

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**10212CS211 – ARTIFICIAL INTELLIGENCE TECHNIQUES LABORATORY**

**LAB TASK**

**Faculty Name: Saranya M.S Slot :**

**Faculty Co-ordinator: Dr. Rajesh Kambattan. K Semester : 25-26 SUMMER**

**Part I**

**Task 1:**

Implementation of Graph search algorithms (Breadth first search and Depth First Search) using following constraints.

**BFS:** Pick any node, visit the adjacent unvisited vertex, mark it as visited, display it, and insert it in a queue. If there are no remaining adjacent vertices left, remove the first vertex from the queue. Apply recursion concept to follow the above steps until the queue is empty or the desired node is found.

**DFS:** Pick any node. If it is unvisited, mark it as visited and recur on all its adjacent nodes. Repeat until all the nodes are visited, or the node to be searched is found.

**Tools- Python**

**PROBLEM STATEMENT: CO1 S3**

**BFS:**

A network engineer working on configuring a large-scale computer network for a company. The network is represented as a graph, where each node represents a network device, and an edge between two nodes signifies a direct connection between the corresponding devices. Your goal is to configure the network routers optimally to ensure the fastest data transmission between two specified devices.

To achieve this, you decide to use the Breadth-First Search (BFS) algorithm to find the shortest path between any two devices in the network. Each device's transmission speed is known, and you want to minimize the latency between devices by choosing the most efficient path.

The task is to implement the BFS algorithm to find the shortest path between two specified devices in the network. Additionally, you must consider the transmission speeds of the links (edges) between devices to determine the most optimal path.

**DFS**

A maze-solving robot navigating through a complex maze represented as a 2D grid. The maze consists of cells, where each cell can be either open ('O') or blocked ('X'). Your mission is to find a path from the starting position ('S') to the goal position ('G') using the Depth-First Search (DFS) algorithm.

The maze is laid out in a grid, and you can move in four directions: up, down, left, and right. However, you cannot move diagonally, and you cannot pass through blocked cells ('X'), as they represent walls that are impassable.

The task is to implement the DFS algorithm to find a valid path from the starting position ('S') to the goal position ('G'). The DFS algorithm will explore the maze in a recursive manner, backtracking whenever it encounters a dead-end or reaches a blocked cell.

**Task 2:**

Implementation of Hill climbing algorithm for Heuristic search approach using following constraints in python.

i. Create a function generating all neighbours of a solution

ii. Create a function calculating the length of a route

iii. Create a random solution generator

iv. Create a Travelling salesman problem

**Tools- Python, Online Simulator - https://graphonline.ru/en/**

**PROBLEM STATEMENT: CO1 S3**

Develop an intelligent agent to solve a challenging puzzle called the "Robot Maze Escape." The puzzle consists of a 2D grid representing a maze, where the agent (robot) needs to find its way from the starting position 'S' to the goal position 'G.' The maze contains walls represented by blocked cells 'X,' and the agent can move in four directions: up, down, left, and right.

To navigate through the maze efficiently, you decide to use the Hill Climbing algorithm as a heuristic search approach. The Hill Climbing algorithm is a local search algorithm that continuously moves towards the most promising solution by exploring neighboring states and choosing the one with the highest heuristic value.

The task is to implement the Hill Climbing algorithm in Python to guide the robot's movement through the maze towards the goal position. You will use a heuristic function that estimates the distance from the current position to the goal position, helping the robot make decisions on the best direction to move.

**Task 3:**

Implementation of A \* Algorithm to find the optimal path using Python by following constraints.

•The goal of the A\* algorithm is to find the shortest path from the starting point to the goal point as fast as possible.

•The full path cost (f) for each node is calculated as the distance to the starting node (g) plus the distance to the goal node (h).

•Distances is calculated as the manhattan distance (taxicab geometry) between nodes.

**Tools- Python, Online Simulator -** [**https://graphonline.ru/en/**](https://graphonline.ru/en/)

**PROBLEM STATEMENT: CO2 S3**

A software developer working on a project to create a GPS navigation system for autonomous vehicles. The system needs to find the optimal path between two locations on a road network to ensure efficient and safe navigation. To achieve this, you decide to implement the A\* algorithm, a popular heuristic search algorithm, in Python.

The road network is represented as a graph, where each node represents an intersection, and an edge between two nodes represents a road segment connecting the intersections. Each road segment has a weight or cost, which corresponds to the distance between the intersections.

The task is to implement the A\* algorithm to find the optimal path between two specified locations on the road network. The A\* algorithm uses a heuristic function that estimates the cost from each node to the goal, guiding the search towards the most promising path while considering the actual cost of reaching each node.

**Task 4:**

Implementation of Mini-Max algorithm uses recursion to search through the game-tree using python by applying following constraints.

•In this algorithm two players play the checker’s game; one is called MAX and other is called MIN.

•Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.

•Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.

•The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.

•The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

**Tools : Python**

**PROBLEM STATEMENT: CO2 S3**

A computer scientist working on developing an AI player for a two-player board game called "Strategic Chess." The game is a simplified version of chess, where each player takes turns moving their pieces on a 4x4 chessboard. The goal is to capture the opponent's pieces and eventually checkmate their king.

To create a formidable AI opponent, you decide to implement the Mini-Max algorithm using recursion in Python. The Mini-Max algorithm is a decision-making algorithm commonly used in two-player games. It explores the game tree by recursively evaluating all possible moves and their outcomes, considering both the AI player's maximizing moves and the opponent's minimizing moves.

The task is to implement the Mini-Max algorithm in Python to enable the AI player to make strategic decisions during the game. The algorithm should be able to evaluate the current board state and select the best move that maximizes the AI player's chances of winning while minimizing the opponent's chances.

**Task 5:**

Implementation of Ant Colony Optimization to Optimize Ride-Sharing Trip Duration using Python by following constraints.

•To forecast travel times between every pair of pick-up and drop-off locations.

•To find the shortest route that visits a set of locations.

•To implement optimization techniques are required to intelligently search the solution space and find near-optimal solutions.

**Tools: Python**

**PROBLEM STATEMENT: CO2 S3**

A data scientist working for a ride-sharing company, "RideSmart," that aims to optimize trip durations and minimize passenger wait times. The company wants to implement an advanced optimization technique to efficiently manage ride-sharing routes. To achieve this, you decide to use the Ant Colony Optimization (ACO) algorithm in Python.

The problem involves finding the most optimal routes for multiple ride-sharing requests based on passenger pickup and drop-off locations. The road network is represented as a graph, where each node represents a pickup or drop-off location, and edges between nodes represent the possible routes between locations. The algorithm will mimic the behavior of ants searching for food, leaving pheromone trails on edges, and making decisions based on these pheromone trails and the heuristic information.

The task is to implement the Ant Colony Optimization algorithm in Python to optimize ride-sharing trip durations for multiple requests. The algorithm should efficiently explore possible routes, considering both the pheromone levels and heuristic information, to find the best routes that minimize the overall trip duration for all passengers involved.

**Task 6:**

Solve a Map Coloring problem using constraint satisfaction approach by applying following constraints

•Assign each territory a color such that no two adjacent territories have the same color by considering following parameters: Domains, Variables and Constraints

•Apply Basic Greedy Coloring Algorithm: Color first vertex with first color, do following for remaining V-1 vertices.

•Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

**Tools- Python, Online Simulator -** [**https://graphonline.ru/en/**](https://graphonline.ru/en/)

**PROBLEM STATEMENT: CO3 S3**

In a fictional country, there are several political districts that need to be colored for an upcoming election. Each district is represented as a region on a map, and the goal is to assign a color to each district such that no two neighboring districts have the same color. The constraint satisfaction approach will be used to solve the Map Coloring problem. The country has multiple political parties, and each party has a specific set of colors associated with it. The districts are interconnected, and no two adjacent districts can be assigned the same color since it represents political affiliations. The objective is to ensure that the map is colored in a way that avoids any potential clashes between neighboring districts' colors.

The goal is to ensure a fair and visually appealing political districting map by assigning colors to districts in a way that respects the political affiliations and avoids color clashes between neighboring regions.

**Task 7:**

Implementation of Monkey Banana Problem in Goal Stack planning using python by applying following constraints.

Imagine a room containing a monkey, chair and some bananas. That have been hanged from the centre of ceiling. If the monkey is clever enough, he can reach the bananas by placing the chair directly below the bananas and climb on the chair. The problem is to prove the monkey can reach the bananas.The monkey wants it, but cannot jump high enough from the floor. At the window of the room there is a box that the monkey can use. The monkey can perform the Following actions: -

1) Walk on the floor.

2) Climb the box.

3) Push the box around (if it is beside the box).

4) Grasp the banana if it is standing on the box directly under the banana.

**Tools: Python**

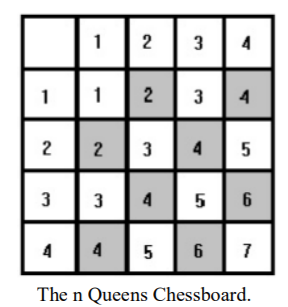
**PROBLEM STATEMENT: CO3 S3**

In a jungle, there is a mischievous monkey who loves bananas. The monkey has spotted a bunch of bananas hanging from the ceiling in a tall room. The goal is for the monkey to reach the bananas and eat them, but there are several obstacles in its way. The monkey must use Goal Stack planning in Artificial Intelligence to come up with a sequence of actions to achieve its goal.The room is represented as a 2D grid, and the monkey starts at a specific location on the ground floor. The bananas are hanging from the ceiling at a higher position. To reach the bananas, the monkey must climb on boxes or furniture placed around the room.The goal is to guide the monkey to make a sequence of actions that involve moving around the room, climbing on furniture or boxes, and avoiding obstacles to eventually reach the bananas and enjoy its delicious treat.

**Task 8:**

Implementation of N-queen problem using backtracking algorithm using python

In the 4 Queens problem the object is to place 4 queens on a chessboard in such a way that no queens can capture a piece. This means that no two queens may be placed on the same row, column, or diagonal.



**Tools: Python**

**PROBLEM STATEMENT: CO3 S3**

In a kingdom, the Queen is facing a challenging puzzle called the N-Queen problem. The Queen wants to place N queens on an NxN chessboard in such a way that no two queens threaten each other, i.e., no two queens share the same row, column, or diagonal. The Queen seeks to use the Backtracking algorithm in Artificial Intelligence to find all possible solutions to this puzzle.

The Queen is well-versed in chess and knows that having multiple solutions to the N-Queen problem is common. However, she wants to explore all the valid arrangements of the queens on the chessboard, ensuring that no queen can capture another.

The program should efficiently explore all valid solutions, considering the constraints of not having two queens threatening each other. The Queen wants to see all distinct arrangements of queens on the board to gain a better understanding of how to solve this classic puzzle.

**Task 9: CO4 S3**

To Build an Intelligent Chatbot system with Python and Dialog-flow using Interactive Text Mining Framework for Exploration of Semantic Flows in Large Corpus of Text.

* To integrate with Google Cloud Speech-to-Text and third-party services such as Google Assistant, Amazon Alexa, and Facebook Messenger.
* Configure Dialogflow to manage your data across GCP services and let you optionally integrate Google Assistant.

**Tools- Python**

**Task 10: CO5 S3**

Implement simple fact for following:

a. Ram likes mango.

b. Seema is a girl.

c. Bill likes Cindy.

d. Rose is red.

e. John owns gold.

**Tools-Python**

**Part II Use Cases**

**Use Case 1:**

Implement Smart Garbage Collector Bot **CO1-CO5 S3**

Design an autonomous bot that navigates a city grid to collect garbage bins placed at various locations using optimal search strategies and plans the route within limited battery constraints

**Use Case 2:**

Implementation of Intelligent Chat-Bot system using Python **CO4,CO5 S3**

Building an Intelligent Travel Chat-Bot

You are a software developer tasked with building an intelligent chat-bot system for a travel agency. The chat-bot will assist customers in planning their vacations, providing travel information, and answering inquiries about destinations, accommodations, activities, and more.

To implement this system, you will use Python for the backend logic and integration with the Dialogflow platform for natural language processing (NLP) and conversational abilities. Dialogflow is a powerful NLP platform that allows you to build conversational interfaces for various applications, including chat-bots.

By building an intelligent chat-bot system, the travel agency can provide instant support and personalized travel recommendations to its customers, enhancing their overall experience and increasing customer satisfaction. Additionally, the chat-bot can handle multiple users simultaneously, saving time for both customers and the agency's support staff.

SIGNATURE OF: FACULTY SIGNATURE FACULTY CO-ORDINATOR **(Dr.M.S.Saranya) (Dr. Rajesh Kambattan. K)**