far as possible along each branch before backtracking.
- It uses a stack (either explicitly or via recursion) to keep track of nodes to visit.
- Pseudocode for DFS:

```plaintext
DFS(G, v):
    mark v as visited
    for each neighbor w of v:
        if w is not visited:
            DFS(G, w)
```

- Implementation steps:
  - Initialize a stack to store nodes to visit.
  - Start with the initial node, mark it as visited, and push it onto the stack.
  - While the stack is not empty:
    - Pop a node from the stack.
    - Mark it as visited.
    - Explore its unvisited neighbors, pushing them onto the stack.

2. **Depth-Limited Search (DLS):**

- DLS is similar to DFS but limits the depth of exploration.
- It's useful for preventing infinite loops in infinite-depth graphs.
- Pseudocode for DLS:

```plaintext
DLS(G, v, limit):
    if limit == 0:
        return
    mark v as visited
    for each neighbor w of v:
        if w is not visited:
            DLS(G, w, limit - 1)
```

- Implementation steps:
  - It's similar to DFS, but with an additional parameter to limit the depth of exploration.
  - Whenever the depth limit is reached, the algorithm backs up to the previous level.

3. **Depth-First Iterative Deepening (DFID):**

- DFID combines the benefits of both DFS and BFS (Breadth-First Search).
- It performs a series of depth-limited searches with increasing depth limits.
- It maintains the advantages of DFS while ensuring complete exploration of the tree.
- Pseudocode for DFID:

```plaintext
DFID(G, start):
    depth = 0
    loop:
        result = DLS(G, start, depth)
```

```
        if result != null:
            return result
        depth = depth + 1
```

- Implementation steps:
  - It repeatedly performs depth-limited searches with increasing depth limits until the goal is found.
  - If the goal is not found within the depth limit, it increases the limit and tries again.

Implementing these algorithms in code involves translating the pseudocode into a programming language of your choice (e.g., Python, Java, etc.). You'll need to represent the graph data structure, handle node visitation, manage the search stack (or recursion), and track the depth level if implementing DLS or DFID. Additionally, you may need to integrate these algorithms into a problem-solving framework depending on the specific application.

Code:

```
def dfs(tree,node,p,goal,closelist,parent):
    parent[node]=p
    print('\nCurrent Node:',node)
    print('ClosedList:',closelist)
    closelist.append(node)

    if node==goal:
        return True;

    for u in tree[node]:
        if u==p:
            continue
        if dfs(tree,u,node,goal,closelist,parent):
```

```
            return True
    return False


n=int(input('Enter the number of node in tree:'))
start=int(input('Enter start node:'))
goal=int(input('Enter goal node:'))

tree=[[] for i in range(n+1)]

print('Enter the edges:')
for i in range(n-1):
    e=input().split()
    u,v=int(e[0]),int(e[1])
    tree[u].append(v)
    tree[v].append(u)

closelist=[]
parent=(n+1)*[-1]
if not dfs(tree,start,-1,goal,closelist,parent):
    print('\nGoal node not found!!!')
else:
    print('\nGoal node found!!!')
    path=[]
    u=goal
    while u!=-1:
        path.append(u)
        u=parent[u]
    print('\nPath: ',end='')
```

```
    path=path[::-1]
    for i in range(len(path)):
        if i==len(path)-1:
            print(path[i])
        else:
            print(path[i],end='->')
```

Output:

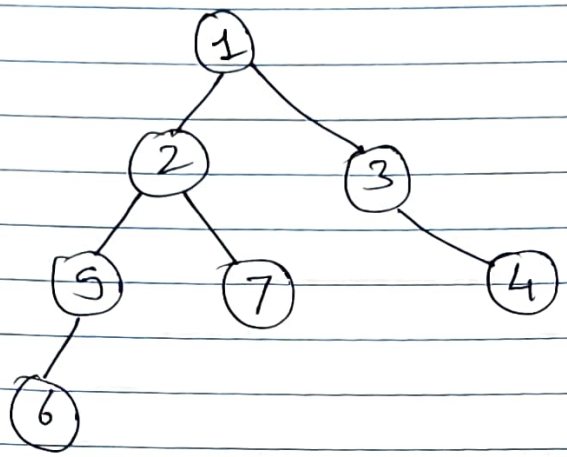
Output

```
Enter the number of node in tree:7
Enter start node:1
Enter goal node:6
Enter the edges:
1 2
1 3
2 4
2 5
3 6
3 7

Current Node: 1
ClosedList: []

Current Node: 2
ClosedList: [1]

Current Node: 4
ClosedList: [1, 2]

Current Node: 5
ClosedList: [1, 2, 4]

Current Node: 3
ClosedList: [1, 2, 4, 5]

Current Node: 6
ClosedList: [1, 2, 4, 5, 3]

Goal node found!!!

Path: 1->3->6

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

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

EXPERIMENT NO. 2

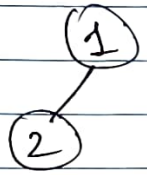
Graph :



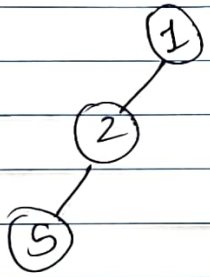
In the above graph, find node (7) using DFS.

Root node

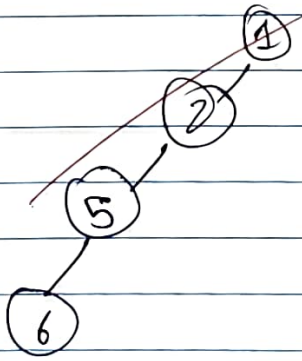
Step - 1



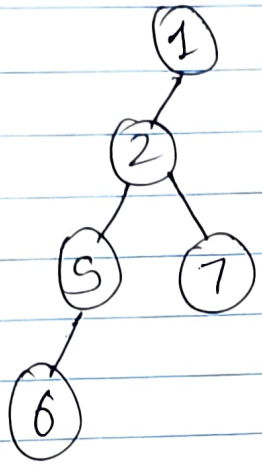
Step - 2



Step - 3



Step - 4



Found node (7)

- Path : (1) → (2) → (7)

AP

SP
2/13/24

Experiment No. 3

1. **Breadth-First Search (BFS):**

Overview:

Breadth-First Search (BFS) is an algorithm used for traversing or searching tree or graph data structures. The key characteristic of BFS is that it explores all the neighbor nodes at the present depth level before moving on to the nodes at the next depth level. This means it systematically expands outward from the starting node and explores all of its neighbors at the current depth before moving deeper into the graph.

Algorithm Steps:

1. **Initialization:**
   - Start by enqueueing the initial or starting node into a queue.
   - Mark the starting node as visited.

2. **Traversal:**
   - While the queue is not empty, repeat the following steps:
     - Dequeue a node from the front of the queue. This node represents the current node being explored.
     - Visit and process the current node.
     - Enqueue all unvisited neighboring nodes of the current node into the queue.
     - Mark each of these neighboring nodes as visited.

3. **Termination:**
   - Terminate the algorithm when the queue becomes empty, indicating that all reachable nodes have been visited.

Key Points:

- BFS is optimal for finding the shortest path in an unweighted graph.

- It guarantees that the shortest path from the starting node to any other node is found first.

- BFS can be used to find the shortest path, level by level, from the starting node to any goal node.

2. **Uniform Cost Search (UCS):**

Overview:

Uniform Cost Search (UCS) is an algorithm used for traversing or searching tree or graph data structures where the edges have associated costs or weights. Unlike BFS, which explores nodes in a breadth-first manner, UCS selects the node with the lowest total cost from the starting node to that node. It ensures that the shortest path to a node is found before moving on to explore other nodes.

Algorithm Steps:

1. **Initialization:**

   - Start by enqueueing the initial or starting node into a priority queue ordered by path cost.

   - Initialize the cost of the starting node as zero.

2. **Traversal:**

   - While the priority queue is not empty, repeat the following steps:

     - Dequeue a node with the lowest path cost from the priority queue. This node represents the current node being explored.

- Visit and process the current node.

- For each unvisited neighboring node of the current node:

  - Calculate the total cost to reach that neighboring node via the current node.

  - Enqueue the neighboring node into the priority queue with its total cost.

  - Update the cost to reach the neighboring node if a lower-cost path is found.

3. **Termination:**

 - Terminate the algorithm when the priority queue becomes empty or when the goal node is reached.

Key Points:

- UCS explores nodes with the lowest total cost first, ensuring that the shortest path to each explored node is found.

- It's optimal for finding the shortest path in weighted graphs.

- UCS can handle graphs with varying edge costs and guarantees the shortest path to each explored node.

Code:

```
from collections import deque


def main():
    n = int(input("Enter the number of nodes:"))
    start = int(input("Enter the source node: "))
    end = int(input("Enter the ending node: "))
    print("Enter the edges of tree:")
    tree = [[] for _ in range(n+1)]
    for _ in range(n-1):
```

```
    u, v = map(int, input().split())
    tree[u].append(v)
    tree[v].append(u)

parent = [-1] * (n+1)
cost = [0] * (n+1)
closelist = []
q = deque()
found = False

q.append(start)
while q:
    currNode = q.popleft()
    print('\nCurrent Node:',currNode)
    print('ClosedList:',closelist)
    closelist.append(currNode)

    if currNode == end:
        found = True
        break

    for neighbor in tree[currNode]:
        if neighbor in closelist:
            continue

        parent[neighbor] = currNode
        cost[neighbor] = cost[currNode] + 1
        q.append(neighbor)
```

```
    if not found:
        print("Goal node not found!!!")
        return

    print("\nGoal node found!!!")
    path = []
    end_copy = end
    while end_copy != -1:
        path.append(end_copy)
        end_copy = parent[end_copy]
    path.reverse()

    print("\nPath Cost:", cost[end])
    print("\nPath:", end=" ")
    for node in range(len(path)):
        print(path[node], end=(" " if node == len(path)-1 else " -> "))
    print()

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

Output:

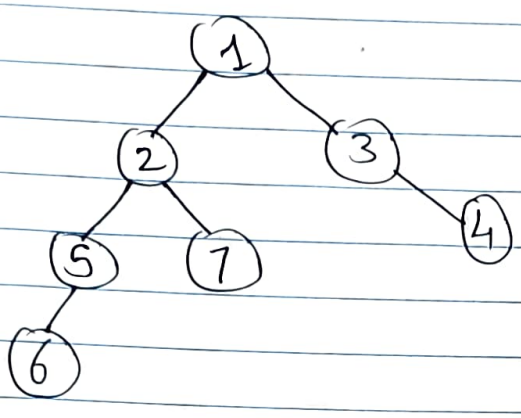
Output

```
Enter the number of nodes:7
Enter the source node: 1
Enter the ending node: 6
Enter the edges of tree:
1 2
1 3
2 4
2 5
3 6
3 7

Current Node: 1
ClosedList: []

Current Node: 2
ClosedList: [1]

Current Node: 3
ClosedList: [1, 2]

Current Node: 4
ClosedList: [1, 2, 3]

Current Node: 5
ClosedList: [1, 2, 3, 4]

Current Node: 6
ClosedList: [1, 2, 3, 4, 5]

Goal node found!!!

Path Cost: 2

Path: 1 -> 3 -> 6

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

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

EXPERIMENT NO. 3

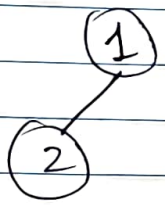
Graph :



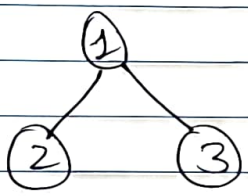
- In the above graph, find node (4) using BFS.

Root node (1)

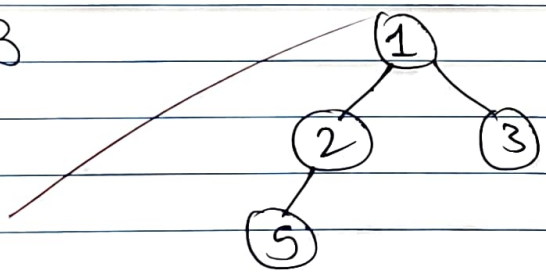
Step - 1



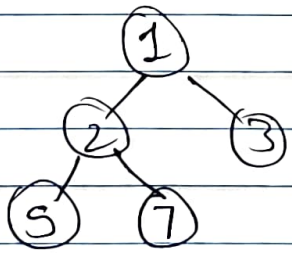
Step - 2



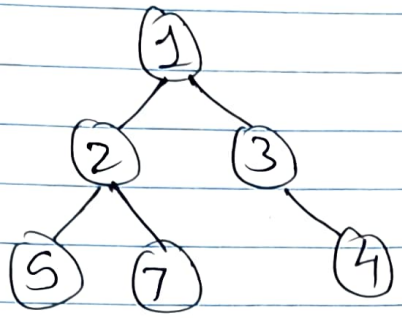
Step - 3



Step - 4



Step-5



Found node (4)

Path : (1) → (3) → (4)

A+

SP
27/3/24

Experiment No. 4

1. **Greedy Search:**

Overview:

Greedy Search is a simple algorithm used for traversing or searching tree or graph data structures. At each step, it selects the node that appears to be the best choice based solely on heuristic information, without considering the overall path to that node. Greedy Search is not guaranteed to find the optimal solution, but it's relatively efficient and can be useful in certain scenarios.

Algorithm Steps:

1. **Initialization:**
   - Start with the initial or starting node.
   - Initialize an empty list to store the path.

2. **Traversal:**
   - While the goal node has not been reached and there are still unexplored nodes:
     - Select the node that appears to be the best choice based on a heuristic evaluation.
     - Move to the selected node and add it to the path.

3. **Termination:**
   - Terminate the algorithm when the goal node is reached or when there are no more nodes to explore.

Key Points:

- Greedy Search makes decisions based solely on local information, without considering the overall path.

- It's not guaranteed to find the optimal solution, but it can be efficient in certain scenarios, especially when the search space is large.

- Greedy Search may get stuck in local optima and fail to find the global optimum.

2. A* Search:

Overview:

A* Search is an informed search algorithm used for traversing or searching tree or graph data structures. It combines the advantages of both uniform cost search (UCS) and greedy search by considering both the cost to reach a node and an estimate of the cost from that node to the goal. A* Search is widely used due to its optimality and efficiency in finding the shortest path.

Algorithm Steps:

1. **Initialization:**

   - Start with the initial or starting node.

   - Initialize an empty priority queue ordered by the sum of the cost to reach a node and the estimated cost from that node to the goal.

   - Initialize the cost to reach the starting node as zero.

2. **Traversal:**

   - While the priority queue is not empty, repeat the following steps:

     - Dequeue a node with the lowest total cost from the priority queue. This node represents the current node being explored.

     - If the dequeued node is the goal node, terminate the algorithm and return the path.

     - Visit and process the current node.

     - For each unvisited neighbouring node of the current node:

       - Calculate the total cost to reach that neighbouring node via the current node.

       - Enqueue the neighbouring node into the priority queue with its total cost.

       - Update the cost to reach the neighbouring node if a lower-cost path is found.

3. **Termination:**

- Terminate the algorithm when the goal node is reached or when the priority queue becomes empty.

Key Points:

- A* Search is optimal and complete when using consistent heuristic functions.

- It considers both the cost to reach a node and an estimate of the cost from that node to the goal, ensuring efficient and effective pathfinding.

- A* Search guarantees the shortest path from the starting node to the goal node when using admissible heuristic functions.

Code:

```
def astar(tree, start, goal, heuristic):
    n = len(tree)

    open_list=[(start,heuristic[start])]
    parent = [-1] * n
    cost = [float('inf')] * n
    cost[start] = 0
    closed_list = []

    step = 1
    found=False

    while open_list:
        open_list=sorted(open_list, key=lambda x: x[1])
        current_node = open_list[0]
        current=current_node[0]
```

```
    print("\nStep:", step, "\nOpen List:",open_list, "\nClosed List:", closed_list)
    step += 1
    del open_list[0]
    closed_list.append(current)

    if current == goal:
        found=True
        break

    for neighbor, edge_cost in tree[current]:
        if neighbor in closed_list:
            continue

        cost[neighbor] = cost[current] + edge_cost
        parent[neighbor] = current
        open_list.append((neighbor,heuristic[neighbor]+cost[neighbor]))

if not found:
    print("\nGoal node not found!!!")
    return None,None

path = []
while goal != -1:
    path.append(goal)
    goal = parent[goal]

path.reverse()
return path, cost[path[-1]]
```

```
n = int(input("Enter the number of nodes: "))

tree = [[] for _ in range(n+1)]
print("Enter the edges of the tree 1-based indexing (u v weight):")

for _ in range(n-1):
    u, v, weight = map(int, input().split())
    tree[u].append((v, weight))
    tree[v].append((u, weight))

start = int(input("Enter the source node: "))
goal = int(input("Enter the goal node: "))
if start>n:
    print('\nInvalid start node!!!')
    exit(0)

heuristic =[float('inf')]+[int(i) for i in input("Enter heuristic for each node:").split()]

path, path_cost = astar(tree, start, goal, heuristic)

if path:
    print("\nA* Path:", path)
    print("A* Path Cost:",path_cost)
```

Output:

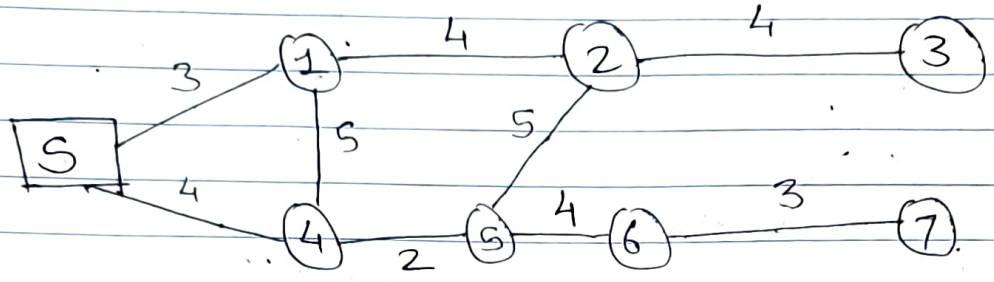
Output

```
Enter the number of nodes: 7
Enter the edges of the tree 1-based indexing (u v weight):
1 2 1
1 3 1
2 4 1
2 5 2
3 6 2
3 7 2
Enter the source node: 1
Enter the goal node: 6
Enter heuristic for each node:5 1 2 2 3 1 4

Step: 1
Open List: [(1, 5)]
Closed List: []

Step: 2
Open List: [(2, 2), (3, 3)]
Closed List: [1]

Step: 3
Open List: [(3, 3), (4, 4), (5, 6)]
Closed List: [1, 2]

Step: 4
Open List: [(4, 4), (6, 4), (5, 6), (7, 7)]
Closed List: [1, 2, 3]

Step: 5
Open List: [(6, 4), (5, 6), (7, 7)]
Closed List: [1, 2, 3, 4]

A* Path: [1, 3, 6]
A* Path Cost: 3

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

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

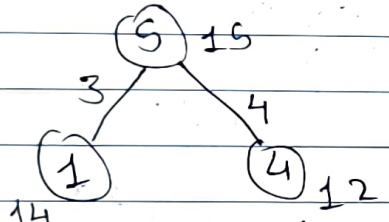
EXPERIMENT NO. 4

A* algorithm



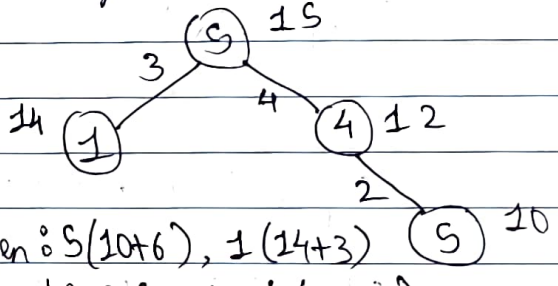
Start state (S), Goal state = 7. Find path using A* search, given h(1) = 14, h(2) = 10, h(3) = 8, h(4) = 12, h(5) = 10, h(6) = 10, h(7) = 15

Step - I



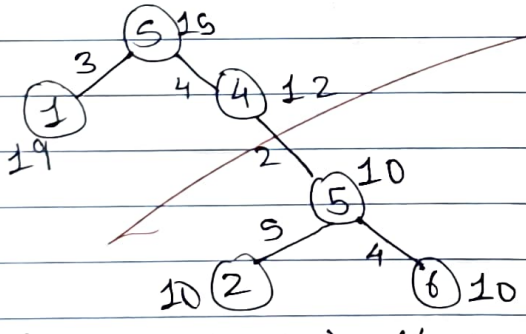
Open : 4 (12+4), 1(14+3)
Closed : S(15)

Step - II



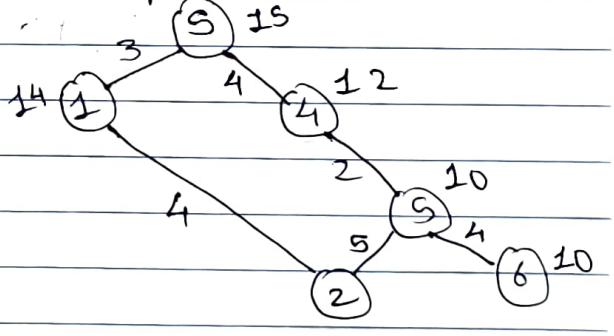
Open : S(10+6), 1(14+3)
Closed : S(15), 4(12+4)

Step - III



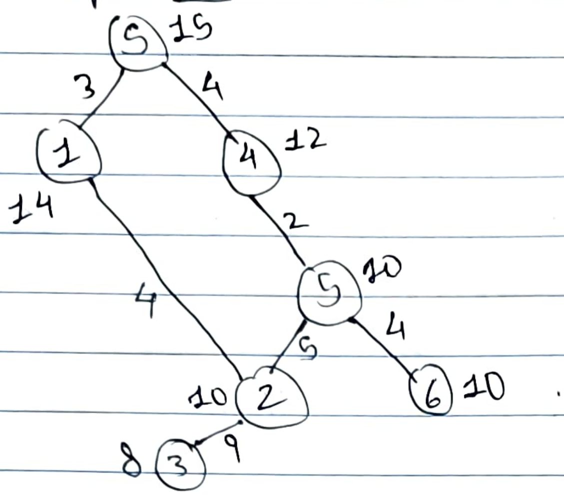
Open : 1(14+3), 6(10+10), 2(10+11)
Closed : S(15), 4(12+4), 5(10+6)

Step - IV



Open : 2(10+7) 6(10+10)
Closed : S(5), .., 1(14+3)

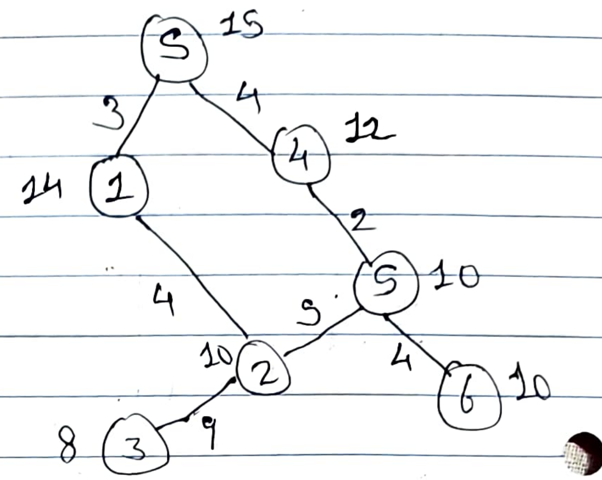
Step - V



Open : 3(8+11), 6(10+10)

Closed : S(15), 4(12+4), 5(10+6),
1(14+3), 2(10+7)

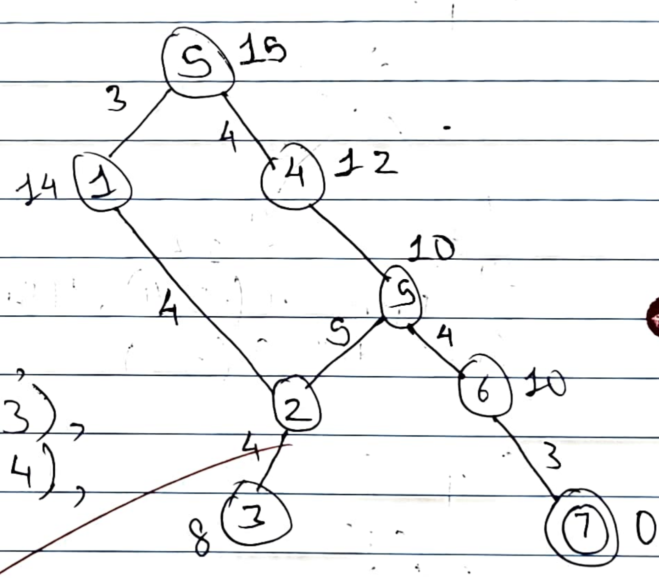
Step - VI



Open : 6 (10+10)

Closed : S(15), 4(12+4), . . ,
2(10+7), 3(8+11)

Step - VII



Open : 7(0+13)

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

AI 27/3/24

Experiment No. 4

1. **Genetic Algorithm (GA):**

Overview:

Genetic Algorithms (GA) are optimization algorithms inspired by the process of natural selection and evolution. They are used to find approximate solutions to optimization and search problems by mimicking the process of natural selection.

Basic Concepts:

- **Representation:** Solutions to the problem are represented as individuals or chromosomes in a population. These individuals are typically encoded as strings, arrays, or other data structures.

- **Fitness Function:** A fitness function is defined to evaluate the quality of each individual in the population. It assigns a numerical value to each individual indicating how well it solves the problem.

- **Selection:** Individuals with higher fitness values have a higher chance of being selected for reproduction in the next generation. Various selection methods like roulette wheel selection, tournament selection, or rank-based selection can be used.

- **Crossover:** During reproduction, pairs of individuals are selected from the population and combined to create offspring. Crossover operators are used to exchange genetic information between parents to generate new solutions.

- **Mutation:** After crossover, mutation operators are applied to introduce small random changes into the offspring's genetic material, adding diversity to the population.

- **Replacement:** The new offspring population replaces the old population using various replacement strategies like generational replacement or steady-state replacement.

Algorithm Steps:

1. **Initialization:** Generate an initial population of individuals randomly.

2. **Evaluation:** Evaluate the fitness of each individual in the population using the fitness function.

3. **Reproduction:** Select individuals from the population based on their fitness to serve as parents. Apply crossover and mutation operators to create offspring.

4. **Replacement:** Replace the old population with the new offspring population.

5. **Termination:** Repeat steps 2-4 until a termination condition is met (e.g., a maximum number of generations or convergence to a satisfactory solution).

Key Points:

- Genetic Algorithms are stochastic optimization techniques that can efficiently search large solution spaces.
- They are particularly useful for complex optimization problems with many local optima.
- Parameters such as population size, crossover rate, mutation rate, and termination criteria need to be carefully tuned for optimal performance.

2. **Hill Climbing Algorithm:**

Overview:

Hill Climbing is a local search algorithm used for optimization problems. It starts with an initial solution and iteratively moves to neighbouring solutions that offer better objective function values, ultimately reaching a local optimum.

Basic Concepts:

- **Current State:** The algorithm maintains the current state, representing the current solution in the search space.

- **Neighbour Generation:** Neighbours of the current state are generated by making small incremental changes to the current solution.

- **Objective Function:** An objective function or fitness function evaluates the quality of a solution. The algorithm seeks to maximize or minimize this function based on the problem's requirements.

- **Local Optimum:** The algorithm terminates when no better neighbours can be found, indicating that the current state is a local optimum.

Algorithm Steps:

1. **Initialization:** Start with an initial solution.

2. **Evaluation:** Evaluate the objective function value of the initial solution.

3. **Neighbour Generation:** Generate neighbouring solutions by making small modifications to the current solution.

4. **Selection:** Select the neighbour with the best objective function value among all neighbouring solutions.

5. **Update:** Move to the selected neighbour and update the current state.

6. **Termination:** Repeat steps 3-5 until no better neighbours can be found or a termination condition is met.

Key Points:

- Hill Climbing is simple and easy to implement, making it suitable for small and medium-sized problems.
- It's prone to getting stuck in local optima, especially in problems with rugged landscapes.
- Variants like stochastic hill climbing, simulated annealing, and genetic algorithms are used to overcome its limitations and improve performance in more complex optimization problems.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 5

Aim :- To implement Genetic algorithm in Python.

Theory :-

Genetic algorithms are based on ideas of natural selection and genetics.

Important functions :

1) Initialization :

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

4> Crossover (Recombination) :

It is the process of combining genetic material from two parent solutions to produce off spring solutions.

5> Mutation :

Mutations typically involves randomly altering some compunents of the chromosome, such as flipping bits or swapping genes.

6> Termination :

Termination criteria 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 :-

Thus, we were able to understand and successfully implement GA in python..

AI 27/3/24

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 6

Aim :- Knowledge representation and knowledge base for Wumpus World.

Theory :-

Q.1] (a) "Some students took French in Spring 2001".

Sol^n $\exists x$: student $(x) \wedge$ takes $(x,$ French, Spring 2001)

(b) "Every student who takes French passes it".

$\forall x$: student $(x) \wedge$ takes $(x,$ French$) \rightarrow$ Passes$(x,$ French$)$

(c) "Only one student took Greek in Spring 2001".

$\exists x$: student $(x) \wedge$ takes $(x,$ Greek, Spring 2001$)$
$\wedge [\forall y : \neg (y = x)] \rightarrow \neg$ takes $(y,$ Greek, Spring 2001$)$

(d) "The best score in Greek is always higher than the best score in French".

$\exists x \forall y$ score $(x,$ Greek$) \wedge$ score $(y,$ French$)$
$\rightarrow$ higher (x, y)

(e) "Every person who buys a policy is smart".

$\forall x \exists y$: person $(x) \wedge$ policy $(y) \wedge$ buys (x, y)
$\rightarrow$ smart (x)

(f) "No person buys an expensive policy"

$\forall x \forall y : person(x) \wedge policy(y) \wedge expensive(y) \longrightarrow \neg buys(x, y)$

(g) "There is an agent who sells policies only to people who are not insured."

$\exists x \exists y \exists z \; agent(x) \wedge policy(y) \wedge people(z) \wedge sells(x, y, z) \wedge \neg insured(z)$

(h) "There is a barber who shaves all men who do not shave themselves"

$\exists x \forall y \; barber(x) \wedge men(y) \wedge \neg shave(y, y) \longrightarrow shave(x, y)$

(i) "A person born in UK; Each of whose parents is a UK citizen or a UK resident; is a UK citizen by birth.

$[[\forall x : Person(x) \wedge Born(x, UK)] \wedge [\forall y \; Parent(y, x) \longrightarrow citizen(y, UK) \vee citizen(y, resident(y, UK))]]$

$\longrightarrow [Birth\ citizen(x, UK)]$

(j) "A person born outside UK, one of whose parents is a UK citizen by birth, is a UK citizen by descent."

$$\forall x \left[\left[\text{Person}(x) \wedge \neg \text{Born}(x, UK) \right] \wedge \left[\exists y : \text{Parent}(y, x) \wedge \text{Birth citizen}(y, UK) \right] \right] \rightarrow \left[\text{Citizen Descent}(x, UK) \right]$$

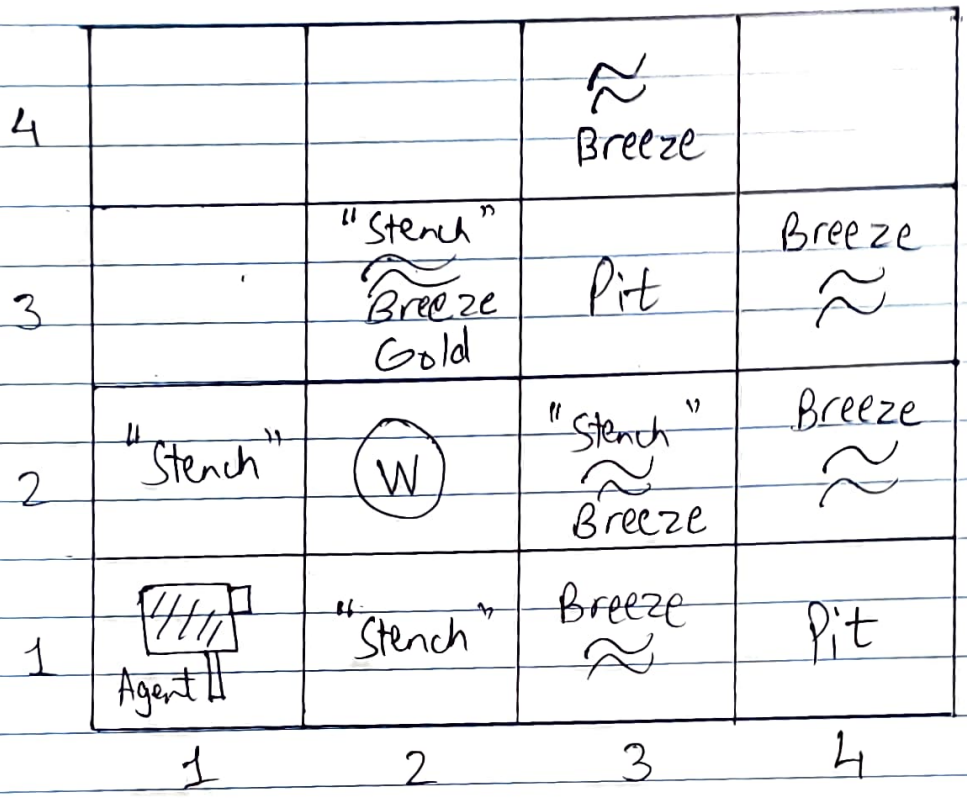
Q. 2 Describe the Wumpus World Problem.

Soln Wumpus world is a cave which has 4/4 rooms connected with passage ways. So there are a total 16 rooms which are connected with each other. We have knowledge-based agent who will go forward in this world. The cave has a room with a beast called Wumpus. It can be shot by an arrow, but the agent has a single arrow. There is one room with gold.

Its knowledge base comprises of collection of proposition variables and rules, and the logical relationship between them.

Wumpus World

[A] Initial State



Wumpus = {[2,2]}
Gold = {[2,3]}
Agent = [1,1]
State = {None, None, None, None}

KB
[1,1] = Ok
[2,1] = Ok
[1,2] = Ok
[1,3] = Ok

Step 1 : Agent moves to [2,1]

State : {Stench, None, None, None}

$B_{1,1}$	$B_{2,1}$	$B_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	false	false	false	false	false	false	true	true	true	true	false	false
false	false	false	false	false	false	true	true	true	false	true	false	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	true	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	false	true	false	true	true	true	true	true	true
false	true	false	false	false	true	true	true	true	true	true	true	true
false	true	false	false	true	false	false	true	false	false	true	true	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	true	true	true	true	true	true	false	true	true	false	true	false

Step 2] Agent moves to [1,2]

State : { Stench, None, None, None }

Step 3] Agent decides to move to [1,3]

State : { None, None, None, None }

Step 4] Agent moves to [1,4]

State : { None, None, Bump, None }

Step 5] Agent is in [2,3]

State : { Stench, Breeze, None, Glitter }

Since it perceives glitter, agent gets gold and backtracks from all 'ok' states to reach start.

AT 27/3/24

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 7

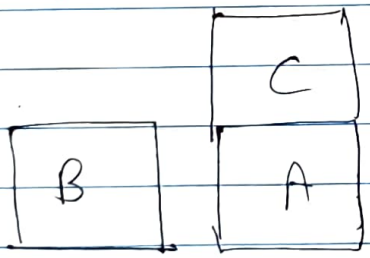
Aim :- To solve blocks world problem using Planning in AI.

Theory :-

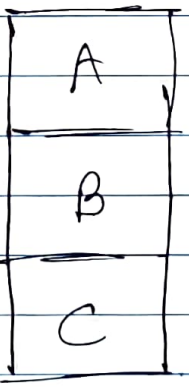
Blocks World Problem :

The blocks world problem domain consists of a set of cube-shaped blocks sitting on a table. The blocks can be stacked, but only one block can fit directly on top of another. A robot arm can pick up a block only one block at a time. The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks.

INITIAL STATE :



GOAL STATE :



Predicates used :

- On (b, x) is used to indicate that block b is on x, where x is either another block on the table.
- Move (b, x, y) is used to indicate movement of block b from the top of x to the top of y.
- Clear (x) It is used to indicate that nothing is on top of x.

Action (Move (b, x, y)),

PRECOND : On (b, x) ∧ Clear (b) ∧ Clear (y)
EFFECT : On (b, y) ∧ Clear (x) ∧ ¬ On (b, x) ∧ ¬ clear (y))

For next step,

Action (MoveToTable (b, x)),

PRECOND : On (b, x) ∧ Clear (x)
EFFECT : On (b, Table) ∧ Clear (x) ∧ ¬ On (b, x))

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

PLANNING :

Sequence Solution :

[Move To Table (C, A), Move (B, Table, C),
Move (A, Table, B)]

Init (On (A, Table) ∧ On (B, Table) ∧ ....
Clear (B) ∧ Clear (C))

Goal (On (A, B) ∧ On (B, C))

Action (Move (b, x, y),

PRECOND : On (b, x) ∧ Clear (b) ∧ Clear (y) ∧
Block (b) ∧ Block (y) ∧ (b ≠ x) ∧ (b ≠ y)
∧ (x ≠ y),

EFFECT : On (b, Table) ∧ Clear (x) ∧ ¬ On (b, x))

Conclusion :

Thus, we understood planning in AI and was able to solve the blocks world problem by using appropriate planning techniques.

Experiment No. 8

Aim : - Implementing Family tree using prolog.

Prolog Code:

```
male(jack).
male(oliver).
male(ali).
male(james).
male(simon).
male(harry).
female(helen).
female(sophie).
female(jess).
female(lily).
spouse(jack, helen).
spouse(helen, jack).
spouse(oliver, sophie).
spouse(sophie, oliver).
spouse(ali, jess).
spouse(jess, ali).
spouse(james, lily).
spouse(lily, james).
father(jack, jess).
```

```
father(jack, lily).
father(oliver, james).
father(simon, james).
father(jess, simon).
father(ali, simon).
father(lily, harry).
father(james, harry).
mother(helen, jess).
mother(helen, lily).
mother(sophie, james).
mother(jess, simon).
mother(lily, harry).
/* Rules */
grandfather(X,Y) :- father(X,Z), father(Z,Y).
grandfather(X,Y) :- father(X,Z), mother(Z,Y).
grandmother(X,Y) :- mother(X,Z), father(Z,Y).
grandmother(X,Y) :- mother(X,Z), mother(Z,Y).
brother(X,Y) :- male(Y), not(X=Y), not(spouse(X,Y)), father(X,Z),
father(Y,Z), mother(X,W), mother(Y,W).
sister(X,Y) :- female(Y), not(X=Y), not(spouse(X,Y)), father(X,Z),
father(Y,Z), mother(X,W), mother(Y,W).
uncle(X,Y) :- male(Y), father(X,Z), brother(Z,Y).
uncle(X,Y) :- male(Y), mother(X,Z), brother(Z,Y).
aunt(X,Y) :- female(Y), father(X,Z), sister(Z,Y).
aunt(X,Y) :- female(Y), mother(X,Z), sister(Z,Y).
```

```
aunt(X,Y) :- uncle(X,Z), spouse(Z,Y).
sibling(X, Y) :-
father(F, X), father(F, Y), mother(M, X), mother(M, Y), X \= Y.
calculate(X, '+', Y, Result) :-
Result is X + Y.
calculate(X, '-', Y, Result) :-
Result is X - Y.
calculate(X, '*', Y, Result) :-
Result is X * Y.
calculate(X, '/', Y, Result) :-
Y =\= 0,
Result is X / Y.
calculate(_, '/', 0, 'Cannot divide by zero').
calculate(_, Operator, _, 'Invalid operator'):-
\+ member(Operator, ['+', '-', '*', '/']).
```

OUTPUT QUERIES : -

***male*(jack)**

1

True

***grandmother*(helen,simon)**

1

True

***grandmother*(sophie,_)**

1

True

***grandmother*(sophie,_var)**

_var = harry

***sibling*(jess, _x)**

_x = lily

***sibling*(jack, _x)**

False

***grandfather*(oliver,harry)**

1

True

***grandfather*(ali,harry)**

False

_x is 5 + 3.

_x = 8

***calculate*(5, '+', 3, Result).**

Result = 8

***calculate*(5, '*', 3, Result).**

Result = 15

***calculate*(5, '/', 3, Result).**

Result = 1.6666666666666667

Family Tree : -

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

ASSIGNMENT NO. 1

Q.1] Give one definition on AI for each of the following approaches :-

i) Acting Humanly :

"The art of creating machines that perform functions that require intelligence when performed by people".

ii) Thinking Humanly :

"The existing new effort to make computers think .... machines with minds, in the full and literal sense."

iii) Acting Rationally :

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

iv) Thinking Rationally :

"The study of mental facilities through the use of computational models."

Q.2] Explain the components of the AI system in detail.

i) Learning :

Just like human beings, the first step in the development process with regard to AI is the learning stage. The learning process involves the memorization of individual items including different solutions to problems, vocabulary, and foreign languages, among others. Simplest form of learning is by trial and error. Through this learning process, programs that utilize AI are able to keep notes of all actions or moves that led to the positive results, allowing program to solve problems arising in the future.

ii) Reasoning :

Reasoning is basically logic or to be able to generate judgements from the given set of facts. It is carried out on the basis of strict rules of validity to perform specified task. Reasoning can be of 2 types, deductive or inductive. In programming logic, deductive inferences are generally used.

iii) Problem Solving :

AI addresses a huge variety of problems. For example, finding out winning moves on the board games, planning actions in order to achieve the defined task, identifying various objects from given images, etc. Problem solving methods are divided into 2 types, special purpose and general purpose methods.

iv) Perception :

In keeping with the comparisons to the function of the human-mind, the way in which individual perceive the world around them is critical to the manner in which they solve problems in their respective minds.

v) Language understanding :

The final component that makes up development of AI is language understanding. It can be defined as a set of different systems signs that justify their various means or methods using the convention.

Q.3] Write a short note on categorization of AI.

i) Based on capabilities :-

(a) Weak AI :

It is a type of AI which is able to perform a dedicated task with intelligence. It cannot perform beyond its field or limitations, as it is only trained for 1 specified task. For example voice assistant.

(b) General AI :

It is a type of Intellisence which could perform any intellectual task with efficiency like a human. The idea behind a general AI is to make a system that can learn, perceive, understand and function entirely like human, with connections and logic.

(c) 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.

ii) Based on Functionality :

(a) Reactive Machines :

These are most basic types of machines that do not store memories or past experiences for future actions.

These machines only focus on current scenarios and react on it as per the circumstance.

(b) Limited Memory :

These machines can store past experiences or some data for a short period of time.

(c) Theory of Mind Machines :

Theory of Mind AI should understand the human emotions, people, beliefs and be able to react socially like humans. They are yet to be developed.

(d) Self-Awareness AI :

It is the future of AI. These machines will be super intelligent and will have their own consciousness, sentiments and self-awareness. They will be smarter than human mind.

Q.4] Explain problem formulation with the help of eg.

Problem formulation is a crucial step in process of solving problems, it involves defining the problem in a way that allows an intelligent agent to understand it and find a solution.

i) State Formulation :
ii) Initial state
iii) Goal Test
iv) Action Sequence
v) Path Cost

It is associated with each action and represents cost or effort required to perform that action.

Q.5] Explain PEAS properties in detail.

i) Performance Measure : It is a criteria used to evaluate performance, i.e success / quanification.

ii) Environment : It is external context, in which environment acts as a medium for agent operations.

iii) Actuators : It is mechanism or component through which agent interacts with environment and perform actions.

iv) Sensors : These are input devices that allow agent to perceive and gather information about it.

PEAS properties help us in designing efficient intelligent agents for any given task / tasks.

Example agent: Waste Sorting AI agent.

Task: To monitor and sort different waste materials using vision based techniques into various categories like recyclable, non-recyclable and compostable-materials.

PEAS Properties:

1) Performance Measure:
- Accurate classification of waste products in a short span of time.
- Min contamination of inappropriate waste materials into wrong categories by implementing adaptive learning.
- Faster identification, using advanced image analysis algorithms.

2) Environment:
- The agent shall operate in a waste sorting facility, being exposed to multiple kinds of waste product.

3) Actuators:
- Robotic arms for faster seggregation of items.
- Wavelength separator for identifying compostable items.

4) Sensors:
- High resolution cameras for capturing clear images of waste products.
- Weight sensing device pre-attached to conveyor belt for measuring weight of items.

Q.6] Describe different types of environment with suitable examples.

Soln i) Fully vs Partially Observable :
If an agent sensor can sense or access complete state of an environment at each point of time then it is a fully observable else partially observable.

ii) Deterministic vs Stochastic :
If an agent's current state and selected action can be determined by next state of environment, then it is deterministic else stochastic.

iii) In an episodic vs sequential rather episodic environment, there is a series of one-shot actions, and only the current percept is required for action, which is not in sequential one.

iv) Single agent vs Multi-agent :
If only one agent is involved in environment, and operating by itself then it is a single agent, else multi-agent.

v) Static vs Dynamic :
If environment can change itself while an agent is deliberating, then it is called a dynamic environment, else it is called a static environment.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

ASSIGNMENT NO. 2

Q.1 Discuss the Forward chaining and Backward chaining algorithms. Illustrate the working of forward chaining and backward chaining for the following problem..

→ "The law says that it is crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American."

Inference Engine

It is the component of the intelligent system in AI, which applies logical rules to the knowledge base to infer new information from known facts.

Forward Chaining:

It is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which starts with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more

data until a goal is reached.

The forward chaining algo, starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until problem is solved.

Backward Chaining:

Backward Chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Using the example given in question;

Prove: Colonel is a Criminal.

Step 1: Convert facts to FOL

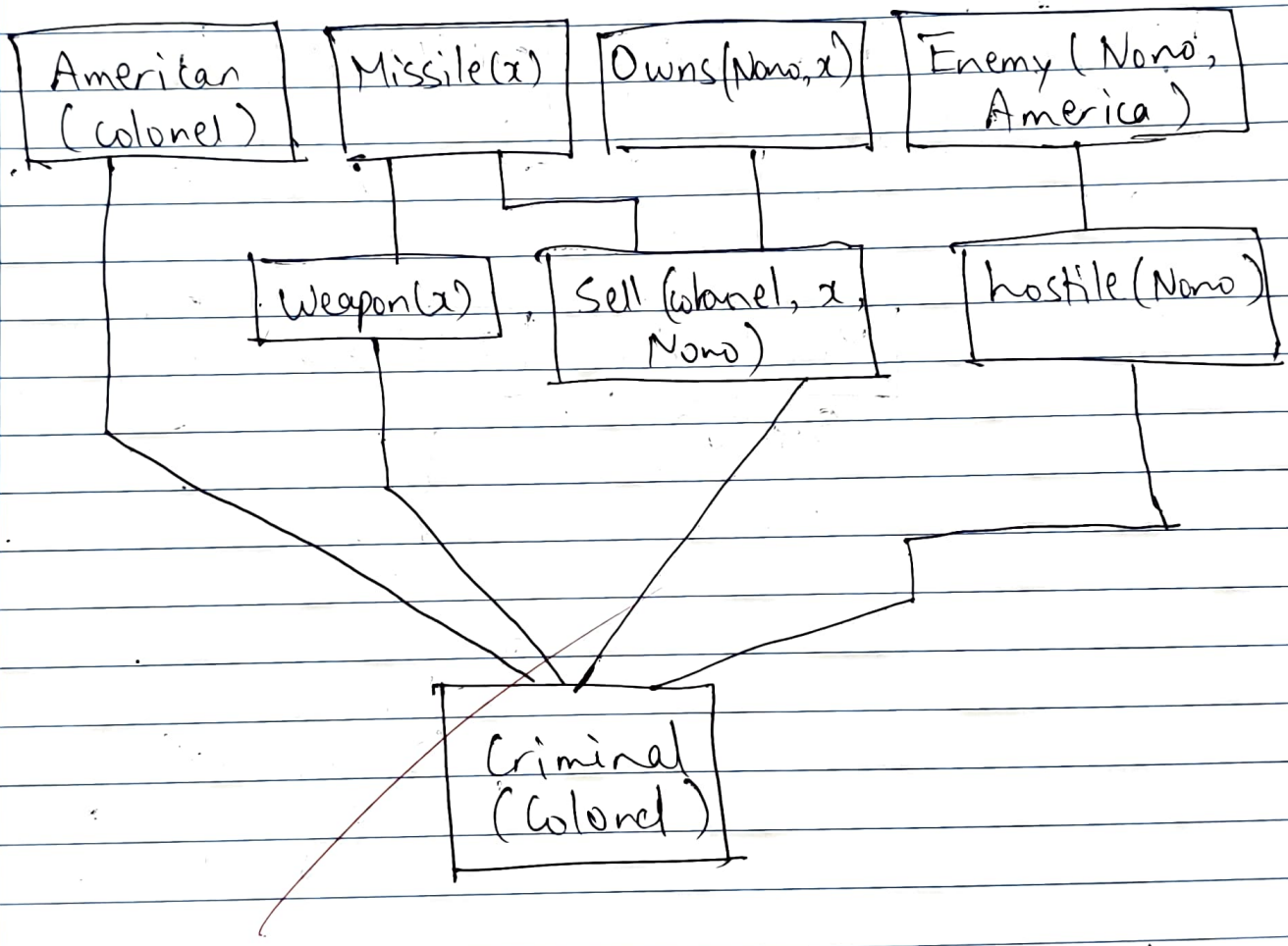
1) American $(x) \wedge$ weapon$(y) \wedge$ hostile$(z) \wedge$ sells$(x, y, z) \longrightarrow$ criminal(x)

2) Enemy (Nono, America)

3) Owns(Nono, x)
4) Missile(x)
5) $\forall x$: Missile(x) $\wedge$ owns(Nono, x) $\rightarrow$ sell(Colonel, x, Nono)
6) Missile(x) $\rightarrow$ weapon(x)

Proof by Forward Chaining :

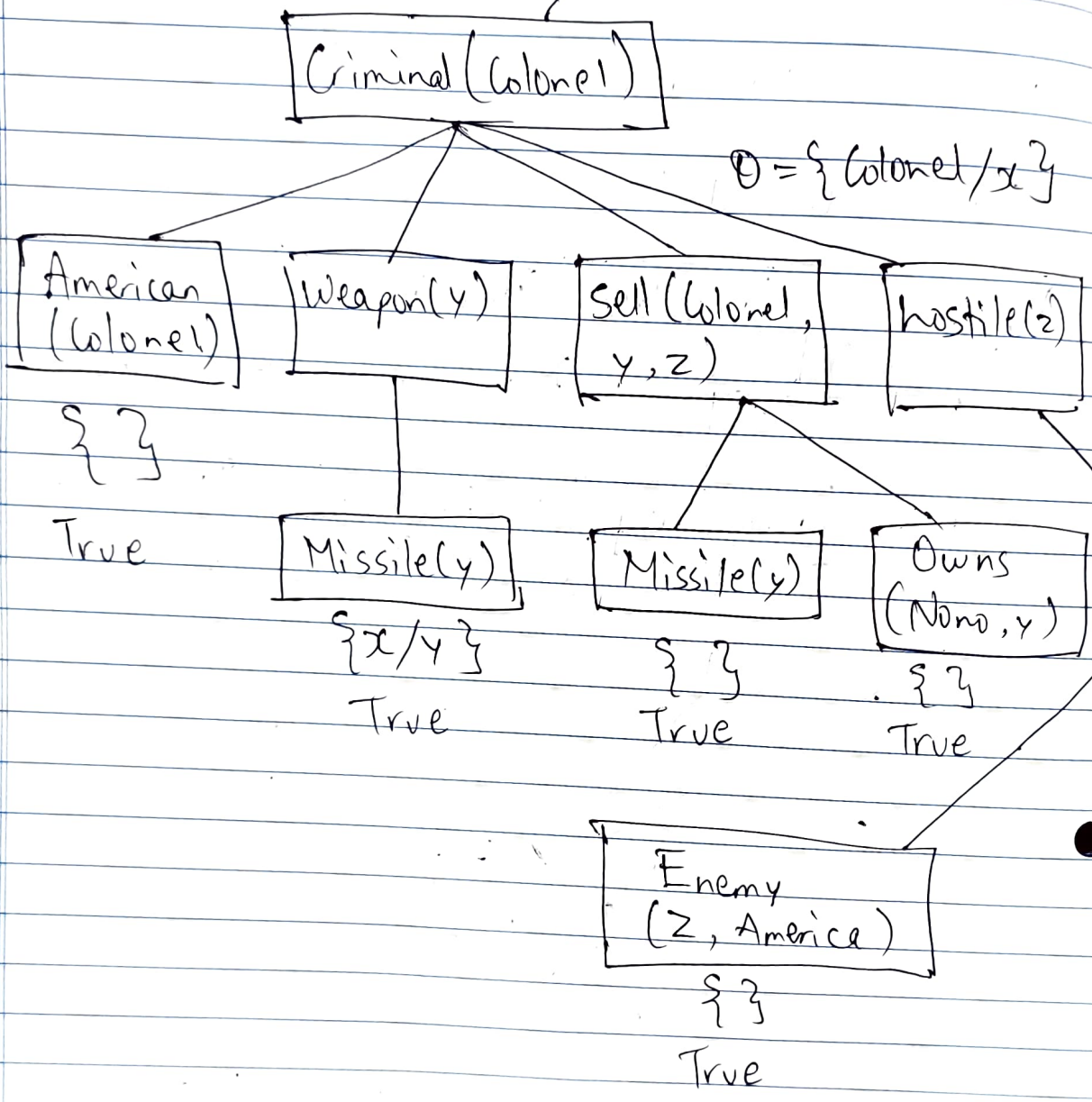
Start with all known Facts



Therefore, it is proved that "Colonel is a criminal".

Proof by Backward chaining

Start with goal (Goal)



Since, all the statements are proved true, using backward chaining, therefore "Colonel is a criminal".

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q.2 Consider following example and prove using resolution "Curiosity killed cat".

→ 1) Everyone who loves all animals is loved by someone.

$\forall x$ people(x) $[\forall y$ Animal$(y) \rightarrow$ loves$(x,y)]$
$\rightarrow \exists z$ people$(z) \wedge$ loves(z,x)

2) FOL: $\forall x [\exists z$ Animal$(z) \wedge$ kills$(x,z)]$
$\rightarrow \forall y \neg$ loves(y,x).

3) $\forall x$ Animal$(x) \rightarrow$ loves$(jack, x)$

4) kill$(jack, Tuna) \vee$ kill$(curiosity, tuna)$

5) Cat$(Tuna)$ } Added rules

6) $\forall x$ cat$(x) \rightarrow$ Animal(x) } Added rules

To prove : kills$(curiosity, tuna)$
By contradiction, $\neg$ kills$(curiosity, tuna)$

Step 1 : Convert to CNF

1) $\forall x$ people(x) $[[\neg \forall y \neg$ Animal$(y) \vee$ loves$(x,y)] \vee$
$[\exists z$ loves$(z,x)]]$
$\rightarrow \forall x [[\exists y$ Animal$(y) \wedge \neg$ loves$(x,y)] \vee$
$[\exists z$ loves$(z,x)]]$

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

• Skolemization :

$\forall x [[Animal(f(x)) \wedge \neg loves(x, f(x))] \vee loves(g(x), x)]$

1) Drop universal quantifiers :

$[Animal(f(x)) \wedge \neg loves(x, f(x)] \vee loves(g(x), x)]$

2) $\neg loves(y, z) \vee \neg Animal(z) \vee \neg kills(x, z)$

3) $\neg Animal(x) \vee loves(jack, x)$

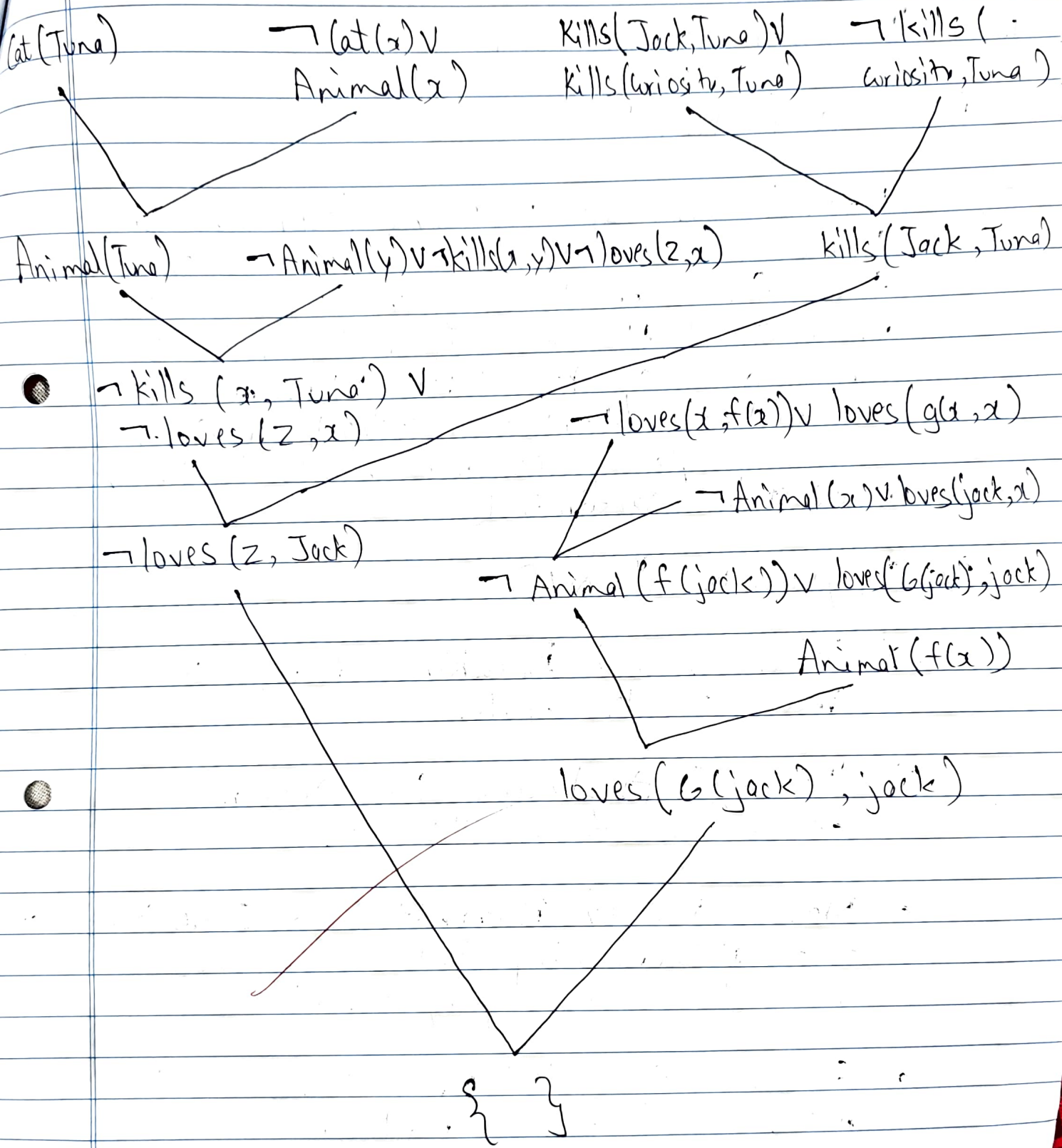
4) $kills(jack, Tuna) \vee kills(curiosity, tuna)$

5) $cat(tuna)$

6) $\neg cat(x) \vee Animal(x)$

7) $\neg kills(curiosity, tuna)$

Resolution graph :—


Cat(Tuna)
¬Cat(x) ∨ Animal(x)
Kills(Jack, Tuna) ∨ Kills(curiosity, Tuna)
¬Kills(curiosity, Tuna)
Animal(Tuna)
¬Animal(y) ∨ ¬Kills(x, y) ∨ ¬Loves(z, x)
Kills(Jack, Tuna)
¬Kills(x, Tuna) ∨ ¬Loves(z, x)
¬Loves(x, f(x)) ∨ Loves(g(x), x)
¬Animal(x) ∨ Loves(jack, x)
¬Loves(z, Jack)
¬Animal(f(jack)) ∨ Loves(g(jack), jack)
Animal(f(x))
Loves(g(jack), jack)
{ }

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q.3 Discuss in detail NLP.

1) What is NLP?

→ Natural Language Processing (NLP) is a branch of AI within computer science that forces on helping computers to understand the way that humans write and speak. With NLP, organizations can analyze text and extract information about people, places and events to better understand sentiments.

2) Components of NLP -

(a) Tokenization : Breaking down text into smaller units.

(b) Parsing : Analyzing the grammatical structure of sentences.

(c) Named Entity Recognition : Identifying and categorizing entities mentioned in text, such as people, places, etc.

(d) Semantic Analysis : Understanding the meaning of words within the given context.

(e) Language generation: Creating human like text on given prompt/input.

3) Difficulties in NLP:

→ They arise due to complexities and ambiguity inherent in human language. Challenges include understanding context, dealing with slang, handling variations in syntax and grammar, resolving ambiguity, and accurately capturing nuances such as sarcasm or irony.

4) Steps involved in NLP:

(a) Preprocessing
(b) Tokenization
(c) Parsing and syntax analysis
(d) Semantic analysis
(e) Feature extraction
(f) Model training
(g) Evaluation and testing
(h) Deployment

5) Role of NLP in AI:

NLP plays a crucial role by bridging gap

between human communication and machine understanding. It enables various apps and facilitates more effective human-computer interaction.

Q.4 What is Robotics? Discuss the role of AI in robotics. Brief up the application of robotics in health care and agriculture.

→ Robotics is a multidisciplinary field that involves the design, construction, operation, and use of robots. Robots are programmable machines capable of carrying out tasks autonomously often with varying degrees of human interaction.

Role of AI in Robotics:

The role of AI is significant, as it enables robots to perceive, learn, reason and act intelligently in dynamic environments.

Healthcare and Agriculture:

Following are some of the applications: Surgery, Rehabilitation, Livestock Monitoring, Harvesting and Sorting.