Savitribai Phule Pune University
(Formerly University of Pune)

T.Y.B.Sc.(Computer Science)

With

Major: Computer Science

(Faculty of Science and Technology)

(For Colleges Affiliated to Savitribai Phule Pune University)

Choice Based Credit System (CBCS) Syllabus Under National Education Policy (NEP)

To be implemented from Academic Year 2026-2027

Title of the Course: B.Sc.(Computer Science)

Foundation of Artificial Intelligence(AI) and Machine Learning(ML)

CS-321-VSC-P

T.Y.B.Sc.(Computer Science)
Semester-V

Work Book

Name:_____

College Name: _____

Roll No.: _____ Division: _____

Academic Year: _____

BOARD OF STUDIES

1. Dr. Patil Ranjeet
2. Dr. Joshi vinayak
3. Dr. Wani Vilas
4. Dr. Mulay Prashant
5. Dr. Sardesai Anjali
6. Dr. Shelar Madhukar
7. Dr. Bharambe Manisha
8. Dr. Deshpande Madhuri
9. Dr. Kardile Vilas
10. Dr. Korpal Ritambhara
11. Dr. Gangurde Rajendra
12. Dr. Manza Ramesh
13. Dr. Bachav Archana
14. Dr. Bhat Shridhar

Co-ordinators

- **Dr. Prashant Mulay**, Annasaheb Magar College, Hadapsar , Pune.
  Member, BOS Computer Science, Savitribai Phule Pune University

Editor

Dr. Prashant Mulay, Annasaheb Magar College, Hadapsar , Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

Prepared by:

Ms. Gadekar Manisha Jankiram	Annasaheb Magar College, Hadapsar, Pune.

Introduction

About the work book:

This LAB Book/Workbook is intended to be used by T.Y.B.Sc.(Computer Science) students for the CS-321-VSC-P Foundation of Artificial Intelligence(AI) and Machine Learning Assignments in Semester–V. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus. The lab book is to be used as a hands-on resource, reference and record of assignment submission and completion by the student. The Lab Book contains the set of assignments which the student must complete as a part of this course.

The objectives of this LAB-Book are:

Defining the scope of the course.

- To bring uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
- To have continuous assessment of the course and students.
- Providing ready reference for the students during practical implementation.
- Provide more options to students so that they can have good practice beforefacing the examination.
- Catering to the demand of slow and fast learners and accordingly providing the practice assignments
  to them.

Instructions to the students

- Students are expected to carry this book every time they come to the lab for computer sciencepractical.
- Students should prepare oneself beforehand for the Assignment byreading the relevant material.
- Instructor will specify which problems to solve in the lab during the allotted slot
  and student should complete them and get verified by the instructor. However, student should spend additional hours in Lab and at home to coveras many problems as possible given in this work book.

Submission:

Problem Solving Assignments:

- The problem solving assignments are to be submitted by the student in the form of a journal containing individual assignment sheets.
- Each assignment includes the Assignment Title, Problem statement, Date of submission, Assessment date, Assessment grade and instructors sign.

Programming Assignments:

- Programs should be done individually by the student in the respective login.
- The codes should be uploaded on either the local server, Moodle, Github or any open source LMS. Print-outs of the programs and output may be taken but not mandatory for assessment.

Assessment:

- Continuous assessment of laboratory work is to be done based on overall performance and lab assignments performance of student.
- Each lab assignment assessment will be assigned grade/marks based on parameters with appropriate weightage.
- Suggested parameters for overall assessment as well as each lab assignment assessment include- timely completion, performance, innovation, efficient codes and good programming practices.

Operating Environment:

For 'Computer Network': Operating system: Linux / Windows

Editor: Any linux based editor like vi, edit etc.

Language: Python (3.x) (Programming Assignment Implementation)

Jupyter Notebook / Google Colab

Libraries: NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn.

Students will be assessed for each exercise on a scale from 0 to 5.

Not done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

Instruction to the Instructors

- Explain the assignment and related concepts in around ten minutes using whiteboard if required or by demonstrating the software.
- You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion page of the respective Lab course.

INDEX

Assignment Completion Sheet

CS-321-VSC-P - Foundation of Artificial Intelligence(AI) and Machine Learning			
Sr. No.	Assignment Name	Marks (Out of 5)	Instructor Sign
1	Problem Solving Using Search		
2	Informed Search Techniques		
3	Constraint Satisfaction Problems (CSP)		
4	Game Playing Algorithms for Decision Making		
5	Knowledge Representation and Logical Reasoning		
6	Basics of Machine Learning		
7	Support Vector Machine (SVM)		
8	Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN)		
9	Case Studies		
Total Marks			

This is to certify that **Mr./Ms**._____

have successfully completed and submitted the coursework for lab course **CS-321-VSC-P_Foundation of Artificial Intelligence(AI) and Machine Learning,**

T.Y.B.Sc.(C.S.)– **Sem. V** prescribed by Savitribai Phule Pune University during Academic year-_____and has scored mark_____out of 15 Marks.

Signature of Batch In-charge Head of Department

Internal Examiner External Examiner

Assignment 1 : Problem Solving Using Search (Total slots = 1)

Objective

- To comprehend the idea of problem solving through the use of search in Artificial Intelligence.
- To familiarize oneself with state space representations for AI problems.
- To examine various search methods that lead to problem solutions.

Ready

- AI-based problem-solving techniques include the search among a number of states to obtain an optimal goal state.
- Search algorithms, it is easier to discover the most effective route to a solution.
- Concepts related to search based problem solving include: initial state, goal state, operators and path costs.

Ready Reference

Problem Solving Using Search

The application of problem-solving techniques is among the most common applications of AI. In problem-solving, an intelligent agent uses the technique of searching to select a sequence of moves from a series of options that will take it from the starting state to the goal state.

Definition

Problem solving in AI using search is a strategy where an AI system examines possible states and actions in order to determine a path from an initial state to a goal state.

Components of a Search Problem

- Initial State : Starting position of the problem.
- Goal State : State the problem is expected to reach.
- State Space : Set of all states that can be reached from the initial state.
- Operators/Actions : Activities that change the current state into another state.
- Path Cost : Cost incurred while changing one state into another.

Steps in Problem Solving

- Define the problem.
- Determine the initial state.
- Define the goal state.
- Generate potential actions.
- Find the solution path.
- Implement the chosen solution.

Features of Search Problems

- Distinct states.
- Definite actions.
- Definable goal state.
- Successor state generation capability.

Advantages

- The process of problem-solving with search is important for making sure that AI solves issues effectively.
- It allows AI to decide on a course of action based on exploring alternative possibilities.
- Search processes are used in many real-world scenarios.
- They aid in finding the best or close to the best solution.
- Searches provide a systematic way of solving a problem.

Disadvantages

- Memory can be a critical constraint when it comes to storing states.
- Time may be increased greatly when trying to find a solution within big state space.
- In some search strategies, a lot of unnecessary states may be traversed before arriving at the destination.
- Complex situations may lead to a combinatorial explosion of the number of states.
- Optimal solutions are not always guaranteed through some search strategies**.**

Application

- Search algorithms are employed in route-finding and GPS systems.
- Search algorithms are employed in robotics in terms of planning a motion pathway.
- Search algorithms are employed in games like chess and tic-tac-toe.
- Search algorithms can be used in automated planning and scheduling.
- Search algorithms are employed to solve problems like Water Jug Problem and 8-Puzzle Problem.

State Space Representation

State Space Representation is a mathematical model in Artificial Intelligence which represents a problem as a set of states and transitions among these states. It is helpful for an AI to imagine all the possibilities and the way to achieve the final state.

Definition

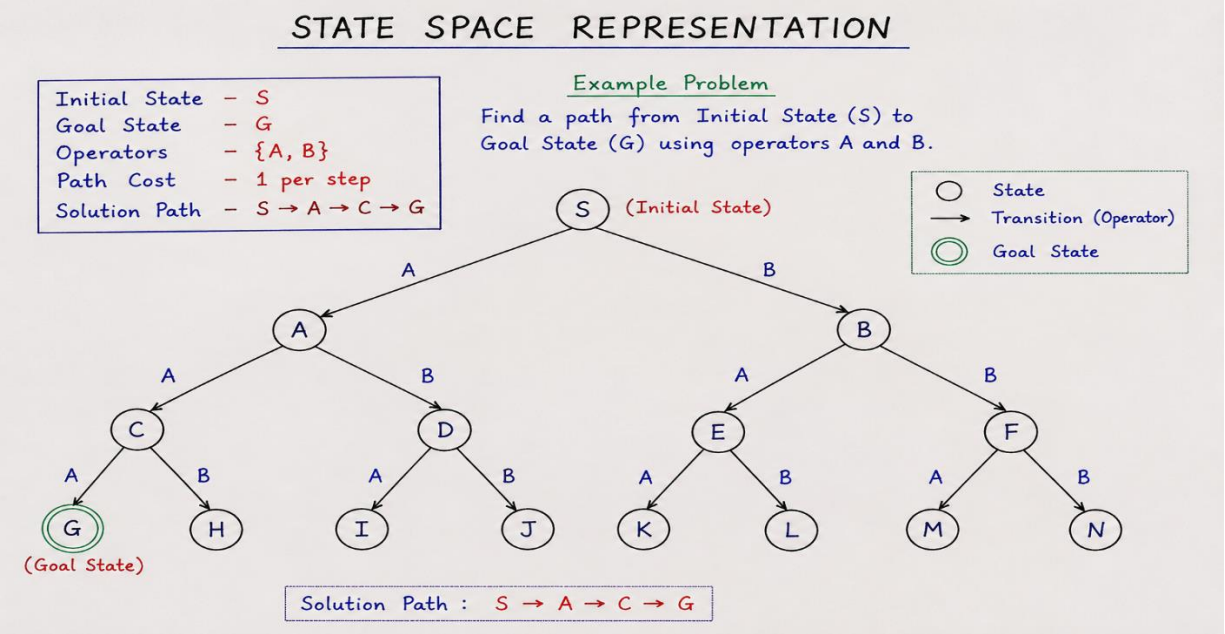
State Space Representation is a graphical or mathematical representation of all the possible states of the problem along with their transitions.

Components of State Space

- States : State refers to a certain instance of the problem.
- Initial State : Beginning instance of the problem.
- Goal State : Final desired state.
- Operators : Operators refer to actions leading from one state to another.
- State Transitions : Transition from one state to another through an action.

Properties of State Space Representation

- Represents the problem completely.
- Allows for a systematic search.
- Provides visualization of the path to the solution.
- Utilized in both informed and uninformed searches.



Advantages

- State Space Representation offers a well-defined and systematic representation of a problem.
- It enables visualization of all states that may exist within the problem.
- It facilitates the use of different kinds of search techniques.
- It makes it easy to derive paths from start state to goal state.
- It can be used for comparative study of problem-solving techniques.

Disadvantages

- The state space can get very large for certain problems.
- It would be necessary to store all states which consumes memory.
- A state space would require much time to be generated.
- There are some problems that cannot be properly modeled by state spaces.
- A state space could lead to inefficient search processes.

Applications

- The State Space Representation technique is employed in puzzle problems like the 8-puzzle and water jug problem.
- State Space Representation technique is used in robotic navigation.
- State Space Representation technique is used in artificial intelligence for game playing.
- The State Space Representation technique aids in planning and decision-making systems.
- State Space Representation technique is extensively used in Artificial Intelligence.

Blind / Uninformed Searches: Breadth First Search (BFS) – Monkey Banana

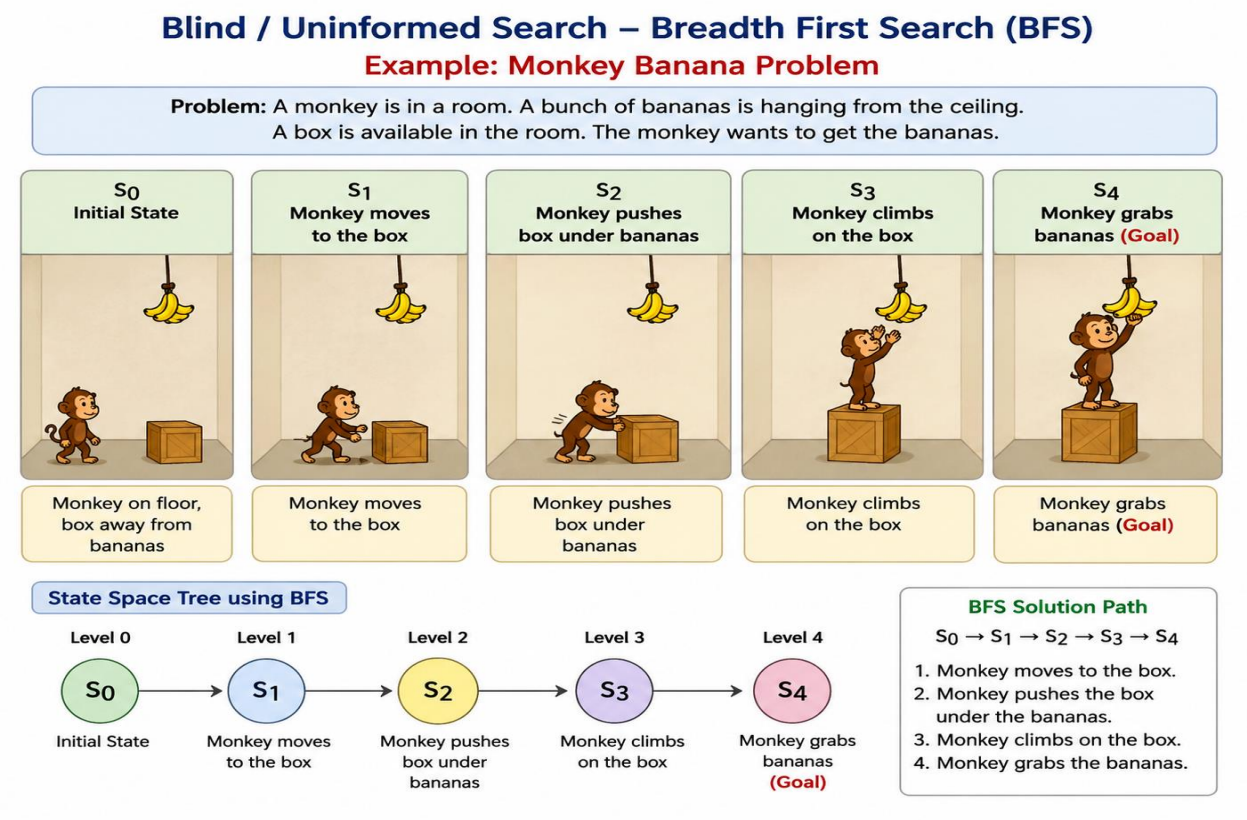
Breadth First Search (BFS) is an uninformed, blind search technique which traverses the state space tree level-wise utilizing the FIFO approach. In the case of Monkey and Banana problem, BFS simultaneously searches all paths including walking, pushing the box, climbing, and grabbing, guaranteeing that the solution obtained is the best, although it uses considerable memory resources**.**

BFS Algorithm

Step 1: Begin with the starting state and place it in the queue.
Step 2: Take out the first state in the queue.
Step 3: Determine if the goal state is attained.
Yes? Terminate and provide the solution.
Step 4: Create all successor states that can be generated.
Step 5: Place the unvisited successors in the queue.
Step 6: Continue performing Steps 2 to 5 until the goal is attained
or the queue runs out.
Step 7: Stop / Finish.

State Representation Monkey Banana State	**Description**
S0	Monkey at A, Box at B, Banana at C
S1	Monkey moves to Box
S2	Monkey pushes Box under Banana
S3	Monkey climbs the Box
S4	Monkey grabs Banana (Goal)

BFS Solution for Monkey Banana Problem



The Monkey Banana Problem is solved via the application of BFS algorithm till the goal state of reaching the banana is achieved.

Water Jug Problem

In the context of Artificial Intelligence problems, the Water Jug Problem is a typical example where there are two jugs of different sizes provided along with the task of measuring a certain amount of water. This problem has been modeled through states and solved via operations.

Problem Statement

Given:

Jug A = 4 liters

Jug B = 3 liters

Goal:

Measure exactly 2 liters of water.

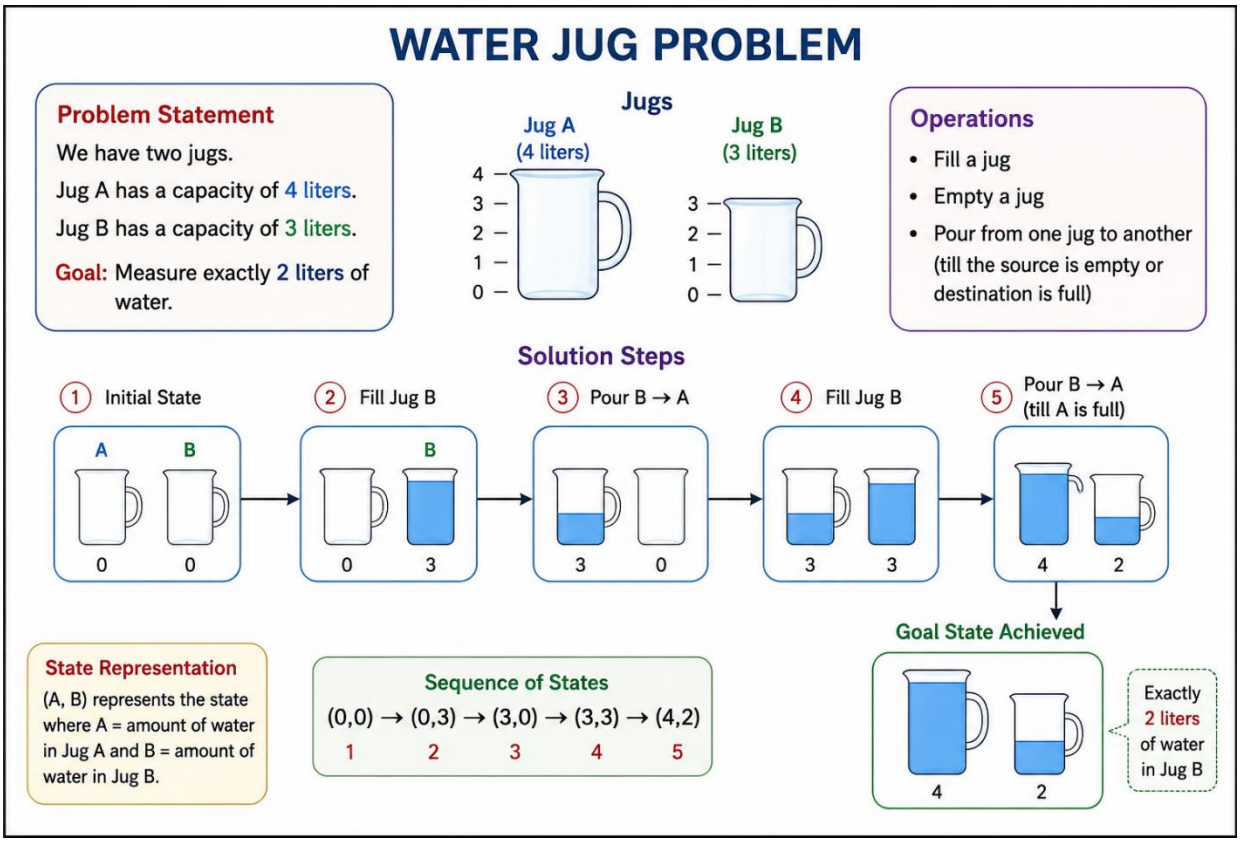
Operators

- Fill Jug A to its fullest.
- Fill Jug B to its fullest.
- Empty Jug A.
- Empty Jug B.

- Transfer water from Jug A into Jug B.
- Transfer water from Jug B into Jug A.
- Depth Limited Search

State Space Solution

Step	State (A,B)	Action
1	(0,0)	Initial State
2	(0,3)	Fill Jug B
3	(3,0)	Pour B → A
4	(3,3)	Fill Jug B
5	(4,2)	Pour B → A until A is full
Goal	(4,2)	2 liters obtained in Jug B



A solution to the Water Jug Problem has been found using an appropriate series of filling and transferring steps, leading to the correct measure of 2 liters of water in Jug B.

Depth Limited Search

DLS (Depth Limited Search) is a variant of DFS (Depth First Search), where a predetermined depth limit is established for search purposes. In DLS, nodes are explored to a limited depth, below which no nodes are expanded during search.

DLS aids in eliminating the infinite path issue faced in DFS.

Key Idea

- Search operates in a depth-first manner.
- Depth limit (L) is fixed.
- Nodes exceeding this limit cannot be expanded.
- The search terminates once the solution is obtained or depth limit is exceeded.

Algorithm

DepthLimitedSearch(Node, Goal, Limit)

Step 1: IF Node == Goal THEN

RETURN Success

Step 2: ELSE IF Limit == 0 THEN

RETURN Cutoff

Step 3: FOR each Child of Node DO

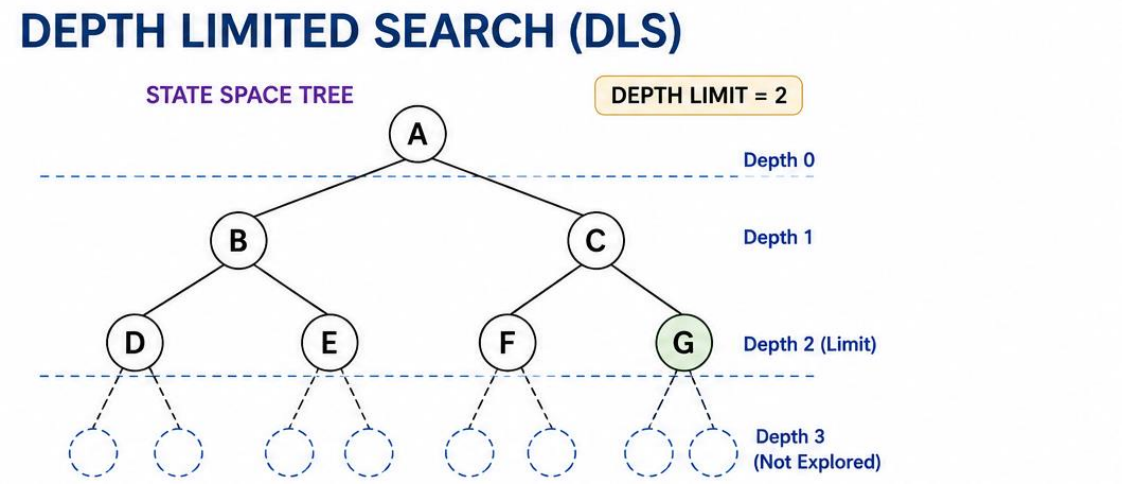
Result ← DepthLimitedSearch(Child, Goal, Limit – 1)

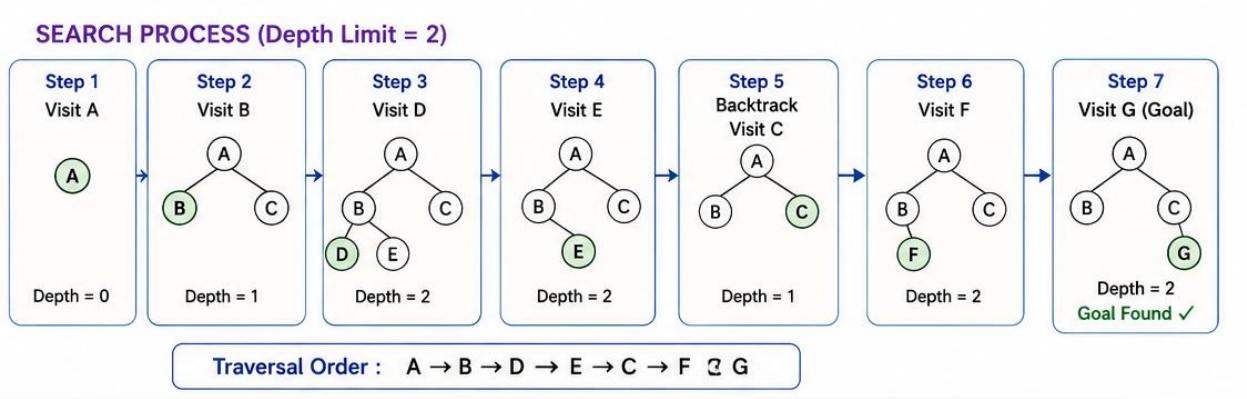
IF Result == Success THEN

RETURN Success

Step 4: Return Failure

Step 5: END






```
from collections import deque

# BFS for Monkey Banana Problem
queue = deque(["Monkey at Door"])
visited = []
while queue:
    state = queue.popleft()

    if state not in visited:
        visited.append(state)
        print(state)

        if state == "Monkey at Door":
            queue.append("Monkey at Box")

        elif state == "Monkey at Box":
            queue.append("Box Under Banana")

        elif state == "Box Under Banana":
            queue.append("Monkey Climbs Box")

        elif state == "Monkey Climbs Box":
            queue.append("Monkey Gets Banana")

        elif state == "Monkey Gets Banana":
            print("\nGoa : Banana Obtained!")
            break
/*OUTPUT
Monkey at Door
Monkey at Box
Box Under Banana
Monkey Climbs Box
Monkey Gets Banana
Goal: Banana Obtained!*/
```

Example - Water Jug Problem using State Space Search

```
from collections import deque

def water_jug():
    visited = set()
    queue = deque([(0,0)])

    while queue:
        x,y = queue.popleft()

        if (x,y) in visited:
            continue

        visited.add((x,y))

        if x == 2 or y == 2:
            print("Solution:", (x,y))
            return

        states = [
            (4,y),(x,3),(0,y),(x,0),
            (max(0,x-(3-y)), min(3,x+y)),
            (min(4,x+y), max(0,y-(4-x)))
        ]

        for s in states:
            queue.append(s)

water_jug()

/*OUTPUT
Solution: (2,3)*/
```

Lab Assignments

SET A

1. Write a program to implement Breadth First Search (BFS) for the Monkey Banana Problem.
2. Write a program to solve the Water Jug Problem using BFS.
3. Write a program toiImplement BFS traversal for a graph.

4. Write a program to solve a state-space search problem using BFS.
5. Write a program to implement Depth Limited Search (DLS).

Signature of the Instructor | | **Date** | |

SET B

1. Write a program to implement DFS traversal for a graph.
2. Write a program to solve the Missionaries and Cannibals Problem.
3. Write a program to solve the Tower of Hanoi Problem using state-space representation.
4. Write a program implement Iterative Deepening Search (IDS).
5. Write a program to analyze the time and space complexity of BFS.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 2: Informed Search Techniques (Total slots = 2)

Objective

- To grasp the notion of informed search methods in artificial intelligence.
- To know how heuristic function directs the searching process towards the goal state in an efficient manner.
- To explore different informed search algorithms such as best first search, A*, AO*, hill climbing, and means-end analysis.

Ready

- Heuristics search algorithms utilize extra information called heuristics in order to limit the scope of the search and make searches more efficient than uninformed search algorithms.
- A heuristic algorithm will provide an estimate of the cost or distance from the current state to the target state.

Ready Reference

Informed Search Techniques (also called heuristic search techniques) are those search techniques that incorporate other knowledge about the problem and search more effectively for the goal state. In contrast to uninformed search techniques, heuristic search makes use of heuristics or an evaluation function to assess the better path towards the goal. An evaluation function is denoted by h(n) and n is the current node.

Characteristics

- Includes domain-specific knowledge.
- Minimizes superfluous search.
- Finds solutions faster than blind search in many situations.
- Applicable to large-scale search problems.
- Often employed in applications of Artificial Intelligence.

Best First Search

Best First Search is a type of heuristic search technique where a node with the greatest potential to lead toward the solution based on heuristics h(n) is selected for expansion.

The OPEN list in Best First Search acts like a priority queue, which contains all the nodes in increasing order of their heuristic evaluation.

Principle of Operation

- Begin at the starting point.
- Determine the heuristic value of the neighboring points.

- Select the point with the lowest estimated heuristic value.
- Continue this process until the target point is found.

Algorithm Best_First_Search

Algorithm Best_First_Search(Start, Goal)

Step 1. Create an OPEN list and a CLOSED list.

Step 2. Insert the Start node into the OPEN list.

Step 3. Repeat until OPEN is empty:

a. Remove the node with the lowest heuristic value from OPEN.

b. If the node is the Goal, return Success.

c. Move the node to CLOSED.

d. Generate all successor nodes.

e. For each successor:

i. If it is not in OPEN or CLOSED,

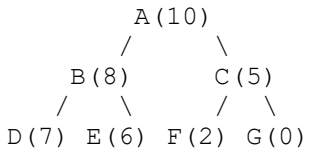
calculate its heuristic value.

ii. Insert it into OPEN.

Step 4. If OPEN becomes empty, return Failure.

Example of Best First Search

Consider the following graph where the number in brackets is the heuristic value **h(n)**.



Goal Node = G

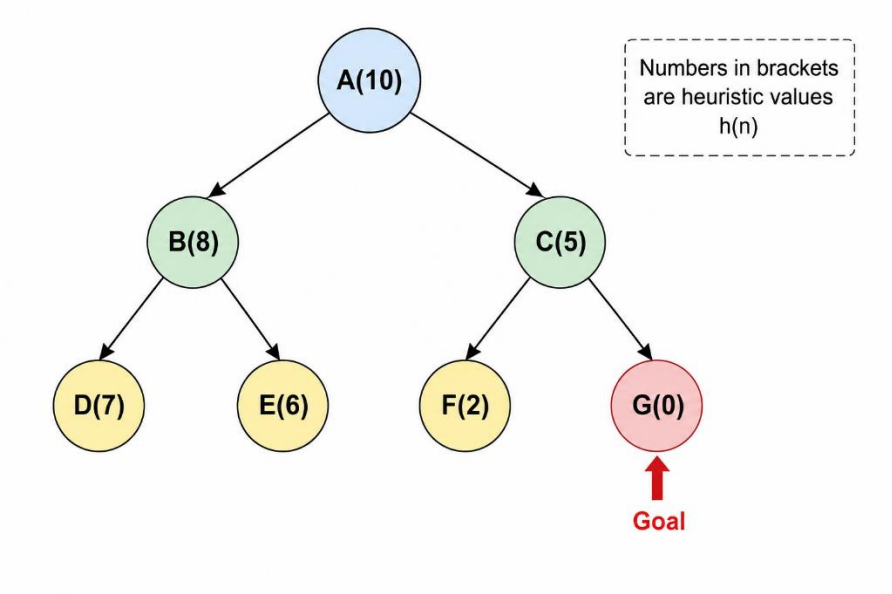
Step-by-Step Execution

Step	OPEN List	Selected Node
1	A(10)	A
2	B(8), C(5)	C
3	B(8), F(2), G(0)	G
4	Goal Found	Stop

Traversal Order

A → C → G

Since **G** has the lowest heuristic value (0), the algorithm reaches the goal quickly without exploring unnecessary nodes.



Time Complexity

- Worst Case: $O(b^d)$
- Space Complexity: $O(b^d)$

Where:

- b = branching factor (number of successors per node)
- d = depth of the search tree

Advantages

- Uses heuristic information to make the search.
- Finds the solution before other algorithms in most cases.
- Saves time because it avoids exploring unnecessary nodes.
- Easy to implement when using a priority queue structure.

- Works well in solving many AI search problems.

Disadvantages

- Very dependent on the heuristic information used.
- The algorithm may yield a wrong solution if the heuristic is not right.
- Takes up memory resources due to the large number of nodes it explores.
- May explore similar states again unless properly coded.

Example of Real Life Usage

If the problem is to identify a route from Home to College, then the navigation system determines the approximate value of the distance from the current point to the destination:

Point	**Heuristic Value for Destination**
Home	12 km
Market	8 km
Park	5 km
Bus Stand	3 km
College	0 km

Then, Best First Search will always pick the current point with minimum heuristic value, producing the following sequence:

Home → Park → Bus Stand → College

```
/* To implement Best First Search using heuristic values.*/

from queue import PriorityQueue

# Graph representation
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': [],
    'E': [],
    'F': [],
    'G': []
}

# Heuristic values
heuristic = {
    'A': 10,
    'B': 8,
    'C': 5,
```

```
    'D': 7,
    'E': 6,
    'F': 2,
    'G': 0
}

def best_first_search(start, goal):
    visited = set()
    pq = PriorityQueue()

    # Insert start node with its heuristic value
    pq.put((heuristic[start], start))

    while not pq.empty():
        h, current = pq.get()

        if current in visited:
            continue

        print(current, end=" ")
        visited.add(current)

        if current == goal:
            print("\nGoal Found!")
            return

        # Add neighbors to the priority queue
        for neighbor in graph[current]:
            if neighbor not in visited:
                pq.put((heuristic[neighbor], neighbor))

    print("\nGoal Not Found!")

# Driver code
start_node = 'A'
goal_node = 'G'

print("Traversal using Best First Search:")
best_first_search(start_node, goal_node)

/*OUTPUT
Traversal using Best First Search:
A C G
Goal Found!
*/
```

A* Algorithm

A* (A Star) Algorithm is an intelligent search algorithm used to discover the shortest and best path from a source vertex to the target vertex. It uses the actual cost from the source vertex to the next vertex and an estimated cost to reach the target vertex.

Evaluation function is:

$$f(n)=g(n)+h(n)$$

Where:

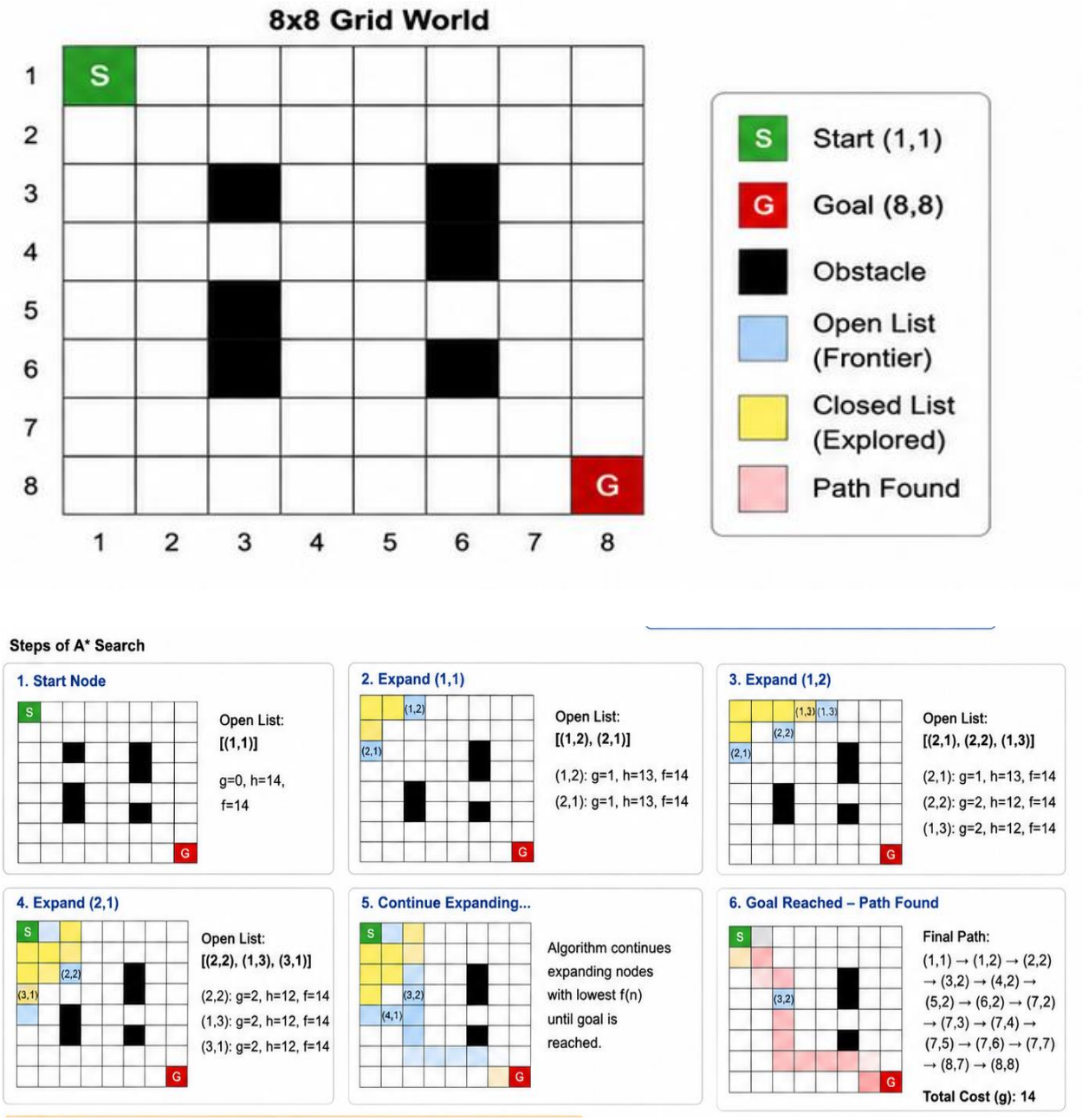
- f(n) = Total estimated cost of the solution through node *n*
- g(n) = Actual cost from the start node to *n*
- h(n) = Estimated cost (heuristic) from *n* to the goal

Characteristics of A* Algorithm

- Incorporates actual as well as heuristic costs.
- Generates shortest path under the condition of heuristic being admissible.
- Faster than most uninformed search algorithms.
- Applied extensively in Artificial Intelligence and path finding.
- Maintains an OPEN list as well as a CLOSED list for nodes.

Algorithm of A* Search

Algorithm AStar(Start, Goal)
Step 1. Create OPEN and CLOSED lists.
Step 2. Put the Start node into OPEN.
Step 3. Set g(Start) = 0.
Step 4. Compute f(Start) = g(Start) + h(Start).
Step 5. Repeat until OPEN is empty:
  a. Select the node with the smallest f(n).
  b. If it is the Goal, stop and return the path.
  c. Move the selected node to CLOSED.
  d. Generate its successors.
  e. For each successor:
    i. Compute g(n).
    ii. Compute h(n).
    iii. Compute f(n) = g(n) + h(n).
    iv. Add or update the node in OPEN if a better path is found.
Step 6. If OPEN becomes empty, report failure.



Assume the algorithm is at node **(3,2)**:

Cost from Start to (3,2): g(n) = 4

Estimated cost from (3,2) to Goal: h(n) = 9

Then, f(n) = 4 + 9 = 13

The algorithm compares the f(n) values of all candidate nodes and expands the node with the smallest f(n).

Cost Table

Node	g(n)	h(n)	f(n) = g(n) + h(n)
Start (1,1)	0	14	14
(1,2)	1	13	14
(2,2)	2	12	14
(3,2)	3	11	14
(4,2)	4	10	14
Goal (8,8)	14	0	14

Total Path Cost if each move costs 1 unit, then:

Number of moves from Start to Goal = 14

Total path cost = 14

Therefore, the final cost reported by the A* algorithm is 14 because 14 steps are taken to reach the target from that particular route. If we take into consideration diagonal movements and weighted edges, then the total cost will be the total of the cost of these movements.

AO* Algorithm

AO* (And-Or Star) algorithm is an informed search algorithm that is used to discover the optimal solution within AND-OR graphs. In comparison to A* algorithm that operates on OR graphs, the AO* algorithm can address problems in which certain tasks need all subtasks to be solved (AND nodes) and other tasks just require one choice (OR nodes).

Heuristic values are used by the algorithm to estimate the cost towards the goal node and constantly refine the path towards the optimal solution graph.

Key Concept

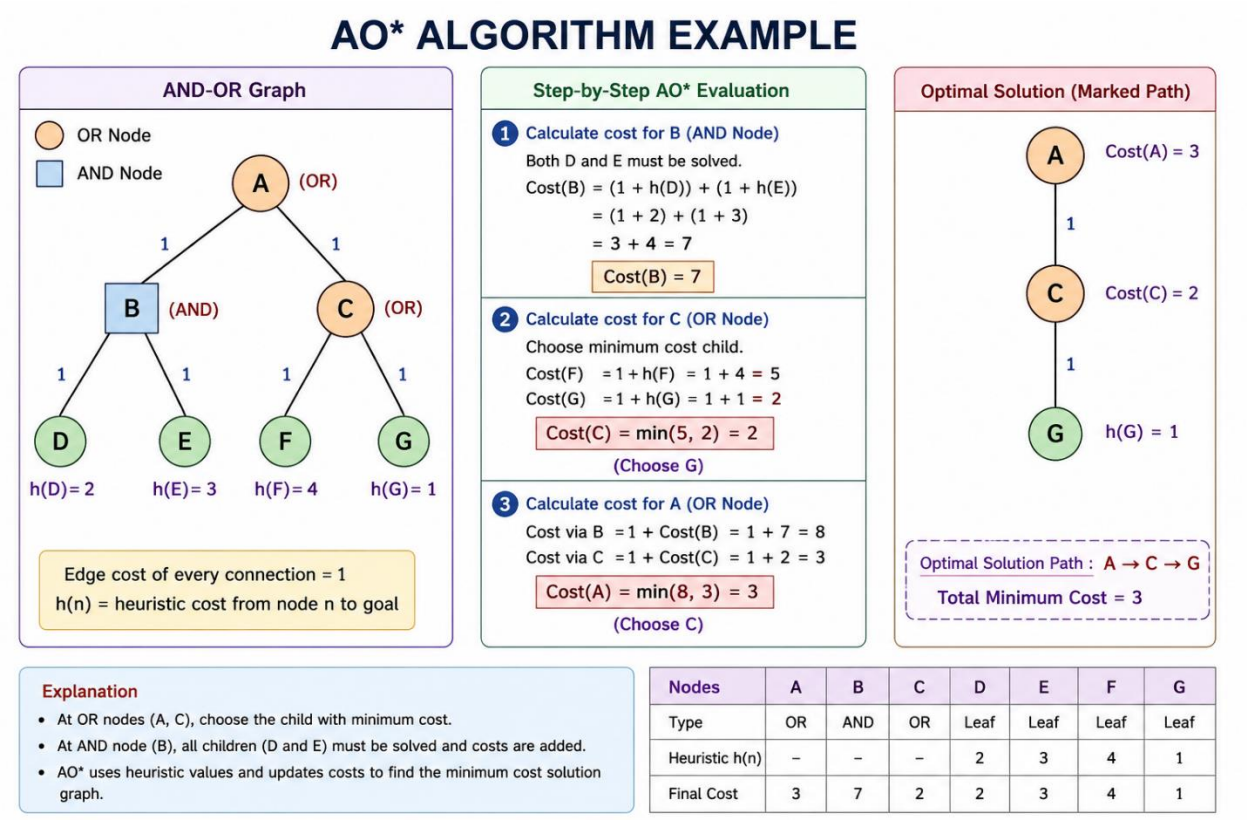
- OR Node: It suffices to select only one of the child nodes.
- AND Node: Solution to all child nodes is necessary for solving the parent node.

- Heuristic Function (h): Estimation of cost left to go from node to goal.
- Cost Function: Total cost = Cost on edge + Cost of heuristic.

AO* Algorithm

Step 1 : Start with the root node.
Step 2 : Heuristic value is assigned to all nodes.
Step 3 : Expand the node which has the best chance of giving us the minimum cost.
Step 4 : In case of OR node, select that node whose cost is minimum.
Step 5 : In case of AND node, select all nodes needed and calculate their cost.
Step 6 : Update the heuristic value for all parent nodes.
Step 7 : Continue till all goal nodes are reached.
Step 8 : Trace the marked nodes to get the optimal solution graph**.**

Example



Nodes	A	B	C	D	E	F	G
Type	OR	AND	OR	Leaf	Leaf	Leaf	Leaf
Heuristic h(n)	–	–	–	2	3	4	1
Final Cost	3	7	2	2	3	4	1

The AO* algorithm is an informed search approach that operates on AND-OR graphs in which some choices are to select among several alternatives, while other choices entail the completion of more than one task.

Through the use of heuristics and cost evaluations, AO* manages to find an optimal graph and has many practical applications in artificial intelligence.

Heuristic Functions

A Heuristic Function is an approach that is employed in Artificial Intelligence (AI) to calculate the estimated cost or distance between the present state and the goal state. This approach enables the search algorithm to make an informed decision about which path will be most effective rather than considering all possible paths.

The heuristic function can be described as:

h(n)=Estimated cost from node n to the goal

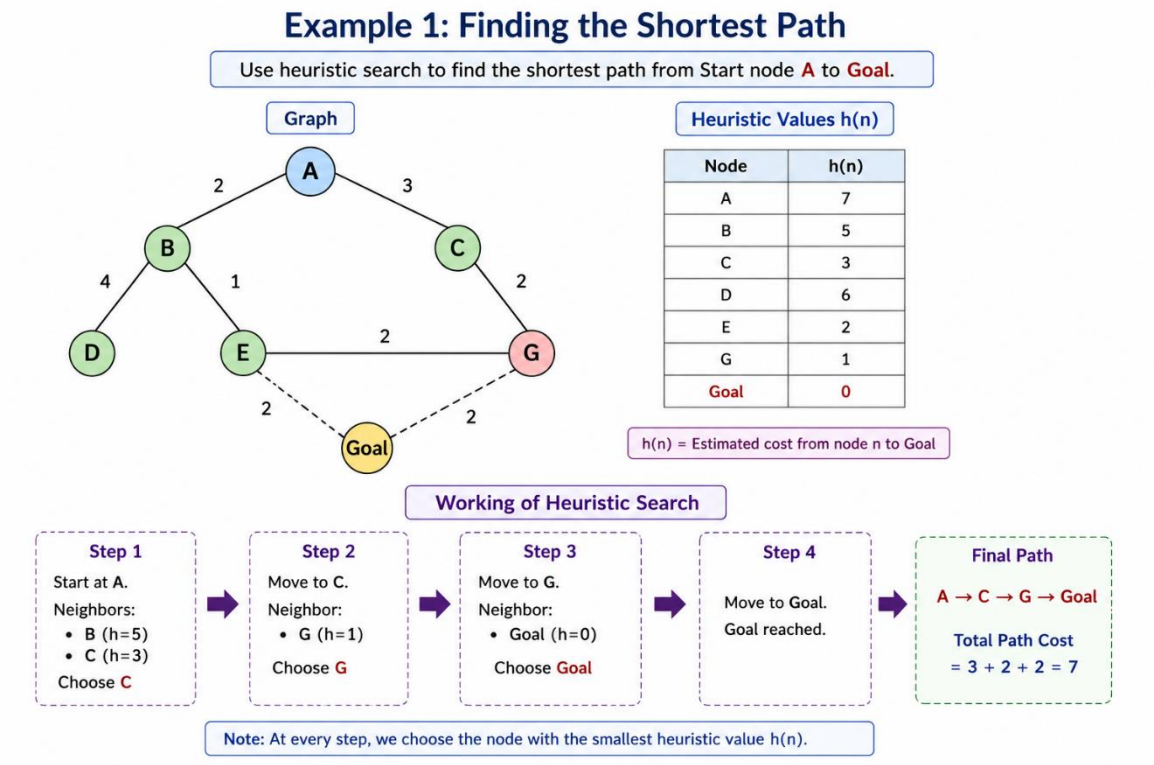
Where:

n=Current node

h(n)=Estimated cost to goal

Pseudocode

```
Algorithm Heuristic_Search(Start, Goal)

1. Place Start node in OPEN list.
2. While OPEN list is not empty:
    a. Select the node with the lowest heuristic value h(n).
    b. Remove it from OPEN.
    c. If it is the Goal, stop and return the path.
    d. Generate its neighboring nodes.
    e. Compute h(n) for each neighbor.
    f. Add unvisited neighbors to OPEN.
3. If OPEN becomes empty, report failure.
```



Types of Heuristic Functions

- Admissible Heuristic
  - Never overestimates the actual cost.
  - Guarantees an optimal solution in algorithms like A*.
- Consistent (Monotonic) Heuristic
  - Satisfies the triangle inequality.
  - Ensures stable and efficient search.
- Manhattan Distance
  - Used in grid-based navigation where movement is limited to horizontal and vertical directions.
- Euclidean Distance
  - Uses straight-line distance between two points.
- Chebyshev Distance
  - Suitable when movement in eight directions is allowed.

The heuristic function represents the estimation of the cost that will be needed in order to reach the goal and is a core element in informed searches. Through pointing the search to promising states, it enables the saving of significant computational time. Proper heuristic functions are extremely important in algorithms like Best-First Search and A*.

Hill Climbing

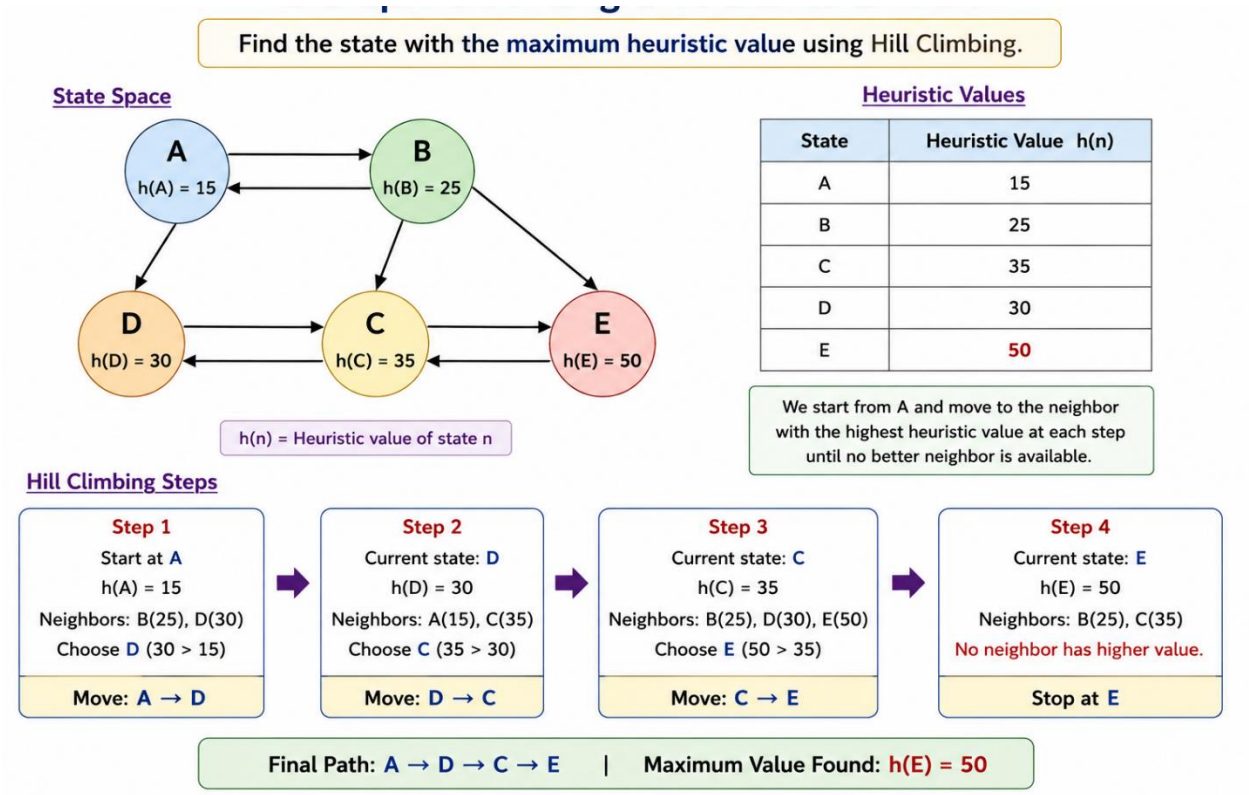
The Hill Climbing technique is one such heuristic technique of informed search employed in AI for optimizing functions. Starting from an initial state, it iteratively transitions to a new state based on better heuristic values. The procedure is continued till there is no further possibility of transitioning to a state with higher heuristic value.

The basic principle is to keep moving towards the goal through the optimal possible path at any stage of transition, like climbing a hill.

Pseudocode

```
Algorithm Hill_Climbing(Start)

1. Set Current = Start.
2. Repeat:
    a. Generate all neighboring states of Current.
    b. Evaluate each neighbor using a heuristic function.
    c. Select the best neighbor.
    d. If the best neighbor is not better than Current,
        return Current (stop).
    e. Otherwise, set Current = Best Neighbor.
3. End.
```

Example : Finding the Maximum Value



Types of Hill Climbing

- Simple Hill Climbing
  - Examines neighbors one by one and moves to the first better state found.
- Steepest-Ascent Hill Climbing
  - Evaluates all neighbors and selects the one with the greatest improvement.
- Stochastic Hill Climbing
  - Randomly chooses among improving neighbors, often helping explore multiple possibilities.

Means-End Analysis

MEANS-END ANALYSIS (MEA) is a problem-solving method utilized in Artificial Intelligence (AI). This involves comparing the initial state with the final state and then determining the differences between the two and choosing an appropriate course of action (means) to minimize the gap between the two. The cycle is continually repeated till the end is met.

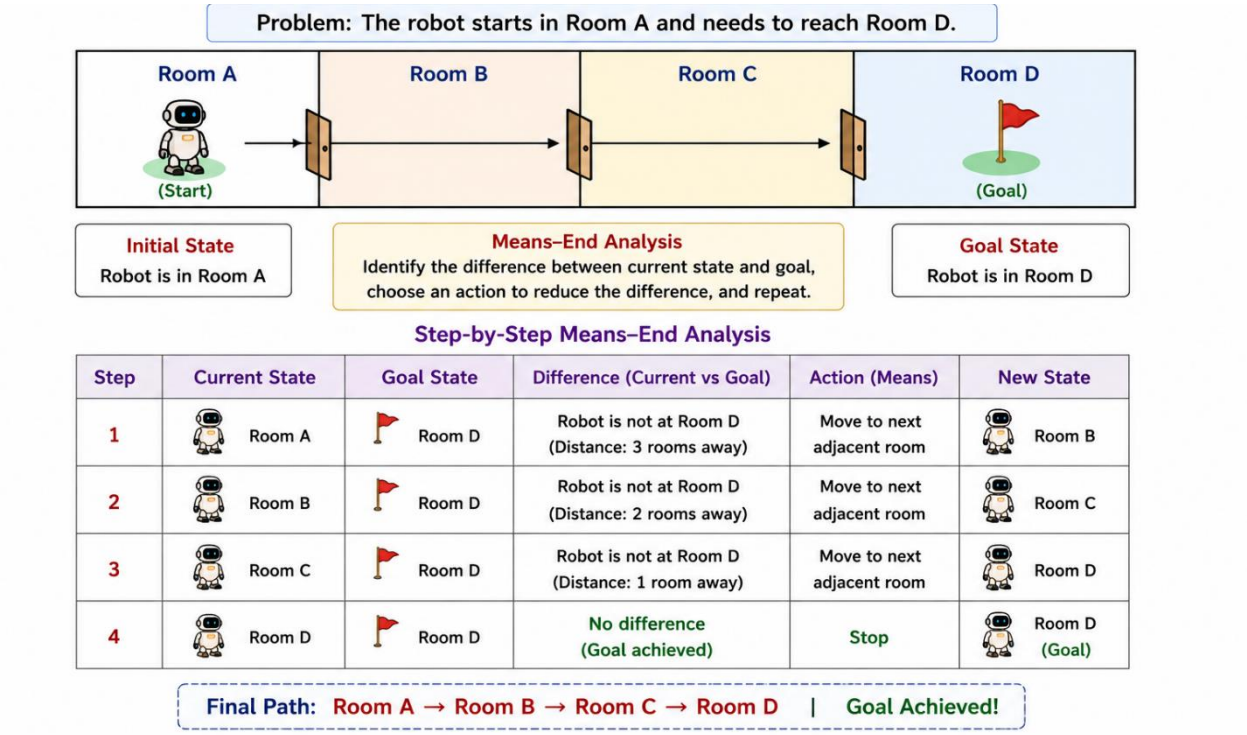
The basic concept involved is:

Determine the difference → Choose means to minimize difference → Execute the means → Continue till the goal is met.

Pseudocode

```
Algorithm Means_End_Analysis(Start, Goal)

1. Set Current_State = Start.
2. While Current_State ≠ Goal do:
    a. Find the difference between Current_State and Goal.
    b. Select an operator that reduces the difference.
    c. Apply the operator.
    d. Update Current_State.
3. Return Goal State.
```

Example : Robot Moving to a Target Room



Means-End Analysis (MEA) is a technique of solving problems heuristically wherein there is a continuous comparison between the current state and the goal state, and taking actions in such a way that the gap between these states is minimized.

Through the decomposition of a difficult problem into subproblems, means-end analysis presents a systematic and effective method of planning and search in artificial intelligence. Means-end analysis works best in situations wherein there are differences to be eliminated between the current and goal state.

Lab Assignments

SET A

1. Write a program to implement the Best First Search Algorithm using heuristic values to find a path from a start node to a goal node.
2. Write a program to implement the A* Search Algorithm to find the shortest path between a source node and a destination node.
3. Write a program to solve the 8-Puzzle Problem using the A* Search Algorithm.
4. Write a program to demonstrate the Hill Climbing Algorithm for finding the maximum value of a simple objective function.

5. Write a program to implement Means-End Analysis for solving a goal-based problem.

Signature of the Instructor | | **Date** | |

SET B

1. Write a program to implement the AO* (AND-OR Star) Algorithm for solving an AND-OR graph.
2. Write a program to implement the Greedy Best First Search Algorithm using heuristic values.
3. Write a program to solve a route-finding problem using a heuristic search algorithm.
4. Write a program to demonstrate the use of heuristic functions in solving a search problem.
5. Write a program that implements both A* and Greedy Best First Search algorithms and compares their performance on the same graph.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 3 : Constraint Satisfaction Problems (CSP) (Total slots = 1)

Objective

- To gain an understanding of Constraint Satisfaction Problem (CSP) in AI.
- To understand the process of using variables, domains, and constraints in problem solving.
- To find solution that satisfies all constraints involved.

Ready

- The Constraint Satisfaction Problem (CSP) is a problem where the assignment of a value to a set of variables occurs based on some constraints.
- The CSP problem can be described based on three major factors, which are: Variables, Domains, and Constraints. Examples of CSP problems include map coloring, scheduling, timetabling, N queens, and Sudoku.

Ready Reference

Constraint satisfaction problem (CSP) refers to an artificial intelligence problem which involves assigning a given number of variables values that will fulfill a given number of constraints. The goal of a constraint satisfaction problem is to achieve fulfillment of all constraints at once.

A constraint satisfaction problem is made up of three elements namely:

- Variables (X): Unspecified quantities that require values.
- Domains (D): Values that the variables can assume.
- Constraints (C): Rules that govern the values assumed by the variables.

Backtracking algorithm

The backtracking technique is an intelligent searching approach that involves trying different approaches for solving the problem on a step-by-step basis. When the selected approach fails to satisfy the given criteria or does not help reach the target state, then the algorithm backtracks and considers other alternatives.

Backtracking is extensively applied in Constraint Satisfaction Problems, puzzles, pathfinding, and combinatorial optimization.

Pseudocode

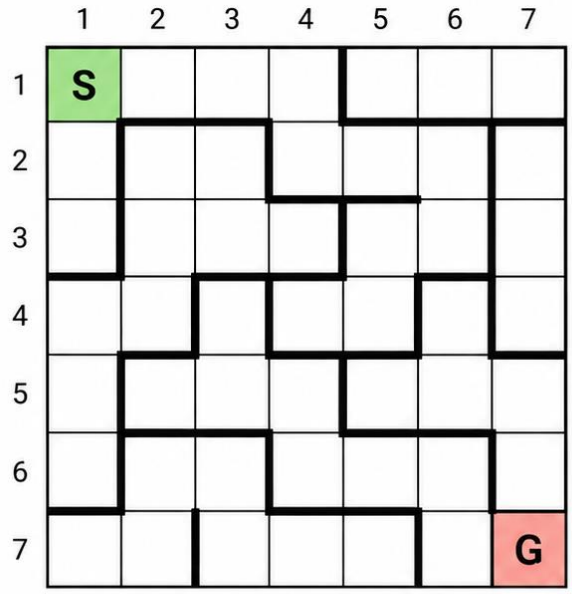
```
Algorithm Backtracking(State)

1. If State is a complete solution:
      Return State.
2. Select the next variable or decision.
3. For each possible choice:
      a. Make the choice.
```

b. If the choice is valid:
Call Backtracking(New State).
c. If the solution is found:
Return it.
d. Otherwise:
Undo the choice (Backtrack).
4. Return Failure if no valid solution exists.

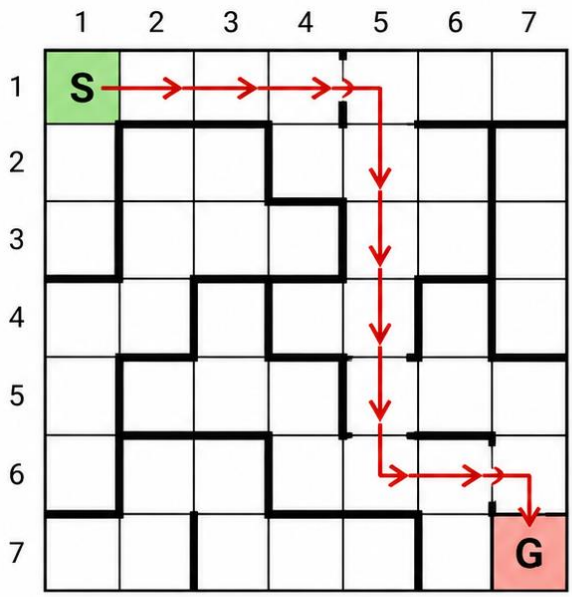
Example: Maze Path Finding

Objective: Find a path from the Start position to the Goal position in the maze.

1. Maze



2. Solution Path



S = Start (1,1)
G = Goal (7,7)
= Wall
= Path / Open cell

Solution Path (one possible path):
(1,1) → (1,2) → (1,3) → (1,4) → (1,5) →
(2,5) → (3,5) → (4,5) → (5,5) → (6,5) →
(6,6) → (6,7) → (7,7)

Explanation:

- The maze is represented as a 7x7 grid.
- S is the Start position and G is the Goal position.
- Black cells are walls (cannot be crossed).
- The red arrows show one possible path from Start to Goal.

Time Complexity

Worst Case: $O(b^d)$, where ;

b = branching factor (number of choices at each step)

d = maximum depth of the search tree

Backtracking Algorithm is a useful search algorithm that builds up solutions one step at a time and discards solutions when the solution violates any constraints or leads to a dead end. This algorithm works best in cases where there is a need to explore various options.

Domain Reduction Works

Domain Reduction is a method applied in CSPs which enables the pruning of the possible values (the domain) of variables before or while searching for a solution. Domain reduction limits the search space and aids in finding a solution by removing those values that go against the constraints. This approach rules out all the wrong choices leaving behind only correct ones.

Objectives

- Minimizing the set of possible values for the variables.
- Elimination of inconsistencies in early stages.
- Improving efficiency of search algorithms.
- Simplification of solving constraint problems.

Algorithm: Domain Reduction

Step 1: Initialize the domains of all variables.

Step 2: Select a variable and examine its constraints.

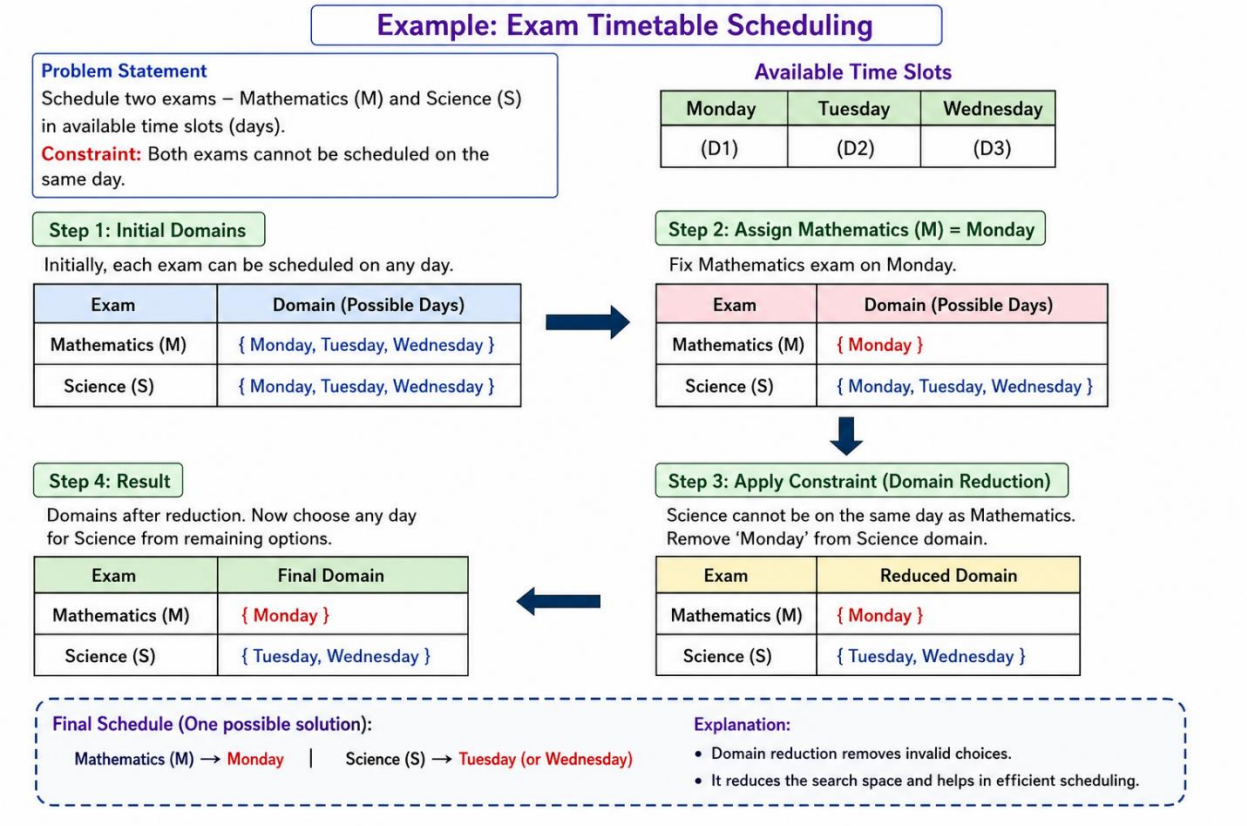
Step 3: Remove values from its domain that violate any constraint.

Step 4: If a domain changes, update related variables.

Step 5: Repeat Steps 2–4 until no more values can be removed.

Step 6: If any variable has an empty domain, report failure.

Step 7: Otherwise, use the reduced domains to find a solution.



Domain reduction is a critical phase in preprocessing techniques applied to Constraint Satisfaction Problems where those values that are incompatible with the imposed constraints are eliminated. This process helps in decreasing the search space and thus makes it easier to find a solution**.**

GRAPH/Map Coloring Problem

The Graph Coloring Problem can be defined as a constraint satisfaction problem, whereby colors are allocated to the vertices of a graph in such a way that no two adjacent vertices will carry the same color.

Map Coloring Problem is one application of graph coloring whereby adjoining regions in a map should be colored differently to avoid confusion.

Objective

- Assign colors to all vertices or regions.
- No two adjacent vertices should be assigned the same color.
- Try to use the least number of colors wherever possible.

Basic Terminology

- Vertex (Node): A vertex in a graph denoting an object or region.

- Edge: An edge connecting two vertices.
- Adjacent Vertices: Vertices which are directly connected via an edge.
- Color: Color assigned to a vertex**.**

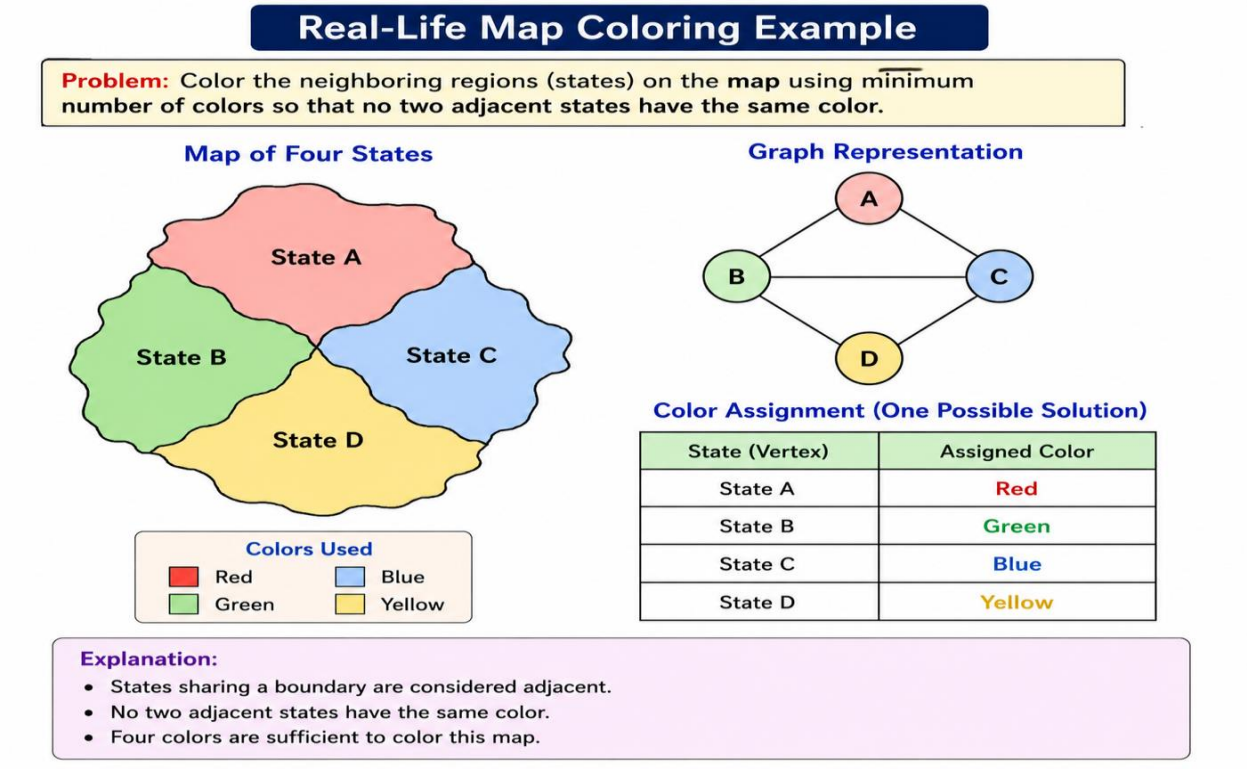
Algorithm for Graph Coloring

Algorithm GraphColoring(vertex)

Input:
Graph G(V, E)
m available colors

Step 1: Start with the first vertex.
Step 2: Assign a color that is not used by any adjacent vertex.
Step 3: Move to the next vertex.
Step 4: If no valid color is available,
remove the previous assignment (Backtrack)
and try another color.
Step 5: Repeat until all vertices are colored.
Step 6: If all vertices receive valid colors,

output the solution.
Otherwise, report that no coloring exists with m colors.



The Graph/Map Coloring Problem is a standard Artificial Intelligence problem in which the colors are allocated in such a way that adjacent vertices or regions are never of the same color. By using methods like backtracking and constraint satisfaction, graph coloring offers an efficient solution to problems related to map making, scheduling, and optimization.\

Program of Graph Coloring using Backtracking

```
# Program of Graph Coloring using Backtracking

# Function to check if assigning color c to vertex v is safe
def is_safe(v, graph, color, c):
    for i in range(len(graph)):
        if graph[v][i] == 1 and color[i] == c:
            return False
    return True

# Backtracking function
def graph_coloring(graph, m, color, v):
    # If all vertices are assigned a color
    if v == len(graph):
        return True

    # Try different colors
    for c in range(1, m + 1):
        if is_safe(v, graph, color, c):
            color[v] = c

            if graph_coloring(graph, m, color, v + 1):
                return True

            # Backtrack
            color[v] = 0

    return False


# -------- Main Program --------

# Adjacency matrix of the graph
# Vertices: A, B, C, D
graph = [
    [0, 1, 1, 0],  # A
    [1, 0, 1, 1],  # B
    [1, 1, 0, 1],  # C
    [0, 1, 1, 0]   # D
]
```

```
# Number of colors available
m = 3

# Color array (0 means no color assigned)
color = [0] * len(graph)

if graph_coloring(graph, m, color, 0):
    color_names = ["", "Red", "Green", "Blue"]
    vertices = ["A", "B", "C", "D"]

    print("Graph Coloring Solution:")
    for i in range(len(vertices)):
        print(vertices[i], "->", color_names[color[i]])
else:
    print("No solution exists with", m, "colors.")

/*OUTPUT
Graph Coloring Solution:
A -> Red
B -> Green
C -> Blue
D -> Red */
```

Sudoku

A Sudoku is a puzzle game and is one of the many examples of CSP in Artificial Intelligence. It involves filling up a 9 × 9 grid using numbers ranging from 1 to 9 such that:

- Each row has all the numbers from 1 to 9.
- Each column has all the numbers from 1 to 9.
- Every 3 × 3 box within the grid has all the numbers from 1 to 9.

The above conditions must be fulfilled while filling up empty spaces.

Objectives

- All the empty cells should have proper numbers.
- No digit should be repeated within any row, column, and 3 × 3 box.
- A proper solution should be found out through logic and/or algorithmic methods.

Rules of Sudoku

- Each row should have the numbers 1–9 without any repetition.
- Each column should have the numbers 1–9 without any repetition.
- Each 3 × 3 box should have the numbers 1–9 without any repetition.

- There is only one number in each cell.

Algorithm SudokuSolver(board)

Step 1: Find an empty cell.
Step 2: Try placing numbers 1 to 9.
Step 3: Check whether the number is valid:
- Not present in the same row.
- Not present in the same column.
- Not present in the same 3×3 box.

Step 4: If valid, place the number.
Step 5: Recursively solve the remaining puzzle.
Step 6: If no valid number works, remove the current number (Backtrack) and try the next one.
Step 7: Repeat until the puzzle is solved.

Example Sudoku Grid

Fill the 9×9 grid so that every row, column and 3×3 subgrid contains the numbers 1 to 9 exactly once.

1. Initial Sudoku Grid (Puzzle)

5	3	.	.	7	.	.	.	.
6	.	.	1	9	5	.	.	.
.	9	8	.	.	.	.	6	.
8	.	.	.	6	.	.	.	3
4	.	.	8	.	3	.	.	1
7	.	.	.	2	.	.	.	6
.	6	.	.	.	.	2	8	.
.	.	.	4	1	9	.	.	5
.	.	.	.	8	.	.	7	9

2. Completed Sudoku Grid (Solution)

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Legend:

`·` Empty Cell

`5` Given Number

Rules:

- Each row must contain numbers 1 to 9.
- Each column must contain numbers 1 to 9.
- Each 3×3 subgrid must contain numbers 1 to 9.

Example Step: The first empty cell (Row 1, Column 3) can have 1, 2 or 4. The solver tries a number, and if it leads to a conflict, it backtracks and tries another number.

Sudoku is an example of a Constraint Satisfaction Problem, where numbers are allocated within the grid by adhering to constraints imposed on rows, columns, and smaller grids. Backtracking algorithm is mostly used in solving a Sudoku puzzle, where trial and error methods are followed until the correct answer is arrived at.

Program of Function to print the Sudoku board

```
# Program of Function to print the Sudoku board
def print_board(board):
    for row in board:
        print(row)

# Function to find an empty cell
def find_empty(board):
    for i in range(9):
        for j in range(9):
            if board[i][j] == 0:
                return (i, j)
    return None

# Function to check whether a number can be placed
def is_valid(board, row, col, num):

    # Check row
    for j in range(9):
        if board[row][j] == num:
            return False

    # Check column
    for i in range(9):
        if board[i][col] == num:
            return False

    # Check 3x3 subgrid
    start_row = (row // 3) * 3
    start_col = (col // 3) * 3

    for i in range(start_row, start_row + 3):
        for j in range(start_col, start_col + 3):
            if board[i][j] == num:
                return False
    return True
# Backtracking function
def solve_sudoku(board):
    empty = find_empty(board)
    if empty is None:
        return True
```

```
    row, col = empty
    for num in range(1, 10):
        if is_valid(board, row, col, num):
            board[row][col] = num
            if solve_sudoku(board):
                return True
            # Backtrack
            board[row][col] = 0
    return False

# ---------- Main Program ----------

sudoku = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]

print("Original Sudoku:")
print_board(sudoku)

if solve_sudoku(sudoku):
    print("\nSolved Sudoku:")
    print_board(sudoku)
else:
    print("No solution exists.")
```

Lab Assignments

SET A

1. Write a program to solve the Map Coloring Problem using the Constraint Satisfaction Problem (CSP) approach.
2. Write a program to implement the Backtracking Algorithm for solving a constraint satisfaction problem.
3. Write a program to solve a 9×9 Sudoku Puzzle using the Backtracking Algorithm.
4. Write a program to demonstrate the Domain Reduction Technique in a Constraint Satisfaction Problem.

5. Write a program to solve the N-Queens Problem using the Backtracking Algorithm.
6. Write a program or report that compares Constraint Satisfaction Problems (CSP) with traditional search techniques such as Breadth-First Search (BFS) or Depth-First Search (DFS).

Signature of the Instructor | | **Date** | |

SET B

1. Write a program to solve a Cryptarithmetic Problem using the Constraint Satisfaction Problem (CSP) approach.
2. Write a program to solve the Graph Coloring Problem using an appropriate algorithm.
3. Write a program to solve the Eight Queens Problem using the Backtracking Algorithm.
4. Write a program to demonstrate Constraint Propagation in a Constraint Satisfaction Problem.
5. Write a program to implement Forward Checking in a Constraint Satisfaction Problem.
6. Write a program to analyze the performance of the Backtracking Algorithm on a constraint satisfaction problem such as N-Queens or Sudoku.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 4 : Game Playing Algorithms for Decision Making (Total slots=2)

Objective

- To know about the game playing algorithms used for decision making in artificial intelligence.
- To understand the way in which intelligent agents make an optimum move in their environment.
- To understand the working principle of the Minimax Algorithm, Alpha Beta Pruning, and Evaluation Function in game playing.

Ready

- The concept of games in artificial intelligence comprises making decisions based on a situation wherein various parties are competing against each other in order to accomplish their goals.
- In gaming with artificial intelligence, the future moves are considered, and an action is chosen which is appropriate at that point of time.
- Algorithms such as Minimax and Alpha-Beta pruning help to make the best decisions with minimum computations involved.

Ready Reference

A Game Playing Algorithm is an algorithm for searching through game state space and making the right choice of moves by taking into account future possibilities and reactions from other players. The goal of the algorithm is to always make the best decision.

Game Playing Algorithms are the Artificial Intelligence methods applied in order to make intelligent decisions in competitive situations involving multiple players who alternate between turns. Game playing algorithms analyze potential actions that might be performed in the future and select the action which has the highest probability of success.

Elements of a Game

- Players: Players of the game (for example, Player 1 and Player 2).
- State of the Game: The present state of the game.
- Moves: Actions that can be taken by the player.
- Endstate of the Game: Final state of the game (either winning, losing, or drawing).
- Utility Function: Numerical value assigned to the end result (e.g., winning: +1, drawing: 0, losing: -1).

Minimax Algorithm

The Minimax Algorithm is an AI decision-making algorithm employed in playing games with two players, such as Chess, Tic-Tac-Toe, and Checkers.

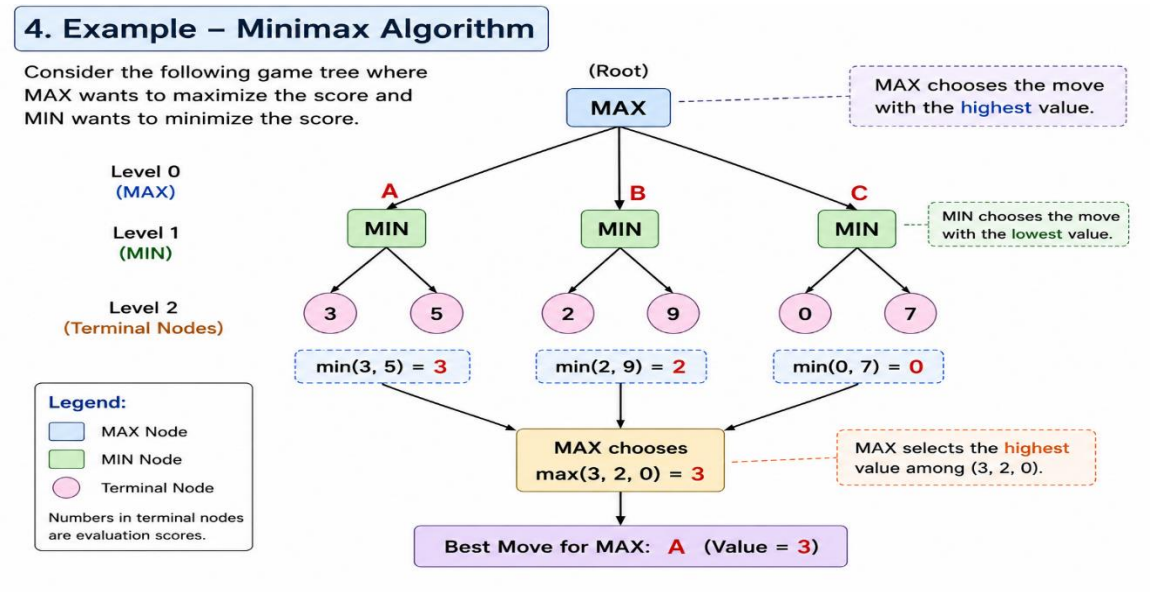
The player can make the optimal move under the condition that his/her opponent will play optimally as well.

- MAX Player: Maximizes the score.
- MIN Player: Minimizes the score.

This algorithm evaluates all future possibilities and makes a choice that gives the best guarantee of success.

Pseudocode

```
Algorithm Minimax(node, depth, maximizingPlayer)

Input:
   node – current game state
   depth – search depth
   maximizingPlayer – True for MAX, False for MIN

1. If node is a terminal state or depth = 0:
      Return the evaluation value of node.

2. If maximizingPlayer is True:
      bestValue = -∞
      For each child of node:
         value = Minimax(child, depth - 1, False)
         bestValue = max(bestValue, value)
      Return bestValue

3. Else (MIN player's turn):
      bestValue = +∞
      For each child of node:
         value = Minimax(child, depth - 1, True)
         bestValue = min(bestValue, value)
      Return bestValue
```



The Minimax Algorithm is an important search algorithm in Artificial Intelligence used in two-person games. Through this approach where one player seeks to maximize his scores and the other player seeks to minimize them, the best move is determined. The algorithm offers the best solutions in many strategy games and is the basis of other more sophisticated algorithms like Alpha-Beta Pruning.

Alpha-Beta Pruning

Alpha-Beta Pruning is a method of optimizing the Minimax Algorithm in Artificial Intelligence. It works on minimizing the number of nodes that are to be evaluated by pruning those nodes which have no impact on the decision-making process.

The algorithm makes use of two parameters:

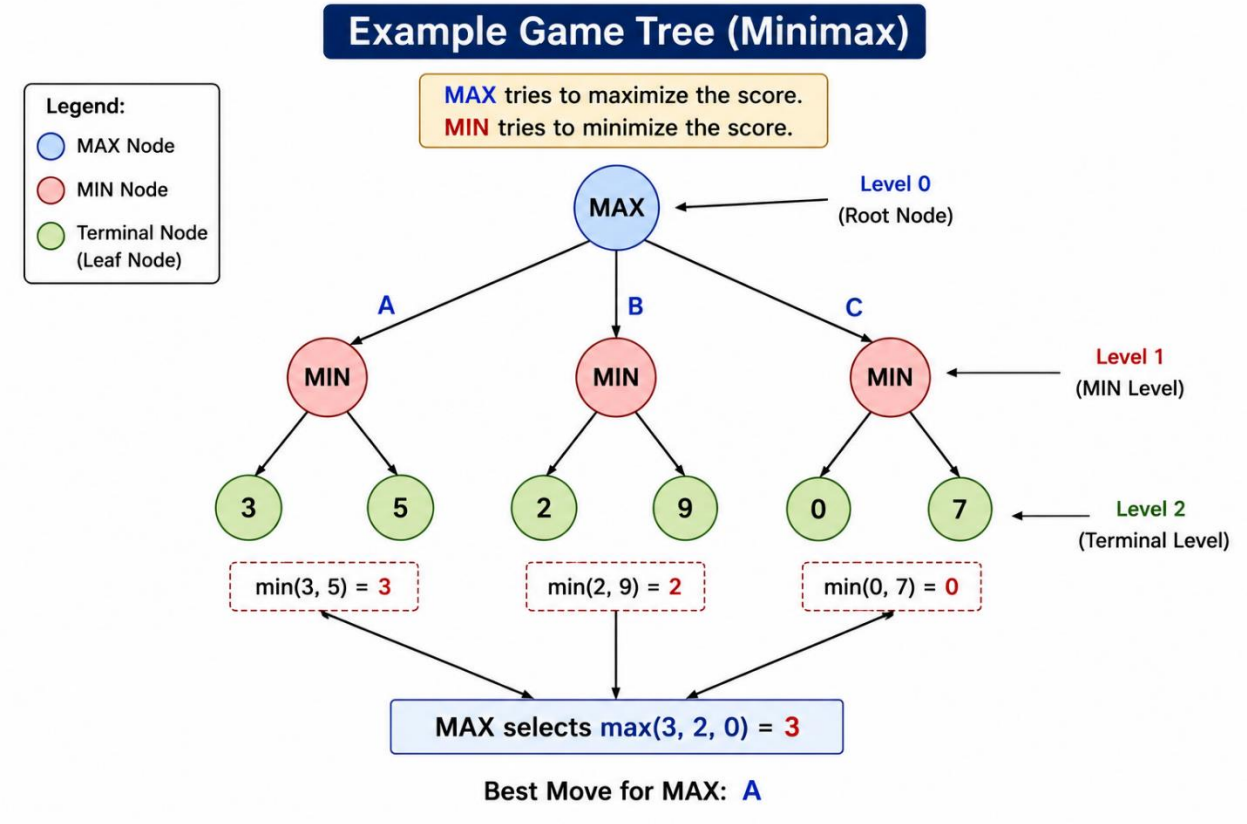
- Alpha (α): The best (highest) value of what the MAX player can ensure so far.
- Beta (β): The best (lowest) value of what the MIN player can ensure so far.

When at any point of time $\alpha \geq \beta$, the nodes that come afterwards are ignored.

Pseudocode

```
Algorithm AlphaBeta(node, depth, alpha, beta, maximizingPlayer)

1. If node is a terminal state or depth = 0:
      Return evaluation(node)
2. If maximizingPlayer:
      value = -∞
      For each child of node:
         value = max(value,
                  AlphaBeta(child, depth-1,
                        alpha, beta, False))
```

```
        alpha = max(alpha, value)
        If alpha >= beta:
            Break   // Beta Cutoff
    Return value
3. Else (MIN player):
    value = +∞
    For each child of node:
        value = min(value,
                AlphaBeta(child, depth-1,
                        alpha, beta, True))
        beta = min(beta, value)
        If alpha >= beta:
            Break   // Alpha Cutoff
    Return value
```



Alpha-Beta Pruning is an improved version of the Minimax Algorithm in which efficiency is increased due to the process of pruning out branches that will not affect the ultimate choice. In this technique, keeping alpha and beta values helps in saving effort without compromising the best move and therefore is highly popular in gaming AI algorithms.

Game AI (Chess, Tic-Tac-Toe)

AI in games means the Artificial Intelligence methods applied to making decisions in games by assessing different possibilities and picking the most appropriate one. Games such as Chess and Tic-Tac-Toe use search algorithms including Minimax and Alpha-Beta Pruning in order to play smartly against the human being.

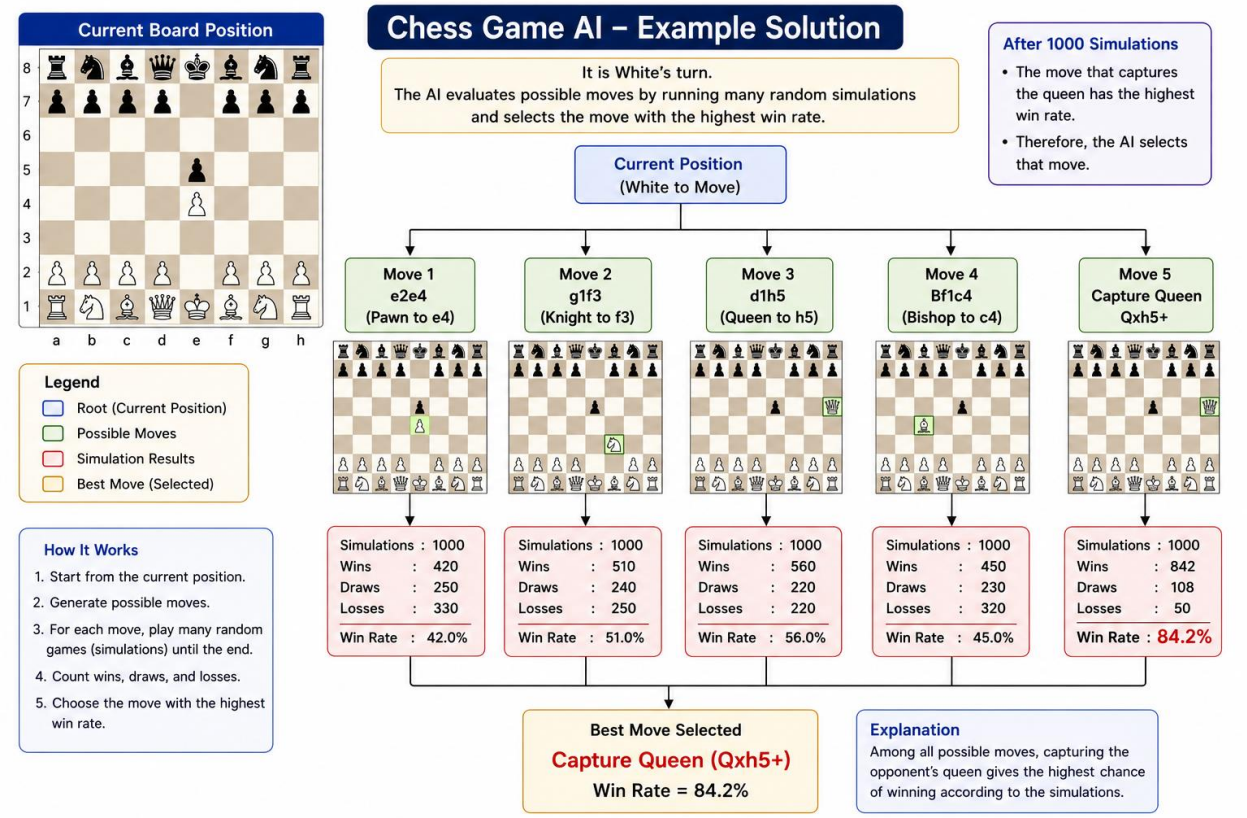
The primary aim of Game AI is to increase the chances of winning for the player while reducing the chances of the opponent.

Objectives

- Make intelligent decisions during gameplay.
- Predict future possible moves of the opponent.
- Choose the most suitable move after assessment.
- Achieve victory or at least a draw always.

Pseudocode

```
Algorithm GameAI(state, maximizingPlayer)

Input:
   Current game state

Step 1: Check whether the game is over.
Step 2: If terminal state, return its score
       (Win = +1, Draw = 0, Loss = -1).
Step 3: Generate all possible legal moves.
Step 4: If maximizing player:
          Choose the move with the highest score.
Step 5: If minimizing player:
          Choose the move with the lowest score.
Step 6: Repeat recursively for future moves.
Step 7: Return the best move.

Output:
   Best move for the current player.
```



Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is a heuristic search technique that can be applied in Artificial Intelligence for making decisions in games and planning tasks. The approach does not examine all possible actions but uses the outcomes of many random experiments (playouts) to choose the most promising action.

The technique is very popular in games where the number of actions is huge, such as Go, Chess, and other strategic board games.

Objectives

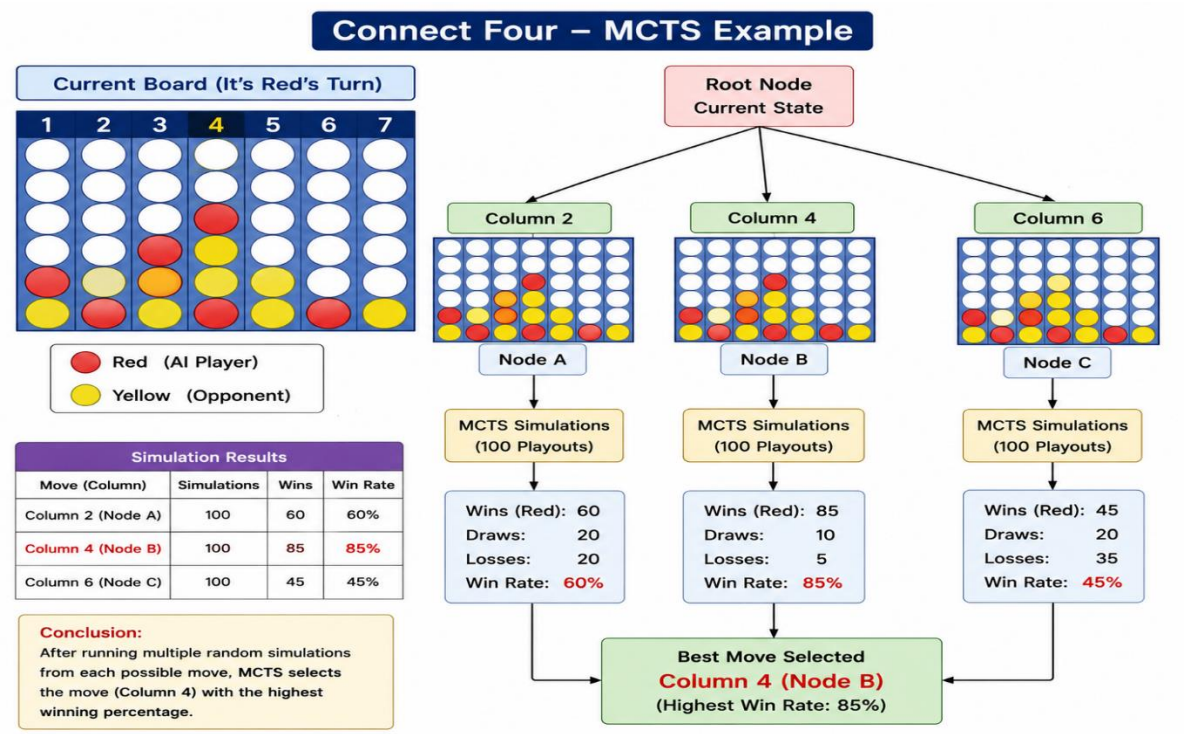
- Choose the optimal move through simulation of many possible outcomes.
- Achieve an adequate balance between the exploration and exploitation in choosing the move.
- Make better decisions in complex games.
- Search efficiently a huge game tree.

Steps of MCTS

- Selection: From the root node onwards, select child nodes till you get a promising node.
- Expansion: Create a child node or more in the search tree.
- Simulation (Rollout): Randomly play the game from the newly created node till its end.
- Backpropagation: Propagate the results to all the nodes visited during the process.

Pseudocode

```
Algorithm MonteCarloTreeSearch(root)

Input:
   Current game state (root)

Step 1: While computation time remains:
          a. Select the best node from the tree.
          b. Expand the selected node by adding a child.
          c. Simulate a random game from the new node.
          d. Backpropagate the simulation result to update scores.

Step 2: Choose the child of the root node
        with the highest visit count or best score.

Output:
   Best move for the current player.
```



MCTS is a sophisticated algorithm that helps computers make choices through simulations of possible outcomes in future games. This algorithm goes through four main phases, namely Selection, Expansion, Simulation, and Backpropagation, in order to discover good moves without going through all the possibilities in the game tree. MCTS is very popular in today's AI for games due to its scalability and the quality of the moves produced.

Program for Minimax Algorithm

```
#Program for Minimax Algorithm
import math
# Recursive Minimax function
def minimax(depth, node_index, is_max, values, max_depth):
    # If leaf node is reached
    if depth == max_depth:
        return values[node_index]
    if is_max:
        # MAX player's turn
        left = minimax(depth + 1, node_index * 2, False, values, max_depth)
        right = minimax(depth + 1, node_index * 2 + 1, False, values, max_depth)
        return max(left, right)
    else:
        # MIN player's turn
        left = minimax(depth + 1, node_index * 2, True, values, max_depth)
        right = minimax(depth + 1, node_index * 2 + 1, True, values, max_depth)
        return min(left, right)
# ---------------- Main Program ----------------
# Leaf node values of the game tree
leaf_values = [3, 5, 2, 9, 12, 5, 23, 23]
# Height of tree (3 levels)
tree_depth = int(math.log2(len(leaf_values)))
# Find the optimal value
best_score = minimax(
    depth=0,
    node_index=0,
    is_max=True,
    values=leaf_values,
    max_depth=tree_depth
)
print("Leaf Node Values:", leaf_values)
print("Optimal value selected by Minimax:", best_score)

/*OUTPUT
```

```
Leaf Node Values: [3, 5, 2, 9, 12, 5, 23, 23]
Optimal value selected by Minimax: 12*/
```

Program of Chess Game AI using Minimax

```
# Program of Chess Game AI using Minimax

import math

# Minimax function
def minimax(depth, node_index, is_max, scores, max_depth):

    # If leaf node is reached, return its score
    if depth == max_depth:
        return scores[node_index]

    if is_max:
        return max(
            minimax(depth + 1, node_index * 2, False, scores, max_depth),
            minimax(depth + 1, node_index * 2 + 1, False, scores, max_depth)
        )
    else:
        return min(
            minimax(depth + 1, node_index * 2, True, scores, max_depth),
            minimax(depth + 1, node_index * 2 + 1, True, scores, max_depth)
        )

# ---------------- Main Program ----------------

# Evaluation scores of possible final positions
scores = [5, 7, 8, 11]

# Height of the game tree
tree_depth = int(math.log2(len(scores)))

# AI (MAX player) chooses the best move
best_score = minimax(0, 0, True, scores, tree_depth)

print("Evaluation scores:", scores)
print("Best score selected by Chess AI:", best_score)
/*OUTPUT
Evaluation scores: [5, 7, 8, 11]
Best score selected by Chess AI: 8*/
```

Lab Assignments

SET A

1. Write a program to implement the Minimax Algorithm for a two-player game.
2. Write a program to implement Alpha-Beta Pruning for optimizing the Minimax search process.
3. Write a program to develop a Tic-Tac-Toe Game in which the computer uses the Minimax Algorithm to play optimally.
4. Write a program or report that compares Minimax and Alpha-Beta Pruning on the same game tree.
5. Write a program to implement a Game Tree Representation for a two-player game.
6. Write a program or report to analyze game-playing strategies in Artificial Intelligence.

Signature of the Instructor | | **Date** | |

SET B

1. Write a program to implement the Monte Carlo Tree Search (MCTS) algorithm for decision-making in games.
2. Write a program to develop a simple Chess Move Evaluator using piece values and board positions.
3. Write a program to implement an AI player for the Connect Four game.
4. Write a program to analyze the effect of Alpha-Beta Pruning on game tree search performance.
5. Write a program to implement an AI agent for Tic-Tac-Toe.
6. Write a program or report to compare deterministic games and stochastic games in Artificial Intelligence.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 5: Knowledge Representation and Logical Reasoning

(Total slots = 1)

Objective

- To Understand the significance of Knowledge Representation in Artificial Intelligence Systems.
- To understand of various ways in which knowledge can be represented.
- To learn about logical reasoning methods employed for problem solving.

Ready

- Knowledge representation (KR) refers to the organization and storage of knowledge into a system whereby artificial intelligence can use it for reasoning and decision making.
- Logical reasoning refers to the capacity of artificial intelligence to generate new knowledge from existing knowledge through the use of logical processes.
- Knowledge representation mechanisms like semantic nets, frames, production rules, and predicate logic enable artificial intelligence to solve challenging tasks quickly.

Ready Reference

Knowledge Representation and Logical Reasoning (KRLR) is one of the essential domains of Artificial Intelligence (AI), which deals with representation of the real-world data into a machine-understandable format for decision making. It allows artificial intelligence systems to store information, make inferences, problem-solving, and answering questions using intelligence.

Knowledge Representation is the process of structuring and storing knowledge in a way that makes it processable by machines.

Logical Reasoning is the application of logical inference to previously known data to make inferences or decisions.

Objectives

- To model real world knowledge in a computer-readable format.
- To make intelligent decisions.
- To perform automatic reasoning.
- To achieve consistency and correctness in AI systems.
- To share knowledge and reuse.

Components of Knowledge Representation

- Facts: Assertions that provide information about the known facts.
- Objects: Objects in the real world.

- Relationships: Relations between objects.
- Rules: Rules for performing actions or conclusions.
- Inference Procedure: The mechanism for deriving knowledge.

Types of Knowledge

- Declarative Knowledge: Information
  Example: "Paris is the capital of France."
- Procedural Knowledge: Information about procedures
  Example: Sorting algorithm
- Heuristic Knowledge: Information based on experience
  Example: "Pick the shortest route available"
- Meta-Knowledge: Knowledge about knowledge
  Example: Information about the organization of the database.
- Structural Knowledge: Information about relationships between concepts
  Example: "A car contains an engine and wheels."

Knowledge Representation and Logical Reasoning (KRLR) is used to establish intelligent behaviour in AI applications through organization of information in a structured manner and then using logic to draw conclusions from the data. It is used in many fields like health care, business, education, and robots to draw conclusions and help in making decisions and solving different kinds of problems.

Propositional Logic and Predicate Logic

Propositional Logic

Propositional Logic (also known as Statement Logic) is a form of logic in which statements are expressed in propositions that can be true or false (T or F).

Features of Propositional Logic

- It deals with complete statements or propositions.
- Each proposition has a single truth value, either True or False.
- It does not show the structure of the statement.
- It is used in rule-based systems, digital circuits, and AI.

Logical Operators

Operator	Symbol	Meaning	Example
AND	$\wedge$	Both statements must be true	$P \wedge Q$
OR	$\vee$	At least one statement is true	$P \vee Q$
NOT	$\neg$	Negates the statement	$\neg P$
Implication	$\rightarrow$	If P then Q	$P \rightarrow Q$
Biconditional	$\leftrightarrow$	P if and only if Q	$P \leftrightarrow Q$

Example

P: It is raining.

Q: The roads are wet.

Rule:

$P \rightarrow Q$

Meaning:

If it is raining, then the roads are wet.

If P = True, then Q is expected to be True.

Predicate Logic

Predicate Logic (or First Order Logic) is an extension of propositional logic that uses objects, predicates, variables, and quantifiers.

Components

- Predicate: Specifies properties or relations.
  Example: Student(Rahul)
- Variable: Symbolizes an object.
  Example: x, y
- Constant: Represents an object or individual.
  Example: Rahul, India
- Quantifiers
  - Universal (∀): All
  - Existential (∃): There Exists

Example

All humans are mortal.

Predicate Logic:

$\forall x \text{ Human}(x) \rightarrow \text{Mortal}(x)$

Fact

Human(Ram)

Example

Every student who studies passes the exam.

Predicate Logic:

$$\forall x\ [\text{Student}(x) \land \text{Studies}(x)] \rightarrow \text{Pass}(x)$$

Facts:

Student(Isha)
Studies(Isha)

Example

Universal Quantifier (∀)

Statement:

All birds can fly.

$$\forall x\ \text{Bird}(x) \rightarrow \text{CanFly}(x)$$

Existential Quantifier (∃)

Statement:

There exists a student who scored 80 marks.

$$\text{`}\exists x\ \text{Student}(x) \land \text{Scored80}(x)$$

Propositional Logic can be used to represent simple statements that are either true or false and their combination using logical connectives. But Predicate Logic is a step ahead, as it represents objects, attributes, relations, and even quantifiers, thus becoming much better suited for knowledge representation in AI.

Rule-Based Systems

Rule Based Systems (RBS) refers to an AI system that makes decisions using a series of predefined IF-THEN rules applied to known facts or information. The process of solving problems in this system depends on logical inference.

Components of Rule-Based Systems

- Knowledge Base – Contains facts and rules.
- Inference Engine – Applies the rules to derive conclusions.
- Working Memory – Maintains facts and data.
- User Interface – Provides for user interaction.

Key Elements

1. Knowledge Base

Stores domain knowledge in form of IF-THEN statements.

Example,

IF temperature > 38°C THEN the person has a fever.

IF the person has fever AND cough THEN suspect influenza.

2. Inference Engine

Performs processing of rules and facts in order to draw conclusions.

3. Facts

Facts can be either data available to the system or supplied by the user through sensor.

Example,

Temperature is 39°C

The person has a cough

4. Rule Set

Logical conditions used for making decisions.

Process Involved

- Accept input facts from the user.
- Match facts against the rules stored in knowledge base.
- Use those IF-THEN rules that match.
- Draw new facts.
- Output result.

Algorithm

Step 1: Start
Step 2: Read input facts.
Step 3: Load IF–THEN rules.
Step 4: Compare facts with rule conditions.
Step 5: If a rule matches, execute its THEN action.
Step 6: Continue until no more rules can be applied.
Step 7: Display the conclusion.
Step 8: Stop

Example

Medical Diagnosis : A patient visits a hospital with a temperature of 39°C and a cough.

Rules

- If Temperature > 38°C then Fever = Yes
- If Fever = Yes and Cough = Yes then Disease = Flu
- If Disease = Flu then Recommend = Rest and medication

Inputs

Temperature = 39°C

Cough = Yes

Reasoning

- If Temperature > 38°C then Fever = Yes
- If Fever = Yes and Cough = Yes then Disease = Flu
- If Disease = Flu then Recommend = Rest and medication

Outputs

- Diagnosis: Flu
- Recommendation: Rest, hydrate yourself, and take medicine as suggested by your doctor.

The Rule-Based system refers to the AI technique, which utilizes the pre-defined set of IF…THEN rules to solve the problem and take the right decision. This approach is quite easy, clear, and efficient enough for the domains wherein the expert knowledge is presented in the form of logical rules.

Forward Chaining

Forward Chaining is a rule-based reasoning method used in artificial intelligence where the reasoning process starts with facts and applies IF–THEN rules to generate new facts in order to reach a particular goal or conclusion. It is also known as data-driven reasoning as it starts from known data and proceeds further towards the goal.

Characteristics of Forward Chaining

- Begins with the known facts/observations.
- Applies matching IF–THEN rules step-by-step.
- Derives new facts from known facts.
- Keeps doing this until the desired conclusion is reached or there are no more rules left.
- Commonly used in expert systems, diagnosis, planning, and decision-making systems**.**

Algorithm

Step 1: Start.

Step 2: Store the known facts in the working memory.

Step 3: Compare the facts with the IF conditions of all rules.

Step 4: If a rule matches, execute its THEN part and add the new fact.

Step 5: Repeat the process with the updated facts.

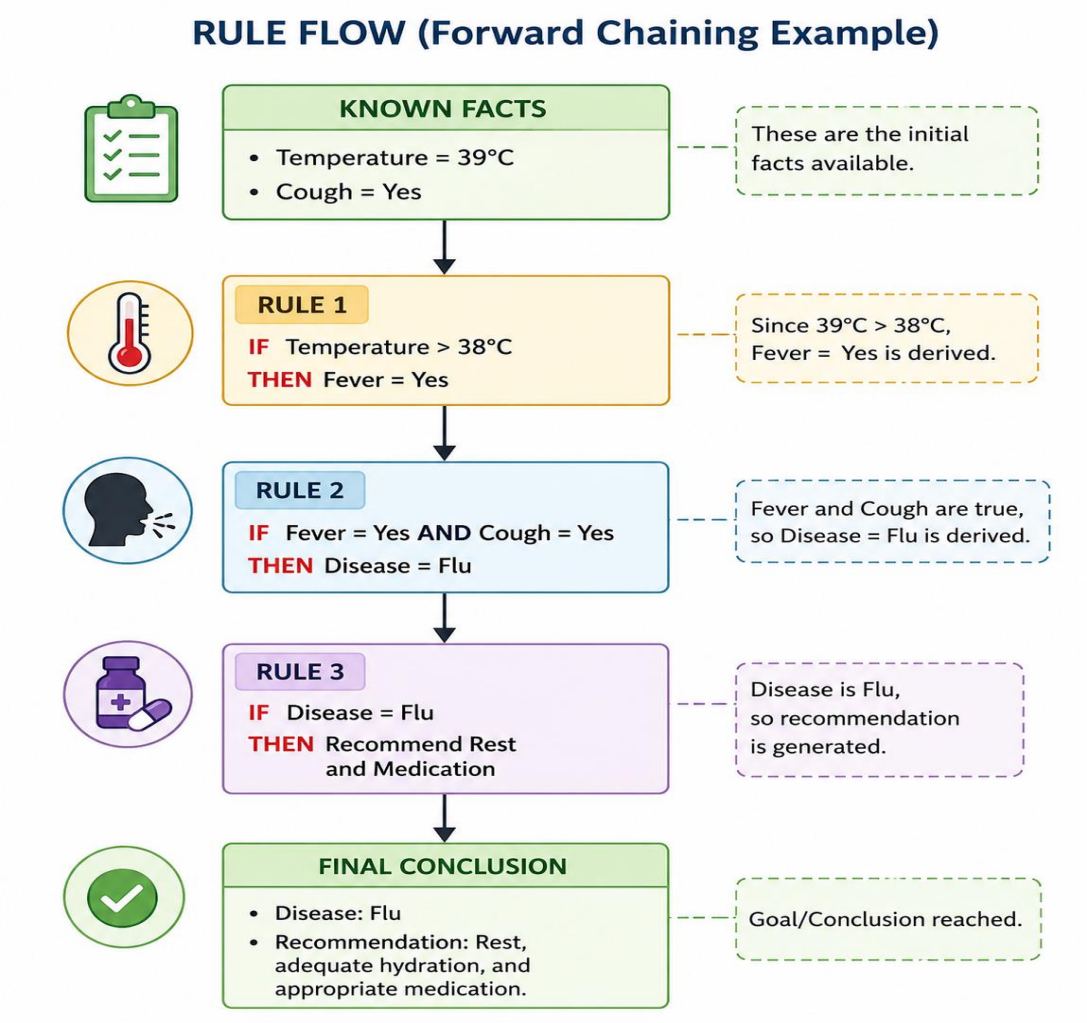
Step 6: Stop when the goal is reached or no more rules can be fired**.**

Example

A medical expert system with the following rules:

- Rule 1: IF the temperature of a patient exceeds 38°C, THEN the patient suffers from fever.
- Rule 2: IF the patient suffers from a fever and cough, THEN the patient may suffer from flu.
- Rule 3: IF the patient may suffer from flu, THEN rest and proper medication are recommended for him/her.

Consider that a patient has arrived at the clinic with a temperature of 39°C and a cough. The system will conclude at first that the patient has a fever since his/her temperature is more than 38°C. Using the derived information along with the cough, the system will further conclude that the patient may be suffering from flu. Finally, it will recommend proper rest and medication to the patient. This is an example of forward chaining.



Forward Chaining is a rule-based reasoning approach that involves deriving new knowledge from existing information using rules of logic. This form of reasoning starts from factual information and derives further facts until a certain conclusion is drawn. Backward Chaining

Backward Chaining

Backward Chaining is a method of reasoning in Artificial Intelligence, where one starts with a goal or a conclusion and moves backwards in order to test whether the available facts satisfy the goal or not. Backward Chaining is also known as goal-driven reasoning since it is based on starting with a goal and finding the facts to support it.

Characteristics of Backward Chaining

- Starts with a goal or a hypothesis.
- Moves backwards by testing which rules can lead to the goal.
- Checks whether the preconditions (facts) are true or not.
- Halts either by reaching the goal or finding that no facts exist for it.
- Popular application areas include expert systems, diagnosis and querying applications.

Algorithm

Step 1: Start.

Step 2: Define the goal to be proved.

Step 3: Find a rule whose THEN part matches the goal.

Step 4: Check whether the IF conditions of that rule are true.

Step 5: If a condition is not known, treat it as a new sub-goal.

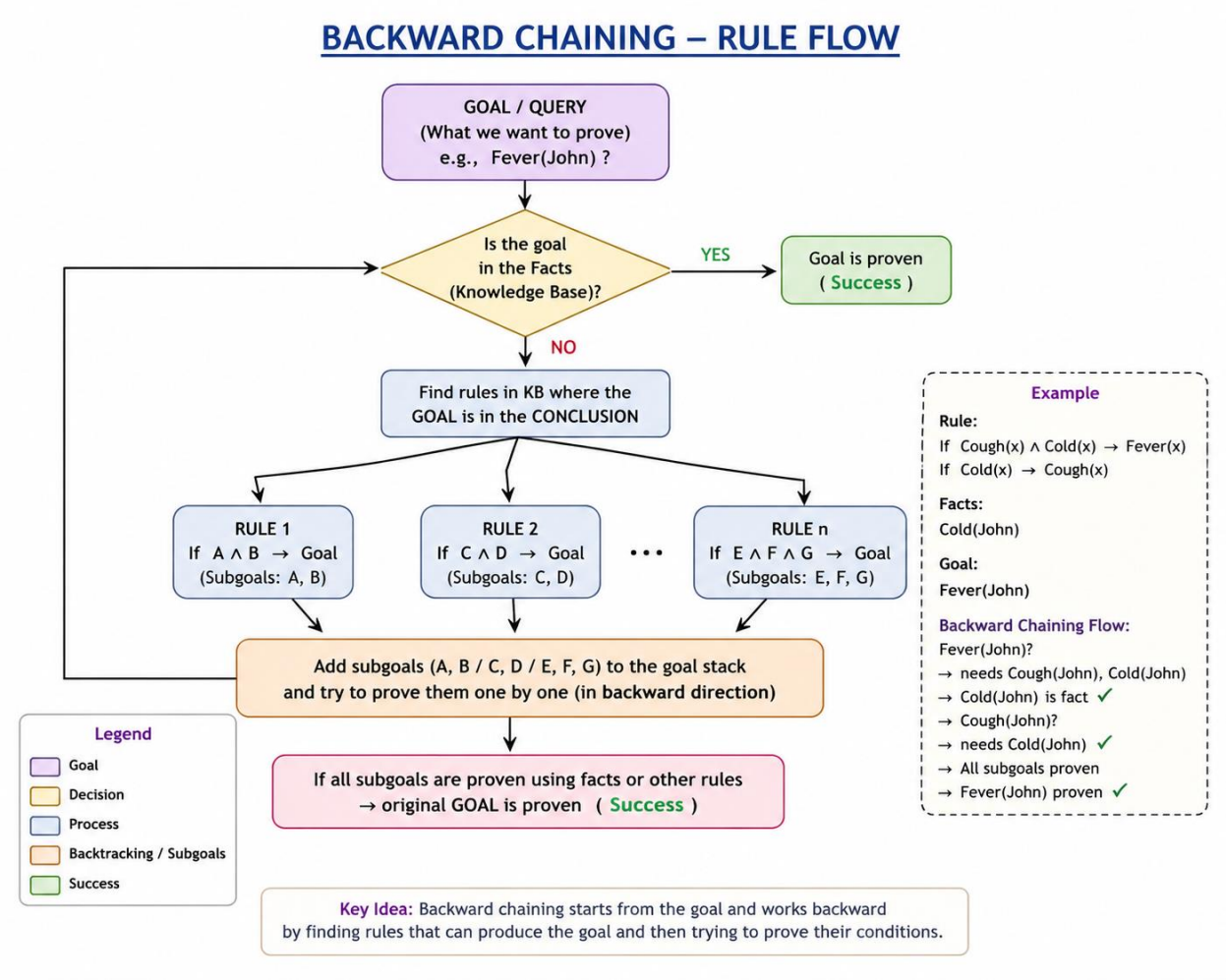
Step 6: Repeat the process until all conditions are satisfied or fail.

Step 7: If all required facts are true, prove the goal.

Step 8: Stop.

Example Rules in a medical expert system are as follows:

A medical expert system, one would like to see if a patient suffers from the flu. In this case, the aim would be “Patient has flu.” The system seeks a rule according to which, if the patient has a fever and a cough, then the patient has the flu. Then, the system seeks if the patient has a fever and a cough. Indeed, the patient has a fever since his/her body temperature is 39°C, and the patient has a cough. As a result, the system proves the aim that the patient has the flu.



Backward Chaining is a form of deduction process which begins from the desired result and goes back through the facts and rules to prove whether those can provide the necessary result. The Backward Chaining is especially helpful in those cases where one needs to check some hypothesis.

Ontological Engineering and Knowledge Graphs

Ontological Engineering refers to the design, creation, and maintenance of an ontology that formally defines concepts and relations in a given domain.

Ontology is a formalized knowledge representation that consists of:

- Classes (concepts)
- Attributes (properties)
- Conceptual relationships
- Constraints and rules

Key point: It serves as a common representation for the domain that machines can understand.

Steps in Ontological Engineering:

- Define scope – Domain choice (medical, educational, etc.)
- Identify concepts – Key entities (Student, Disease, etc.)
- Building hierarchy – Parent and child relationships of concepts
- Relationship specification – Relations such as "treats", "has symptom"
- Formalization – Formal representation using logic/OWL

Example:
In a medical ontology:

- Disease → Fever
- Symptom → High Temperature
- Relationship → Fever *hasSymptom* High Temperature

2. Knowledge Graphs

Knowledge Graphs are structured representations of real-world entities and their relationships in a graph format.

They store knowledge as:

- **Nodes (Entities):** Person, Place, Object
- **Edges (Relations):** "works_at", "located_in", "treats"
- **Triples:** (Subject, Predicate, Object)

Example of triple:

- (Rudra, hasDisease, Fever)

Connection between Ontology and Knowledge Graphs

- Ontology refers to the schema (structure and rules).
- Knowledge graph is where the data is stored based on that structure.
- Knowledge graphs get more meaning through ontology.

Example: Ontology Rule and Knowledge Graph

Ontology Rule:

If a person has both cough and cold, then they may have fever.

This rule can be represented as:

If:

- Person → hasSymptom → Cough

- Person → hasSymptom → Cold

Then:

- Person → mayHave → Fever

Knowledge Graph Facts:

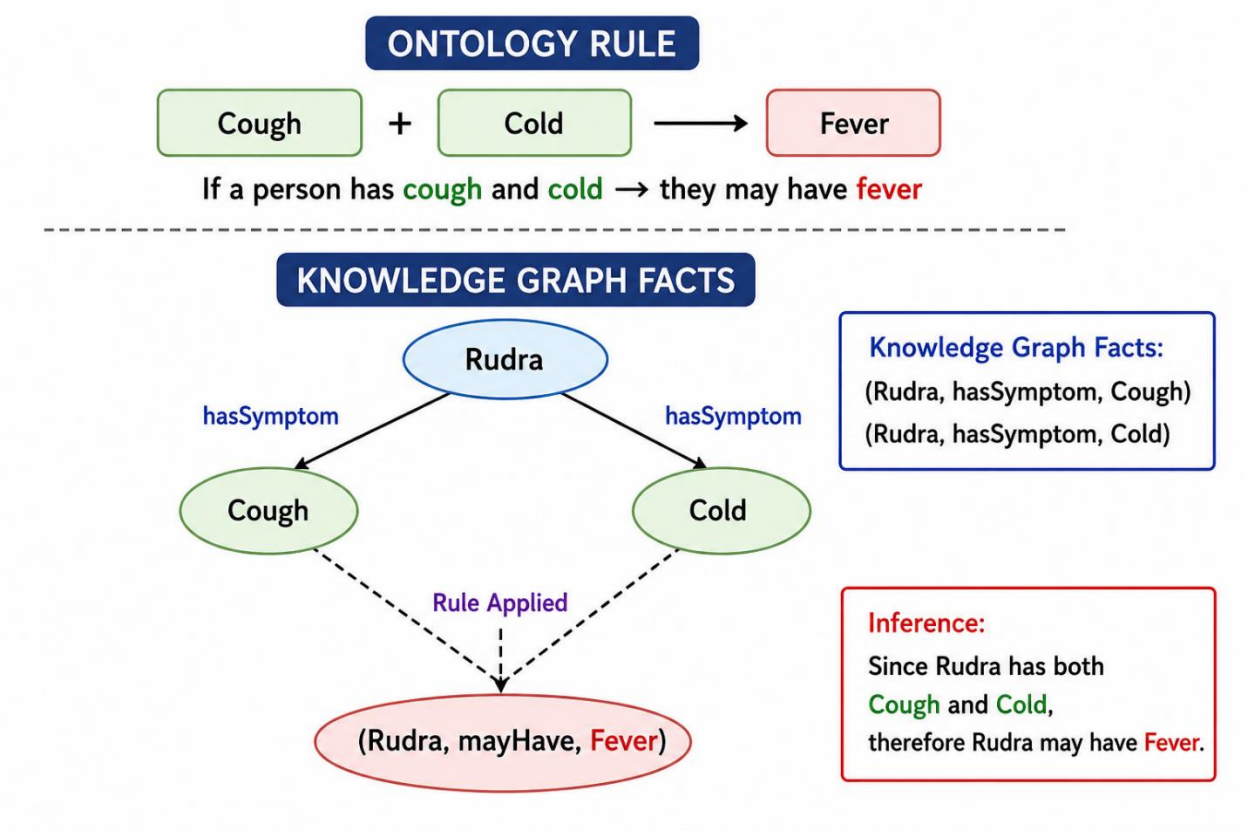
(Rudra, hasSymptom, Cough)

(Rudra, hasSymptom, Cold)

Inference:

Since the knowledge graph states that Rudra has both cough and cold, the ontology rule is applied. Therefore, the system can infer:

(Rudra, mayHave, Fever)



Program for Predicate Logic

```
# Program for Predicate Logic

# Facts
humans = ["Raha", "Atharva", "Pia"]

# Predicate function
def is_human(person):
    return person in humans

# Rule function
def is_mortal(person):
    return is_human(person)

# Test with Atharva
name = "Atharva"

print("Person:", name)

if is_human(name):
    print(name, "is a Human.")

if is_mortal(name):
    print(name, "is Mortal.")
else:
    print(name, "is not found in the human list.")

/*OUTPUT
Person: Atharva
Atharva is a Human.
Atharva is Mortal. */
```

Program of Rule-Based Medical Diagnosis

```
# Program of Rule-Based Medical Diagnosis

# Input from user
temperature = float(input("Enter body temperature (°C): "))
cough = input("Does the patient have a cough? (yes/no): ").lower()

# Rule 1: Check for fever
if temperature > 38:
    fever = True
else:
    fever = False
```

```
# Rule 2 and Rule 3: Diagnose and recommend
if fever and cough == "yes":
    disease = "Flu"
    recommendation = "Take rest, drink plenty of fluids, and consult a doctor."
elif fever:
    disease = "Fever"
    recommendation = "Monitor symptoms and consult a doctor if needed."
else:
    disease = "No major illness detected"
    recommendation = "Maintain a healthy lifestyle."

/*Output
print("\n--- Diagnosis Report ---")
print("Fever:", fever)
print("Diagnosis:", disease)
print("Recommendation:", recommendation)*/
```

Program for Knowledge Graph Triples

```
# Program for Knowledge Graph Triples
facts = [
    ("Rudra", "hasSymptom", "Cough"),
    ("Rudra", "hasSymptom", "Cold")
]

# Extract symptoms
person = "Rudra"
symptoms = []

for subject, relation, obj in facts:
    if subject == person and relation == "hasSymptom":
        symptoms.append(obj)

# Apply ontology rule
if "Cough" in symptoms and "Cold" in symptoms:
    print((person, "mayHave", "Fever"))
else:
    print("Rule not satisfied.")
/*OUTPUT
 ('Rudra', 'mayHave', 'Fever') */
```

Lab Assignments

SET A

1. Write a program to implement Propositional Logic and evaluate logical expressions using operators such as AND, OR, NOT, IMPLIES, and XOR.
2. Write a program to implement Predicate Logic using predicates, variables, and quantifiers.
3. Write a program to design a Rule-Based Expert System for simple decision-making.
4. Write a program to implement the Forward Chaining inference mechanism.
5. Write a program to implement the Backward Chaining inference mechanism.
6. Write a program or report to compare Forward Chaining and Backward Chaining inference techniques.

Signature of the Instructor | | **Date** | |

SET B

1. Write a program to build a Knowledge Base using logical rules and facts.
2. Write a program to design a Rule-Based Expert System for basic medical diagnosis.
3. Write a program to implement inference using Predicate Logic.
4. Write a program to develop a simple Knowledge Graph representing relationships among entities.
5. Write a program to demonstrate basic Ontological Engineering concepts using classes, subclasses, properties, and relationships.
6. Write a program to implement reasoning using logical expressions.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 6 : Basics of Machine Learning (Total slots = 1)

Objective

- To understand the basic concepts and applications of Machine Learning.
- To learn about various Machine Learning algorithms and their working principles.
- To study how machines learn from data and make predictions.

Ready

- Machine learning (ML) is a field of artificial intelligence which allows computers to gain information through data and learn from it without being programmed.
- The algorithm of machine learning recognizes patterns in data and uses them to generate decisions.

Ready Reference

Machine learning (ML) is an area of artificial intelligence (AI) where computer algorithms are made to learn from data without being explicitly programmed to perform a specific task. ML algorithms do not rely on any fixed set of instructions but rather learn from the examples and experience.

Key Concepts

- Machine learning works on data for training the model.
- Model refers to the mathematical formulation which learns from data.
- Features refer to the variables used for training purposes.
- Targets or labels refer to the output in supervised machine learning**.**

Machine Learning Types and Learning Algorithms

Machine Learning Types

1. Supervised Learning

In supervised learning, the learning process occurs using labeled input-output data in which the output values are known.

Example, Spam detection using previously classified emails.

2. Unsupervised Learning

In unsupervised learning, the algorithm identifies hidden structure or grouping in input data that does not have any labels.

Example, Customer segmentation based on their buying habits.

3. Reinforcement Learning

In reinforcement learning, an agent learns using feedback received in response to actions taken in an environment.

Example, Training a robot to navigate through a maze by rewarding it whenever it reaches the target destination.

Machine Learning Process Steps

- Gathering of data.
- Preprocessing and cleaning of data.
- Feature selection.
- Building the machine learning algorithm.
- Evaluation of the model.
- Deployment of the model for prediction and decisions.

Supervised

Supervised Learning is a form of machine learning where a model is trained on data that has been labeled. In each training instance, there is both an input and its output (label). The algorithm then learns the mapping between the inputs and the outputs such that it may predict the output of unseen instances. The aim of supervised learning is to reduce errors by learning from the known instances.

Working Process

- Gather labeled training data.
- Clean and preprocess the data.
- Train the machine learning algorithm.
- Test the algorithm with new data.
- Deploy the trained model for making predictions.

Example: Email Spam Detection

Suppose we have the following training data:

Email Subject	**Label**
"Win a free lottery prize"	Spam
"Meeting at 10 AM tomorrow"	Not Spam
"Claim your free gift now"	Spam
"Project report attached"	Not Spam

The supervised learning model is then given the following unseen email after being trained with the above examples:

"Congratulations! Collect your free gift."

Based on its understanding acquired through training, it makes a prediction as follows:
Prediction: Spam

Types of Supervised Learning

1. **Classification** – Predicts categories or labels.
    - Example: Spam vs. Not Spam, Disease vs. Healthy.
2. **Regression** – Predicts continuous numerical values.
    - Example: House price prediction, Temperature forecasting.

Bayes' Theorem, Gaussian and Bernoulli Naïve Bayes,

Bayes' theorem is a formula based on probabilities that helps in predicting the probability of an event using updated information about previous knowledge. It is the basis for the Naïve Bayes classifier.

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Where:

- **P(A|B)** = Probability of event A given event B
- **P(B|A)** = Probability of event B given event A
- **P(A)** = Prior probability of A
- **P(B)** = Probability of B

Example

Assumptions:

10% of e-mails are spam.

80% of spam e-mails will have "Lottery" in them.

Only 5% of legitimate e-mails have "Lottery" in them.

Using Bayes' Theorem, if an e-mail has "Lottery" in it, we could determine the probability that it is spam.

AdaBoost (Boosting)

The AdaBoost (Adaptive Boosting) algorithm integrates a set of weak learners into a more powerful classifier. The algorithm trains learners iteratively, assigning higher weightage to those samples which have been incorrectly classified previously.

Steps Involved

- Train a weak learner.
- Find out the incorrectly classified samples.
- Give them more weightage.
- Train another learner.
- Integrate all learners to make a prediction.

Example

Let's assume that there are 100 loan applications, which have been labeled:

The first model is able to predict 80 accurately.

The other 20 are given greater weights.

The second model then works on those hard to classify.

Multiple Models (Voting)

In voting, multiple classifiers predict and a final output decision is made through majority voting or probability averages.

Types

- Hard Voting: Makes decision based on the majority prediction.
- Soft Voting: Uses predictions and decides on the class with the highest probability.

Example: Weather Prediction

Three classifiers predict whether it will rain tomorrow:

Model	Prediction
Random Forest	Rain
Decision Tree	Rain
Support Vector Machine (SVM)	No Rain

Rain votes: 2

No Rain votes: 1

Final Prediction: Rain (chosen by majority vote).

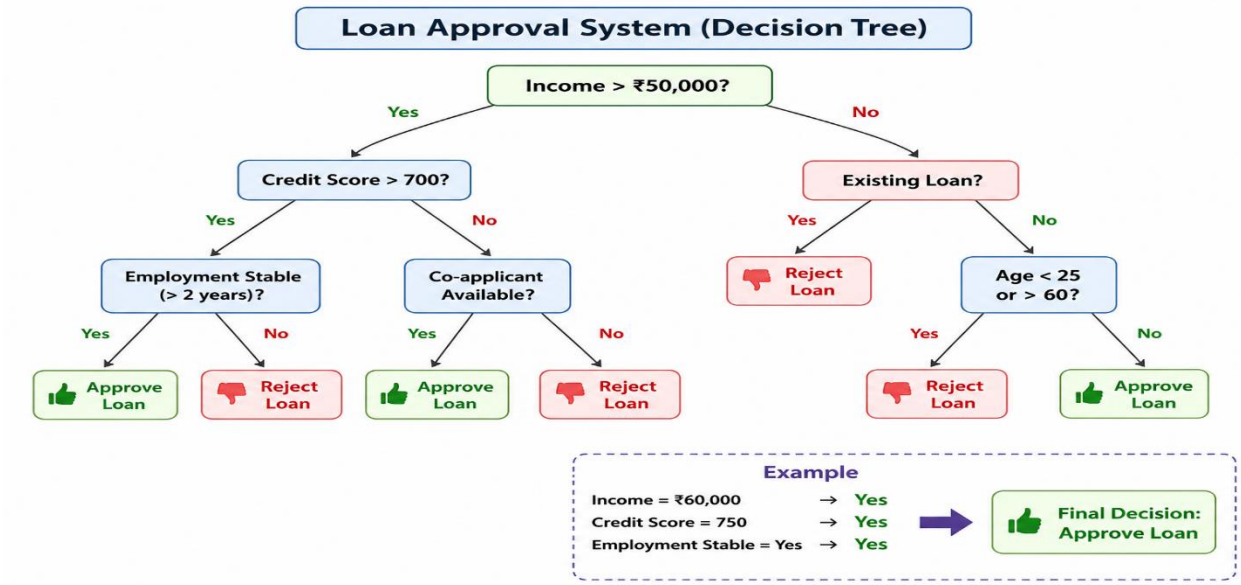
Decision Trees

Decision Tree is a supervised machine learning method that can be employed for both classification and regression problems. It is the process of representing decisions as a tree structure where internal nodes represent conditions applied to an attribute while branches represent outcomes of these decisions and leaf nodes represent decisions themselves.

The decision tree model splits data into smaller and smaller subsets depending on attribute values until it makes a decision.

Basic Components

- Root Node: Starting point of a tree.
- Decision Node: Represents a condition applied to an attribute.
- Branch: Represents outcome of a decision.
- Leaf Node: Represents decision itself.



Random Forest (Bagging)

Random Forest is an ensemble approach which uses many decision trees on different random samples of data.

Steps Involved

- Generate multiple bootstrap samples from the training data set.
- Train one decision tree for each of these samples.
- Each tree operates independently of other trees.
- Prediction is done using majority voting (for classification problem) or averaging (for regression problem).

Example : Problem: Loan Approval Prediction
A bank wants to decide whether a customer's loan application should be Approved or Rejected.

Instead of using a single decision tree, it uses a Random Forest, which consists of several decision trees trained on different random subsets of the data.

Input Data

Applicant	Income	Credit Score	Existing Loan	Actual Decision
A	High	Good	No	Approve
B	Low	Poor	Yes	Reject
C	Medium	Good	No	Approve

For a new applicant with:

- **Income:** High
- **Credit Score:** Good
- **Existing Loan:** No

Each tree makes its own prediction.

Predictions from Individual Trees

Decision Tree	Prediction
Tree 1	Approve
Tree 2	Approve
Tree 3	Reject
Tree 4	Approve
Tree 5	Approve

Majority Voting

- **Approve:** 4 votes
- **Reject:** 1 vote

Final Prediction: Approve the Loan

Unsupervised

Unsupervised Learning is the type of machine learning in which the algorithm learns from unlabelled data. As opposed to supervised learning, no output labels are pre-defined for the algorithm. The algorithm finds out any inherent pattern, relationship or grouping present in the data on its own.

The primary aim of unsupervised learning is to find any underlying structure in the data.

Working Process

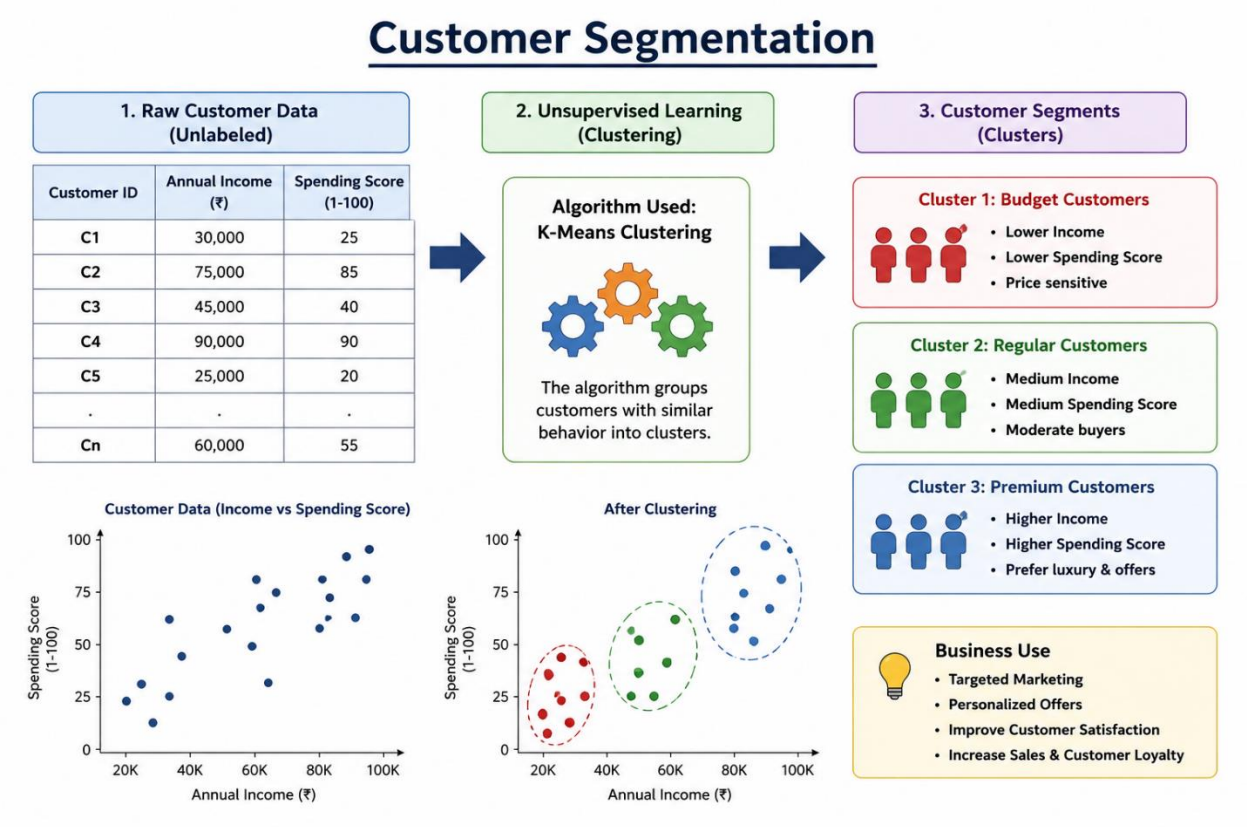
- Collect unlabelled data.
- Clean and preprocess the data.
- Use an unsupervised learning algorithm.
- The algorithm will find out any pattern or group in the data.
- Distance Metrics

Example: Customer Segmentation

A shopping mall collects information about its customers based on:

- Annual Income
- Spending Score

The data has no labels indicating customer type. An unsupervised learning algorithm groups customers automatically.



Objective Functions

An Objective Function is a mathematical function that represents the purpose of a machine learning model or optimization problem. The objective function evaluates the performance of the model and helps with the learning process by telling us whether we should minimize or maximize something.

- Minimization problems, an objective function is called a loss function or cost function.
- Maximization problems, it may be profit, accuracy, or reward.

The optimization algorithm changes the model parameters in order to optimize the objective function.

Simple Formula

For linear regression, a typical objective function is Mean Squared Error (MSE):

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- y_i = Actual value
- y_i^ = Predicted value
- n = Number of data points

The goal is to **minimize the MSE**.

Example : Student Marks Prediction

A machine learning model predicts exam marks.

Student	Actual Marks	Predicted Marks
Rahul	90	88
Dia	75	78
Prithvi	80	79

The objective function measures the difference between actual and predicted marks. During training, the algorithm updates its parameters to minimize this difference and improve prediction accuracy.

K-Means

K-Means is an unsupervised machine learning algorithm that is employed to classify data into K separate clusters based on similarities. In K-means clustering, each cluster has its centroid that is basically the mean of all the points in that cluster. It helps in classifying similar data points and separates different ones.

K-Means aims at reducing the distance between data points and centroid of the cluster assigned to them.

How K-Means Works

1. Fix the number of clusters (K).
2. Initialize K centroids randomly.
3. Allocate every data point to the closest centroid.
4. Calculate centroid of every cluster.
5. Steps 3 and 4 continue till the centroids do not change anymore or maximum iteration limit is reached.

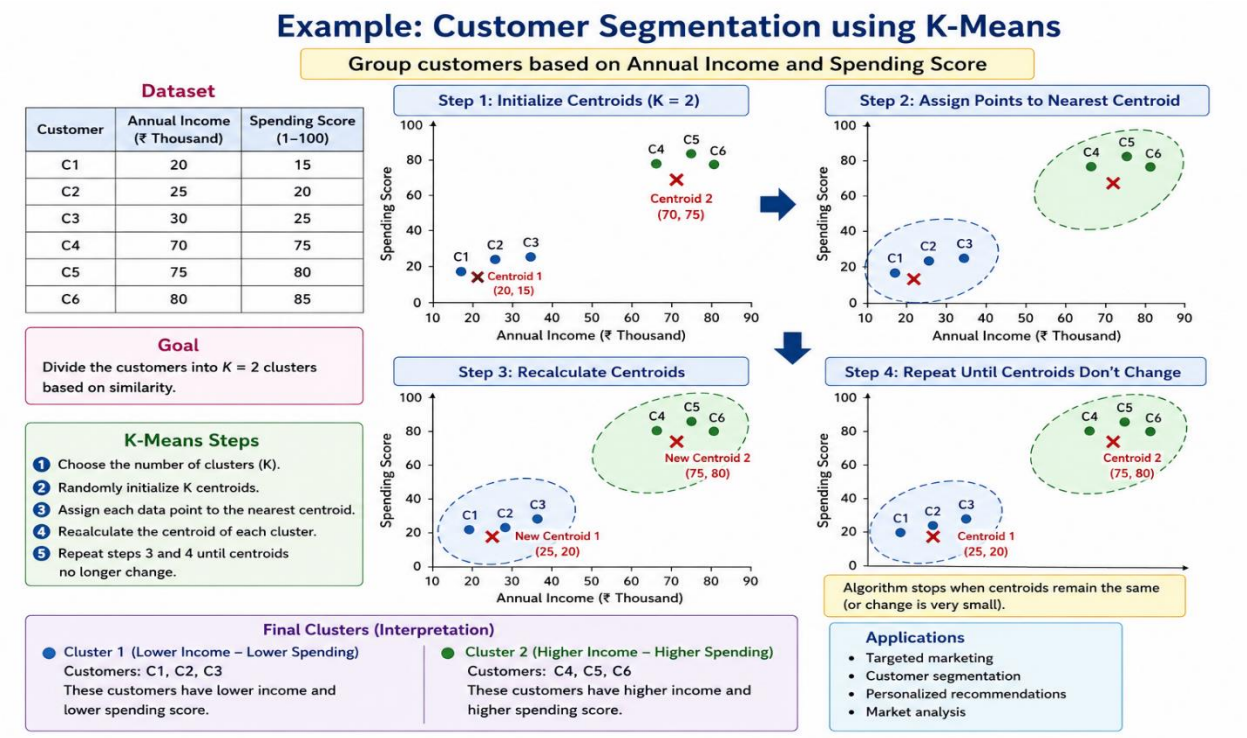
Objective Function

K-Means minimizes the Within-Cluster Sum of Squares (WCSS):

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} \| x - \mu_i \|^2$$

where:

- K = Number of clusters
- C_i = Set of points in cluster iii
- x = Data point
- μ_i= Centroid (mean) of cluster iii



#Program of Decision Tree

```
# Loan Approval System

from sklearn.tree import DecisionTreeClassifier

# Training data
# Features: [Income_High, CreditScore_Good]
# Income_High: 1 = Yes, 0 = No
# CreditScore_Good: 1 = Yes, 0 = No

X = [
    [1, 1],  # High income, Good credit
    [1, 0],  # High income, Poor credit
    [0, 1],  # Low income, Good credit
    [0, 0]   # Low income, Poor credit
]

# Target values
# 1 = Approve Loan
# 0 = Reject Loan
y = [1, 0, 0, 0]

# Create and train the Decision Tree model
model = DecisionTreeClassifier()
model.fit(X, y)

# Test data
# Example: High income and Good credit
test_data = [[1, 1]]

# Make prediction
prediction = model.predict(test_data)

# Display result
if prediction[0] == 1:
    print("Loan Approved")
else:
    print("Loan Rejected")

/*OUTPUT
Loan Approved*/
```

#Program of Mean Squared Error (MSE)

```
# Mean Squared Error (MSE)

# Actual values
actual = [30, 50, 70, 20, 60]

# Predicted values
predicted = [25, 40, 60, 25, 70]

# Calculate MSE
n = len(actual)
sum_squared_error = 0

for i in range(n):
    error = actual[i] - predicted[i]
    squared_error = error ** 2
    sum_squared_error += squared_error

mse = sum_squared_error / n

# Display results
print("Actual Values   :", actual)
print("Predicted Values:", predicted)
print("Mean Squared Error (MSE):", mse)

/*OUTPUT
Sum of Squared Errors = 25 + 100 + 100 + 25 + 100 = 350
Number of observations = 5
          MSE=350 / 5=70
Actual Values   : [30, 50, 70, 20, 60]
Predicted Values: [25, 40, 60, 25, 70]
Mean Squared Error (MSE): 70.0 */
```

Lab Assignments

SET A

1. Write a program to implement the Gaussian Naïve Bayes Classifier for classifying numerical data.
2. Write a program to implement the Bernoulli Naïve Bayes Classifier for binary feature data.
3. Write a program to implement the Decision Tree Classifier for a classification problem.

4. Write a program to implement the Random Forest Classifier for classification tasks.
5. Write a program to implement the AdaBoost Algorithm for supervised classification.
6. Write a program or report to compare different Supervised Learning Algorithms such as Gaussian Naïve Bayes, Bernoulli Naïve Bayes, Decision Tree, Random Forest, and AdaBoost.

Signature of the Instructor | | **Date** | |

SET B

1. Write a program to implement a Voting Classifier by combining multiple machine learning models such as Logistic Regression, Decision Tree, and K-Nearest Neighbors.
2. Write a program to implement the K-Means Clustering Algorithm for grouping data into clusters.
3. Write a program to demonstrate different distance metrics used in Machine Learning.
4. Write a program to demonstrate the objective function used in clustering algorithms.
5. Write a program or report to compare Supervised Learning and Unsupervised Learning techniques.
6. Write a program to analyze and compare the classification accuracy of different machine learning models.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 7 : Support Vector Machine (SVM) (Total slots = 1)

Objective

- To get an idea about SVM (Support Vector Machines) and its working in Machine Learning.
- To get an idea about how SVM classifies objects using the best possible hyperplane.
- To know the applications, benefits, and different types of SVM.

Ready

- The Support Vector Machine (SVM) is a supervised machine learning method that is used for classification and regression problems.
- The main working principle of SVM is that it seeks the optimal decision boundary or hyperplane that separates points belonging to various categories with the largest margin possible.
- The points lying at the extreme ends of the decision boundary or closest to the hyperplane are known as the support vectors.

Ready Reference

Support Vector Machine (SVM) is an artificial intelligence algorithm that is applied for classification and regression problems. This algorithm determines the best possible decision boundary (also known as hyperplane) separating the data points into two different classes based on the maximum possible margin (distance).

The points near the decision boundary are referred to as support vectors, and these help in determining the optimal hyperplane.

Working of SVM

- Gather the training data with labels.
- Determine the classes to be separated.
- Determine the optimal hyperplane with the maximum margin.
- Use support vectors to determine the boundary.
- Classify the new data points based on the side of the hyperplane it lies on.

Example : Fruit Classification

An SVM model classifies fruits based on their weight.

Fruit	Weight (g)	Class
Apple	120	Apple
Apple	130	Apple
Orange	180	Orange
Orange	190	Orange

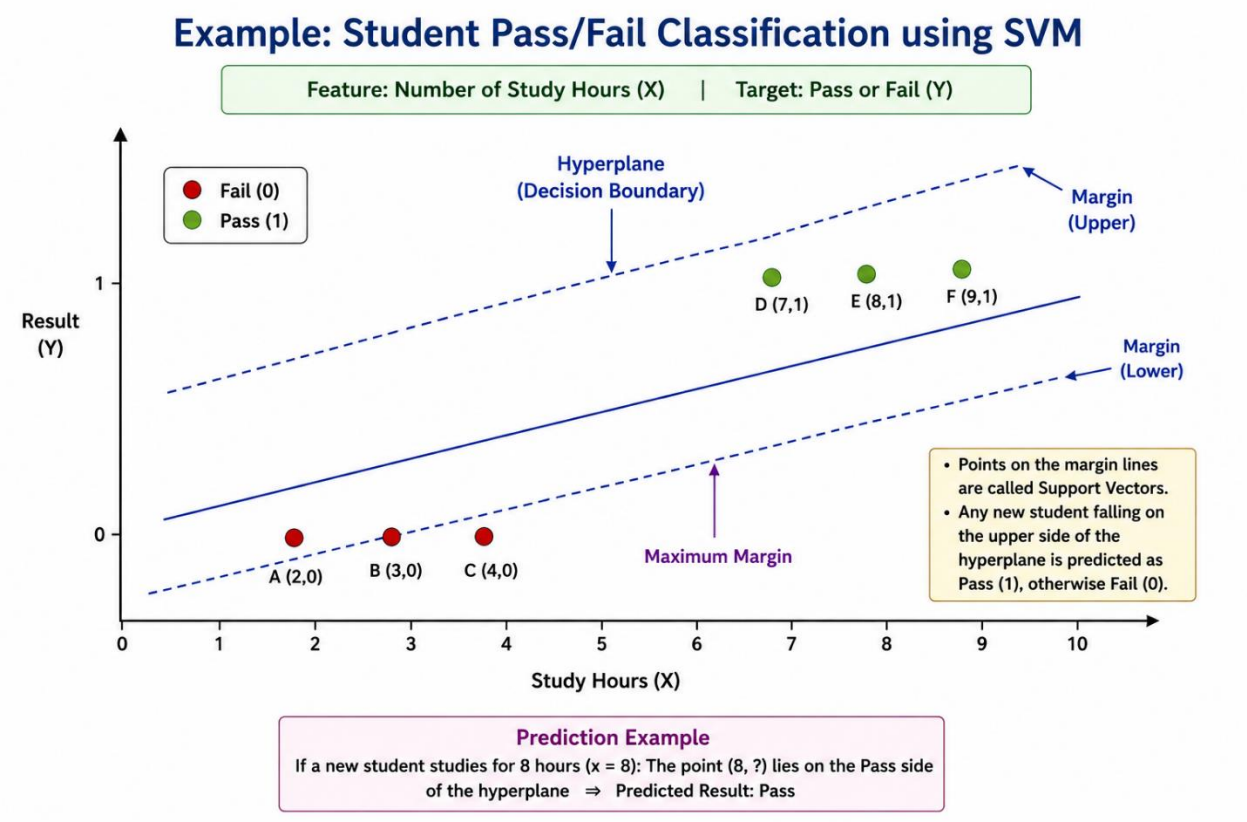
For a new fruit weighing 185 g, the SVM predicts:

Prediction: Orange

Hyperplane

Hyperplane refers to a boundary line adopted by a support vector machine for classification purposes. The hyperplane is a line in two-dimensional space, a plane in three-dimensional space, and the hyperplane when the space has more than three dimensions.

The optimal hyperplane is the one with maximum margin between the classes.



Classification

Classification is a supervised learning method where the learning model learns from data that is labeled and predicts the class of the new inputs. The output is a categorical value like Yes/No, Spam/Non-Spam, and Pass/Fail.

The primary purpose of classification is to classify each input into the right predefined class.

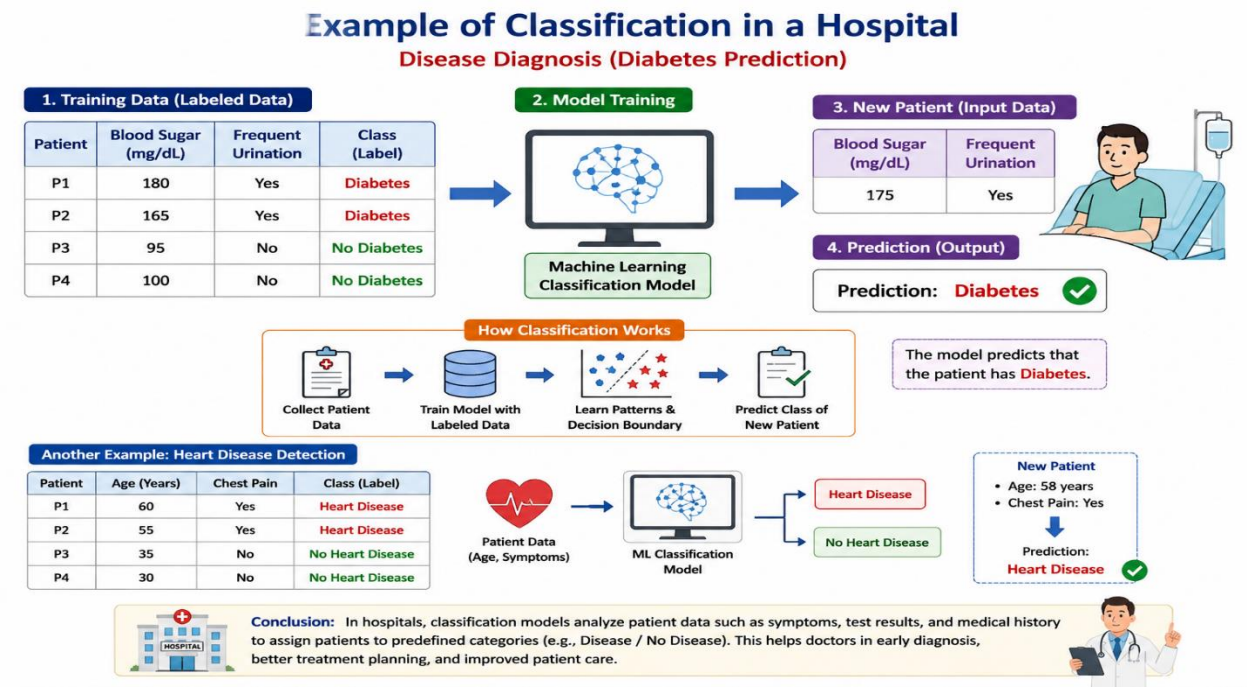
Classification Process

- Gather labeled training data.
- Use the training data to train the classification model.
- Discover patterns and relationships between features and classes.
- Classify the unseen data based on the patterns learned.

Classification Types

1. Binary Classification
   Two classes that exist.
   Example: Spam or Not Spam, Pass or Fail.
2. Multiclass Classification
   More than two classes.
   Example: Animals being classified as either Cats, Dogs, or Rabbits.
3. Multilabel Classification
   An instance may belong to more than one class at once.
   Example: A picture labeled both “Beach” and “Sunset.”

Example: Heart Disease Detection



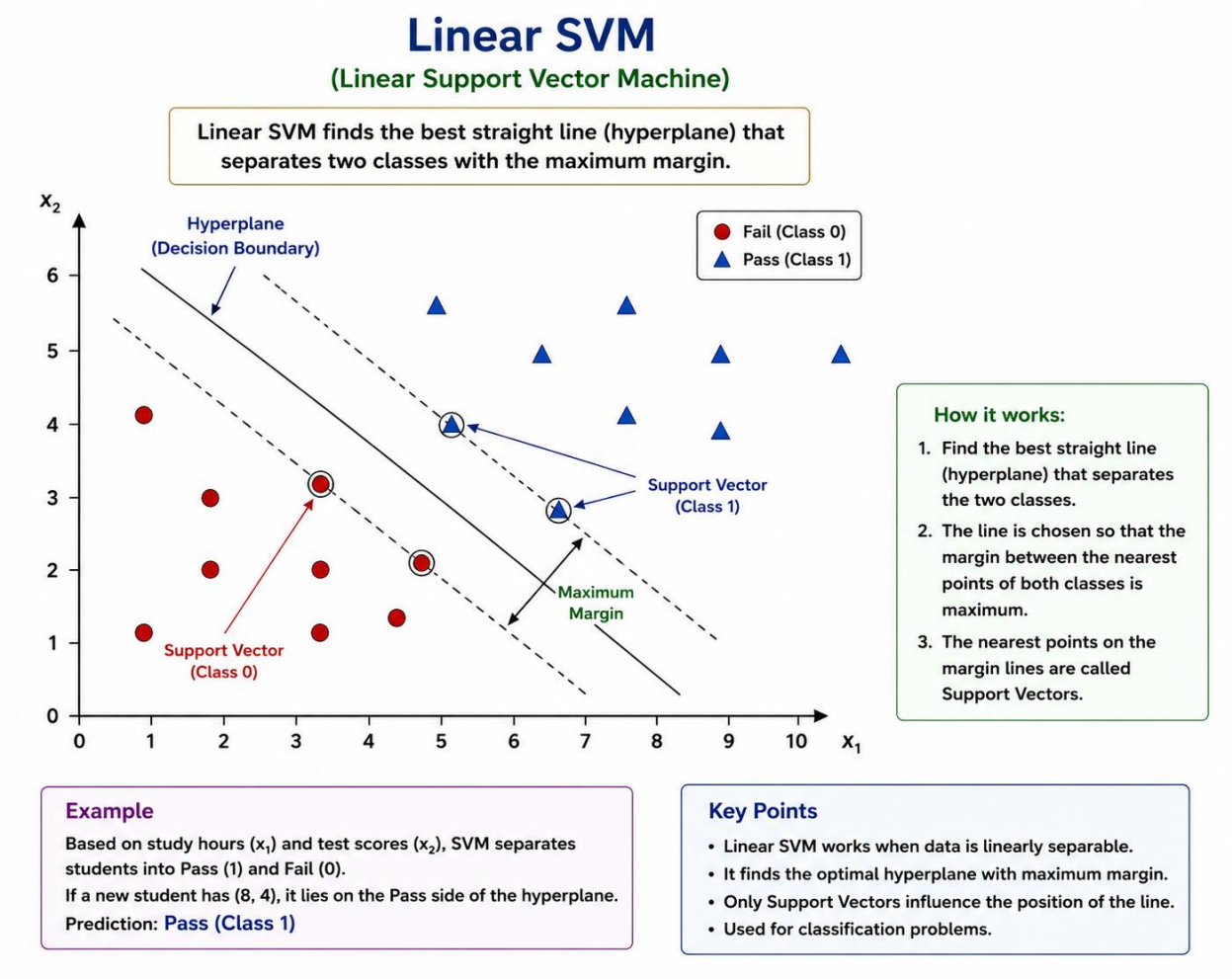
Linear SVM

Linear Support Vector Machine (Linear SVM) is a supervised machine learning technique for classification. It classifies data points into various categories through the process of separating the data points using the best straight line (or hyperplane) that creates the maximum margin between the data sets.Linear SVM algorithm is most effective when the data points are linearly separable.

How Linear SVM Works

- Gather labeled training data.
- Draw data points in the feature space.
- Create the best straight line separating hyperplane.
- Create the maximum margin between the two data sets.
- Classification of new data points through the side of hyperplane they lie.

Example : A school wants to predict whether a student will Pass or Fail based on the number of hours studied.



Non-linear SVM

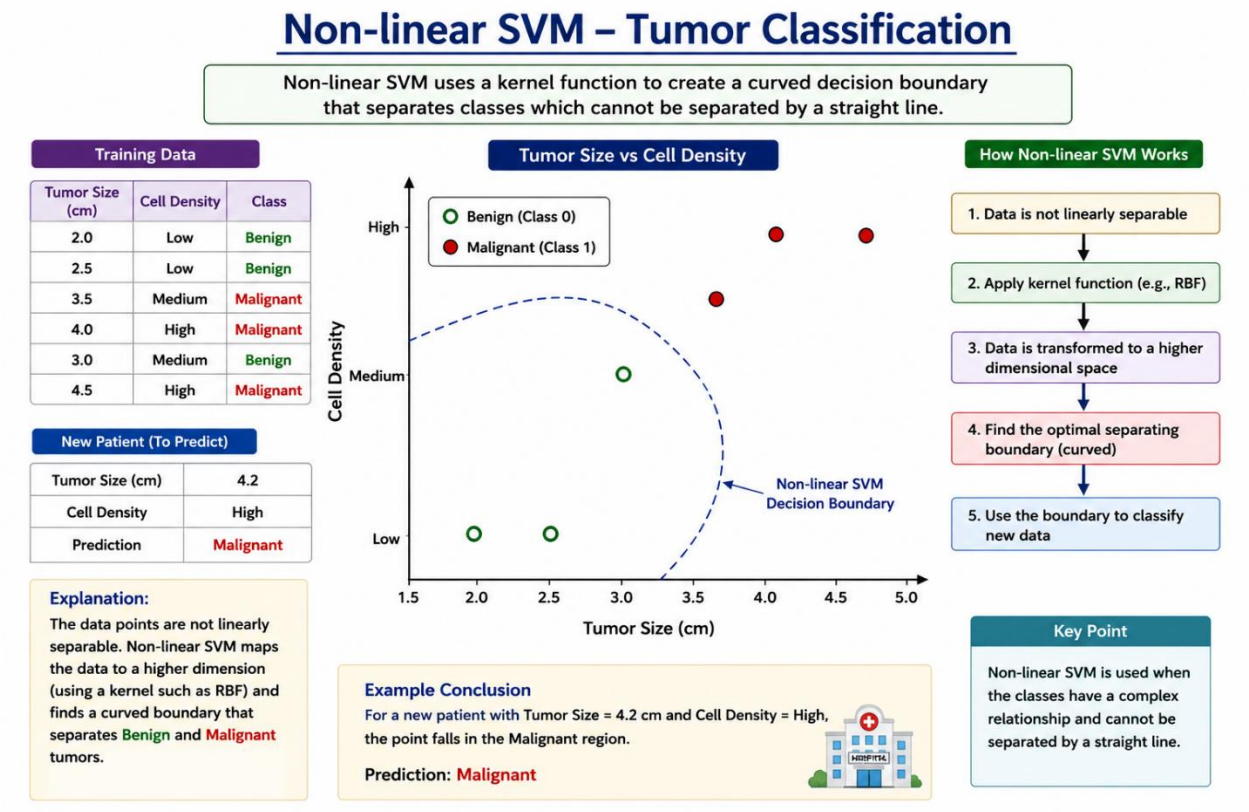
The Non-linear Support Vector Machine (Non-linear SVM) is a supervised machine learning method used for classification in cases where the data is not separable using a straight line. The kernel functions used are RBF, Polynomial, and Sigmoid, which are responsible for mapping the data to a higher dimensional space where a separation boundary can be identified.

In contrast to the Linear SVM, the Non-linear SVM identifies the curved separation boundary used for classifying data sets.

Working of Non-linear SVM

- Get the labeled training data.
- Determine the non-linearly separable data.
- Map the data using a kernel function.
- Identify the optimal separation boundary.
- Classify the new data using the boundary.

Example : Tumor Classification



 # Program for Support Vector Machine (SVM) Spam Email Classification

```
from sklearn import svm

X = [
   [15, 5],
   [12, 4],
   [3, 1],
   [2, 0],
   [14, 6],
   [4, 1]
]

y = [1, 0, 0, 0, 1, 1]

model = svm.SVC(kernel='linear')
model.fit(X, y)

test_email = [[13, 5]]

prediction = model.predict(test_email)

if prediction[0] == 1:
   print("Prediction: Spam")
else:
   print("Prediction: Not Spam")
/*OUTPUT
Prediction: Spam*/
```

 # Program for Hospital Disease Classification

```
from sklearn.tree import DecisionTreeClassifier

# Training data
# Features: [Blood Sugar, Frequent Urination]
X = [
   [180, 1],
   [165, 1],
   [95, 0],
   [100, 0]
]

# Updated target labels
# 1 = Diabetes
# 0 = No Diabetes
y = [1, 0, 0, 1]
```

```
# Train the model
model = DecisionTreeClassifier(random_state=42)
model.fit(X, y)

# New patient
new_patient = [[175, 1]]

# Predict
prediction = model.predict(new_patient)

# Print result
if prediction[0] == 1:
   print("Prediction: Diabetes")
else:
   print("Prediction: No Diabetes")
/*OUTPUT
Prediction: Diabetes */
```

Lab Assignments

SET A

1. Write a program to implement a Linear Support Vector Machine (SVM) for a binary classification problem.
2. Write a program to implement a Non-Linear Support Vector Machine (SVM) using the Radial Basis Function (RBF) Kernel.
3. Write a program to demonstrate hyperplane classification using a Support Vector Machine.
4. Write a program or report to compare Linear SVM and Non-Linear SVM (RBF Kernel).
5. Write a program to visualize the decision boundaries created by an SVM classifier.
6. Write a program to analyze the performance of a Support Vector Machine (SVM) on a classification dataset.

Signature of the Instructor | **Date**

SET B

1. Write a program to implement a Support Vector Machine (SVM) classifier on the Iris dataset.
2. Write a program to implement a Support Vector Machine (SVM) using the Polynomial Kernel for classification.

3. Write a program or report to compare Support Vector Machine (SVM) and Decision Tree classifiers.
4. Write a program to classify handwritten digits using a Support Vector Machine (SVM).
5. Write a program to demonstrate the concept of margin maximization in a Support Vector Machine.
6. Write a program to evaluate the performance of a Support Vector Machine (SVM) using a Confusion Matrix.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 8 : Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN) (Total slots = 3)

Objective

- To Understand the structure and working principle of Artificial Neural Network (ANN) and Convolutional Neural Network (CNN).
- To Learn about the way in which neural networks work with data and perform pattern recognition.
- To Learn about the applications of Artificial Neural Networks and Convolutional Neural Networks in machine learning and image processing.

Ready

- Artificial neural networks are a computational paradigm which was developed based on the way the human brain functions.
- (CNN) is a special form of the neural network which is capable of handling visual images.
- To use cases of both types of neural networks include deep learning processes like image recognition, speech processing, natural language processing, and disease diagnosis.

Ready Reference

Basic Neural Network

An artificial neural network (ANN) is an advanced learning system that is based on the functioning of the human brain. This model is made up of several processing elements known as neurons, which are interconnected and arranged in different layers. The working mechanism of the ANN involves modifying the weights between neurons during the training process.

Artificial neural networks are used to solve classification, regression, and prediction problems.

ANN Architecture

- Input layer: Provides the input to the ANN.
- Hidden layer(s): Information processing through weights and activation functions.
- Output layer: Output generation.

Functioning of a Simple Neural Network

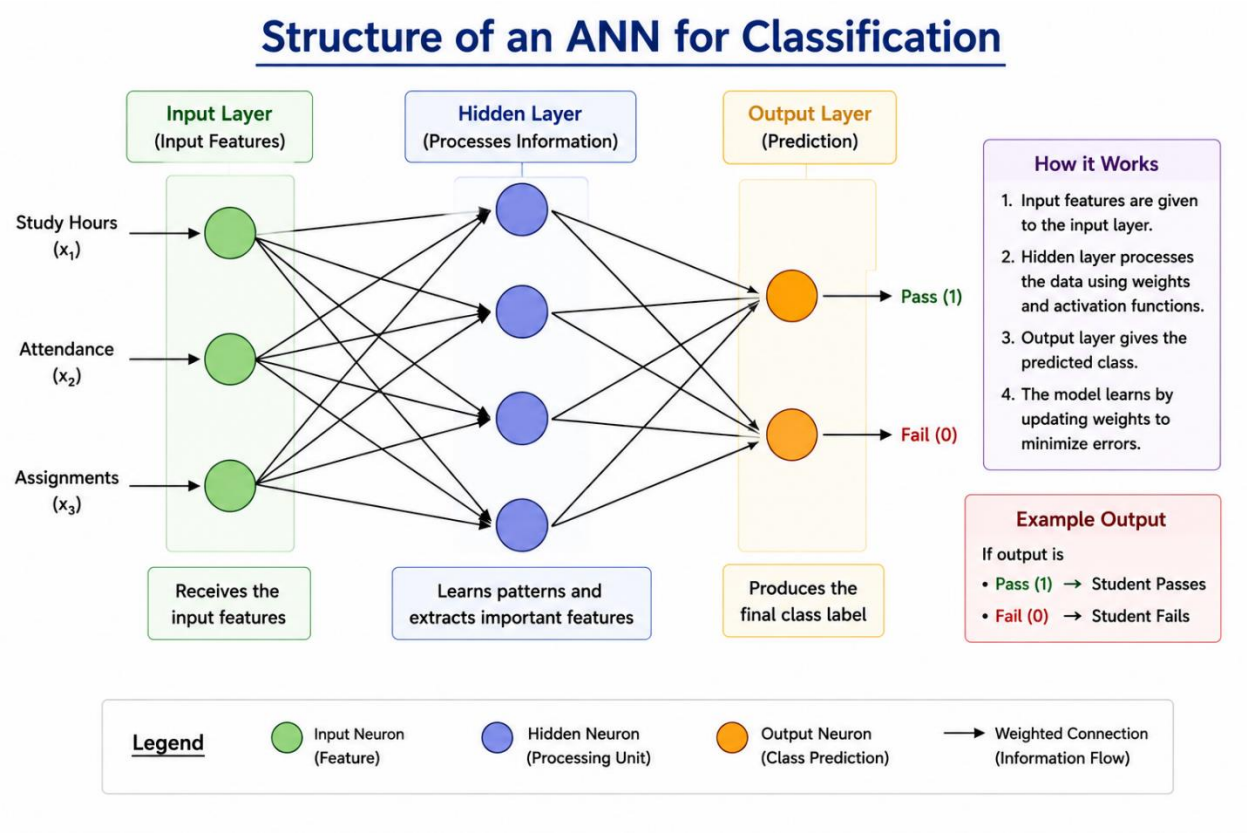
- The inputs are provided to the input layer.
- The hidden layer performs computations based on the input through weighted connections.
- The output is obtained using an activation function for the neurons.
- Finally, the output layer provides the output.

Example: Loan Approval
A bank uses a neural network to decide whether to approve a loan.

Income (₹)	Credit Score	Decision
25,000	550	Reject
35,000	600	Reject
70,000	750	Approve
90,000	800	Approve

New Applicant
Income: **₹80,000**
Credit Score: **780**
Prediction: Approve Loan

ANN model for classification



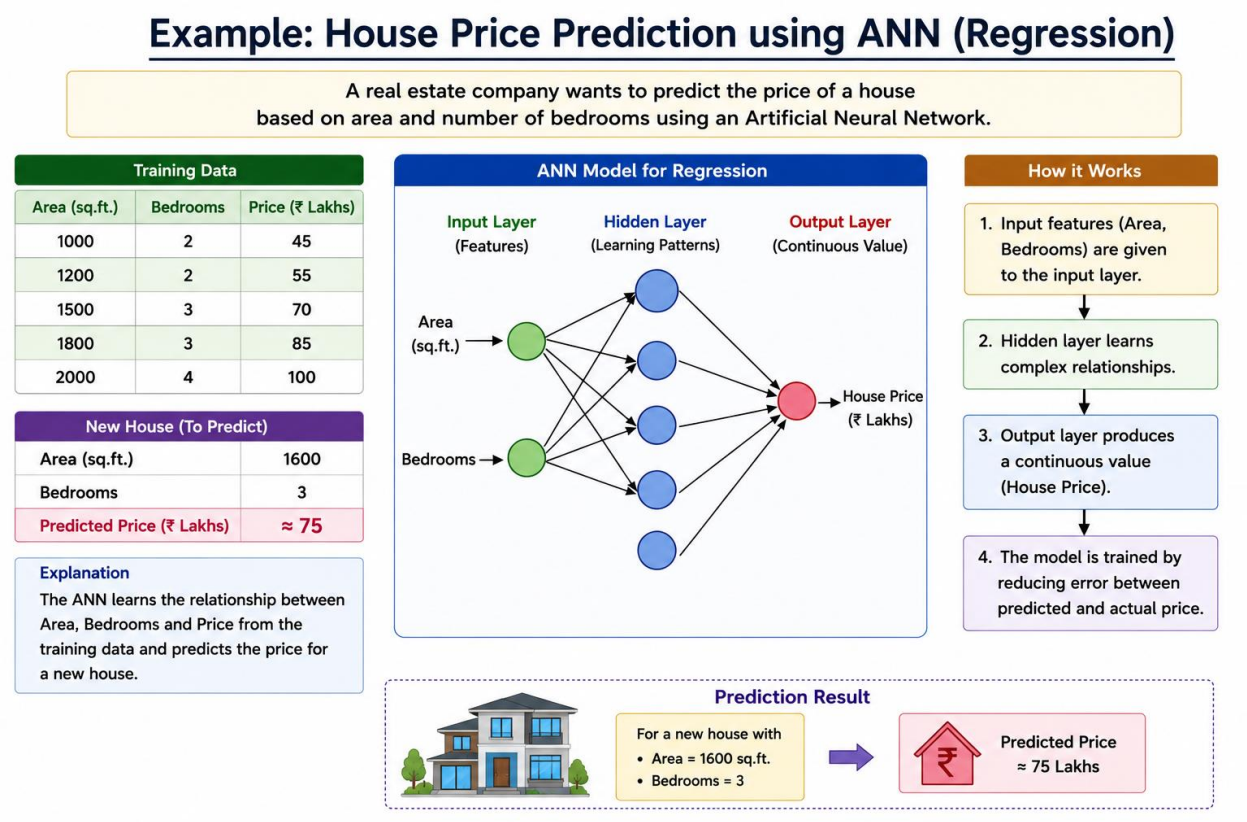
ANN for Regression

Artificial Neural Network (ANN) for Regression is a supervised learning technique for predicting real numerical values instead of classifying into discrete categories. The algorithm learns how the input variables are connected to the numeric output by making weight adjustments in the network's neurons. Classification differs from regression since while in classification the output is discrete (e.g., pass/fail), in regression the output could be prices of houses, temperature, salary levels, or sales.

Structure of ANN for Regression

ANN for regression involves:

- Input Layer – Contains input variables.
- Hidden Layers – Analyzes the information.
- Output Layer – Gives an output variable.



Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) is a form of Artificial Neural Network (ANN) which specializes in image processing. The CNNs are able to automatically discover key features such as edges, patterns, shapes and textures through the use of convolution filters.

Components of a Convolutional Neural Network

- Input Layer – Takes the input image as input.
- Convolutional Layer – Filters are applied to extract relevant features.
- Pooling Layer – The dimension of feature maps is reduced without losing relevant information.
- Fully Connected Layer – The extracted features are combined for decision making.
- Output Layer – Generates the output decision or classification.

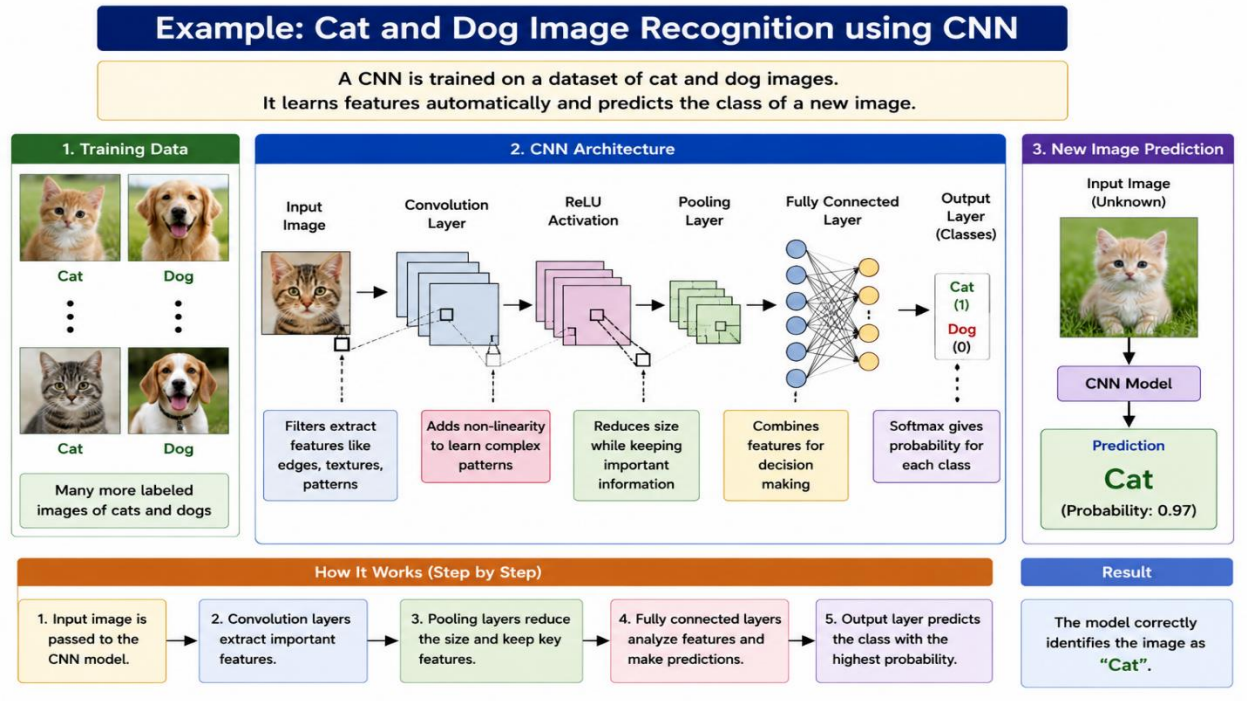
CNN for Image recognition

Convolutional neural network (CNN), on the other hand, is a neural network model that is specifically built for solving the problems related to image recognition and computer vision.

It uses an automatic approach for extracting important visual features like edges, corners, shapes, textures, objects, etc. from images**.**

Structure of CNN in Image Classification

- Input layer – Receives the image.
- Convolutional layer – Finds the relevant features through the use of filters (or kernels).
- Activation layer (ReLu) – Adds non-linearity to the model.
- Pooling layer – Decreases the dimensions of the feature maps without losing important features.
- Fully connected layer – Merges the learned features to predict an outcome.
- Output layer – Returns the image category**.**



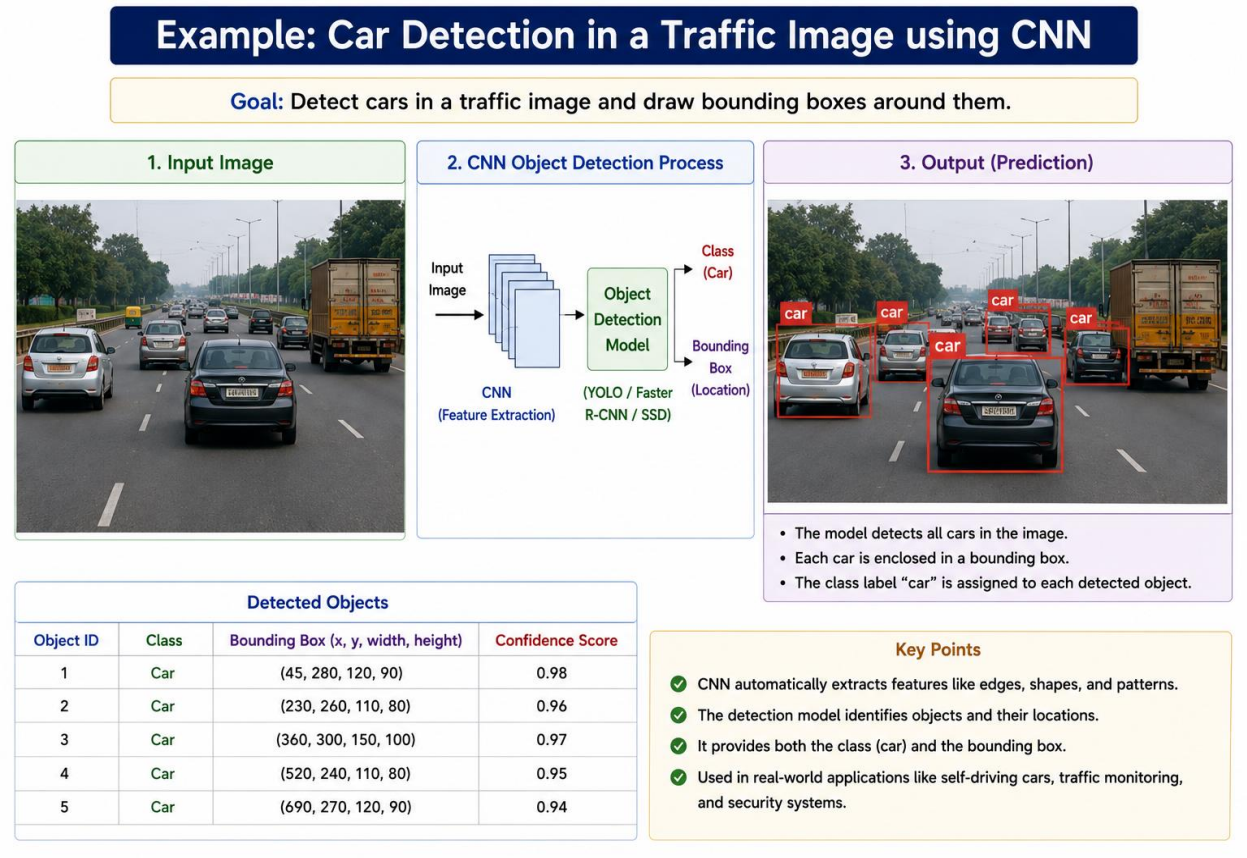
CNN for Object detection

Convolutional Neural Network for Object Detection is a deep learning algorithm that not only detects the kind of object in an image but also its location using a bounding box.

Object detection is different from image classification as the latter assigns a label to the whole image while the former finds out what objects are there and where they exist.

How does CNN for Object Detection work?

- Image is provided as input.
- CNN extracts features like edges, shapes, etc.
- CNN analyzes features and detects objects.
- CNN classifies the objects.
- It creates a bounding box for each object detected.
- CNN for Computer vision tasks



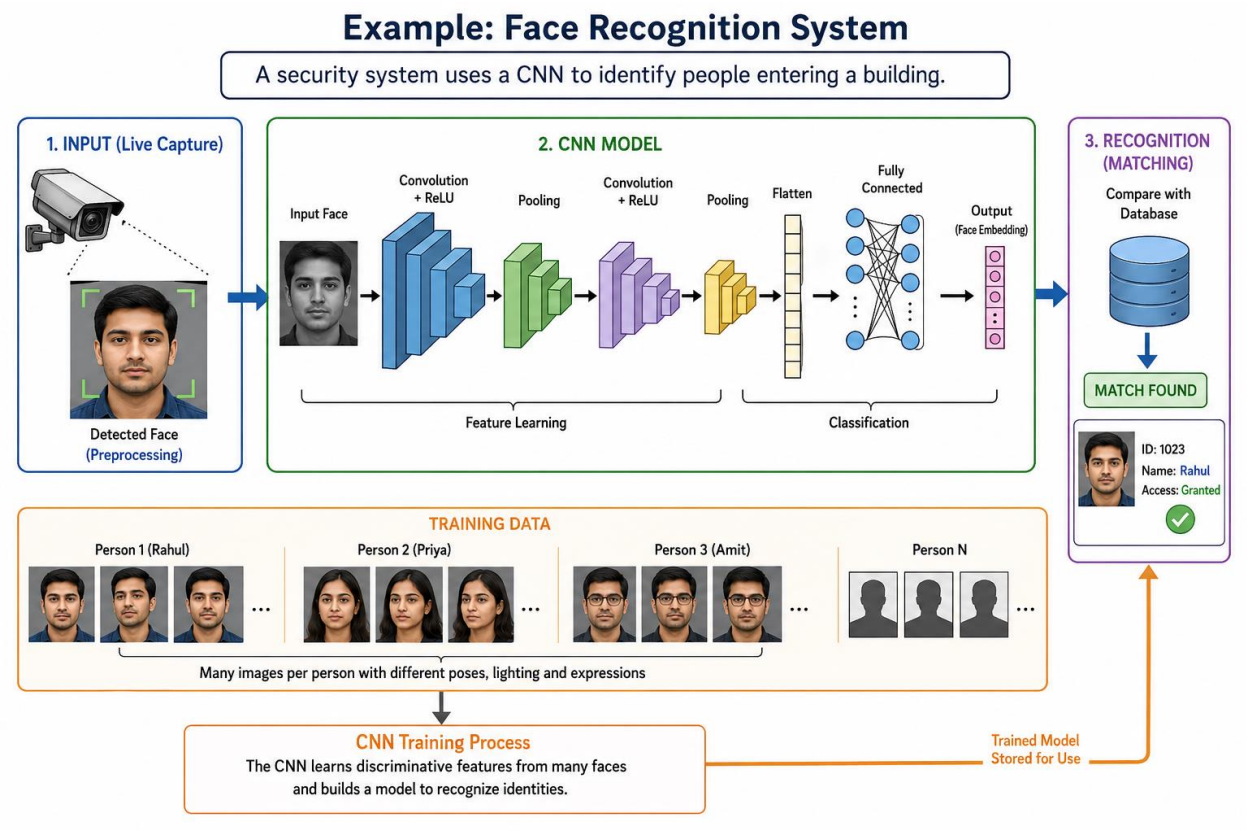
CNN for Computer Vision Tasks

A Convolutional Neural Network (CNN) is a deep learning model that processes images and videos. It is popularly applied in computer vision as it has the ability to automatically learn visual features like edges, shapes, texture, and objects.

Computer vision application allows computers to analyze and comprehend images for applications in object detection, image classification, facial recognition, and medical imaging.

How Does CNN Work in Computer Vision?

- The image is fed as an input.
- Convolution layers identify visual features.
- Pooling layers help in reducing the feature map size without losing essential information.
- Fully connected layers integrate the visual features.
- The output layer identifies the object or the class of the image.



#Program for Basic Neural Network (ANN) : Student Pass/Fail Prediction

```
from sklearn.neural_network import MLPClassifier

# Training data
X = [
    [2, 60],   # Fail
    [3, 65],   # Fail
    [6, 85],   # Pass
    [8, 95]    # Pass
```

```
]

# Labels
y = [0, 0, 1, 1]

# Create and train the model
model = MLPClassifier(hidden_layer_sizes=(5,),
                      max_iter=1000,
                      random_state=42)

model.fit(X, y)

# New student data
new_student = [[9, 95]]

# Predict
prediction = model.predict(new_student)

if prediction[0] == 1:
   print("Prediction: Pass")
else:
   print("Prediction: Fail")

/*OUTPUT
Prediction: Pass*/
```

Example : MNIST handwritten digit dataset

 # Program for Handwritten Digit Recognition using CNN

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Dense

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Reshape and normalize the images
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
```

```
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0

# Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')   # 10 output classes (0–9)
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
model.fit(x_train, y_train, epochs=5)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)

print("Test Accuracy:", accuracy)

# Predict the first test image
prediction = model.predict(x_test[:1])

# Display the predicted digit
predicted_digit = prediction.argmax()

print("Predicted Digit:", predicted_digit)
print("Actual Digit   :", y_test[0])

/*OUTPUT
Test Accuracy: 0.98
Predicted Digit: 7
Actual Digit   : 7*/
```

Lab Assignments

SET A

1. Write a program to implement a Basic Artificial Neural Network (ANN) for predicting outputs from input data.
2. Write a program to implement an Artificial Neural Network (ANN) for a classification task.
3. Write a program to implement an Artificial Neural Network (ANN) for a Linear Regression problem.
4. Write a program to build a Convolutional Neural Network (CNN) for image recognition.(Orange vs Apple)
5. Write a program to implement a Convolutional Neural Network (CNN) for basic object detection.
6. Write a program or report to compare Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN).

Signature of the Instructor | | **Date** | |

SET B

1. Write a program to implement a Multi-Layer Perceptron (MLP) for a classification problem.
2. Write a program to perform handwritten digit recognition using an Artificial Neural Network.
3. Write a program to build a Convolutional Neural Network (CNN) using TensorFlow/Keras.
4. Write a program to classify images using a Convolutional Neural Network (CNN).
5. Write a program to demonstrate feature extraction using a Convolutional Neural Network (CNN).
6. Write a program to analyze the performance of a Convolutional Neural Network (CNN) on an image classification dataset.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 9 : Case Studies

1. A bank wants to reduce financial losses caused by fraudulent transactions by developing a machine learning system that analyzes transaction patterns and identifies suspicious activities. The system is trained on historical data to classify transactions as either legitimate or fraudulent, helping improve security and prevent fraud.(Fraud Detection using Machine Learning)

2. A postal service aims to automate the sorting of mail by recognizing handwritten ZIP codes using artificial intelligence. A neural network model is trained to identify digits from 0 to 9 accurately, reducing manual effort and increasing processing speed. (Handwritten Digit Recognition)

3. A warehouse uses autonomous robots to transport goods between different locations. The robots employ AI search algorithms to find the shortest and safest path while avoiding obstacles, ensuring efficient and collision-free navigation. (Robot Navigation System)

4. A college needs to generate class and examination timetables without conflicts involving teachers, classrooms, or student groups. An AI-based scheduling system applies constraint satisfaction techniques to create an optimized timetable that satisfies all required conditions. (Timetable Scheduling System)

5. A company develops a voice-controlled application that converts spoken language into text. By using artificial intelligence and deep learning techniques, the system accurately recognizes speech and enables users to interact with devices through voice commands. (Speech Recognition System)

6. An autonomous vehicle relies on artificial intelligence to detect roads, traffic signals, pedestrians, and nearby vehicles. Using data from cameras and sensors, the system makes real-time driving decisions to navigate safely without human intervention.(Self-Driving Car Case Study)

7. A hospital implements an AI-powered diagnostic system that analyzes patient symptoms, medical records, and test results to assist doctors in identifying possible diseases. This technology supports faster diagnoses and helps improve treatment planning. (AI in Healthcare)

8. An online learning platform uses artificial intelligence to personalize education by analyzing each student's performance and learning style. The system recommends appropriate study materials and practice exercises to enhance learning outcomes. (AI in Education)

9. An e-commerce website uses a recommendation system to suggest products based on a customer's browsing history, purchase records, and preferences. Personalized recommendations improve the shopping experience and increase customer satisfaction.(Recommendation System)

10. A city implements an AI-based traffic management system that monitors road conditions and vehicle movement in real time. The system dynamically adjusts traffic signal timings to reduce congestion, shorten travel times, and improve road safety.(Smart Traffic Management)

11. A financial institution develops an AI model to monitor credit card transactions and identify unusual spending patterns that may indicate fraud. The system helps prevent unauthorized transactions and protects customers from financial losses.(Credit Card Fraud Detection)

12. A telecommunications company uses machine learning to predict which customers are likely to discontinue its services. By analyzing customer behavior and usage patterns, the company can take proactive measures to improve retention and reduce churn. (Customer Churn Prediction)

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge