Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

Jhanvi Parekh

60009210033

D11

Experiment 10

(Planning)

Aim: Implement a plan using AO*.

Theory:

The Depth-first search and Breadth-first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined since all goals following an AND node must be realized; whereas a single goal node following an OR node will do. So for this purpose, we are using AO* algorithm. Like A* algorithm here we will use two arrays and one heuristic function.

OPEN: It contains the nodes that have been traversed but yet not been marked solvable or unsolvable.

CLOSE: It contains the nodes that have already been processed.

AO* Search Algorithm

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved.

If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

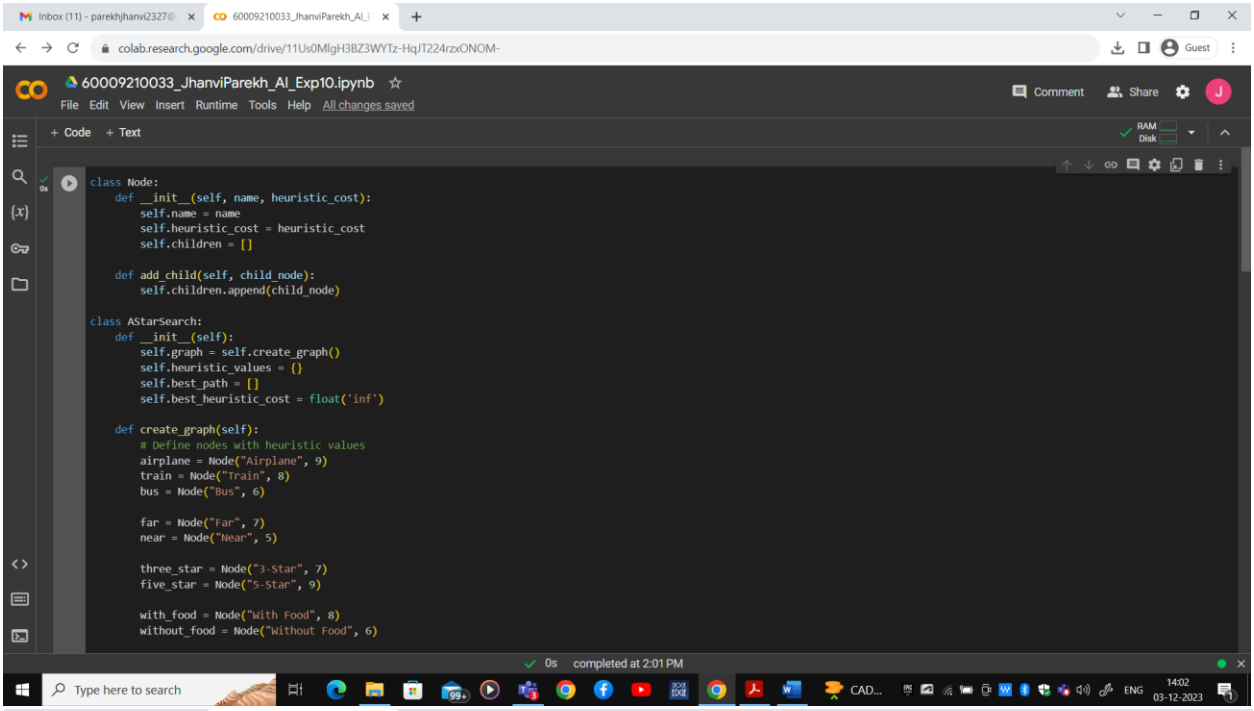
Step 8: Exit.

Lab Assignment to do:

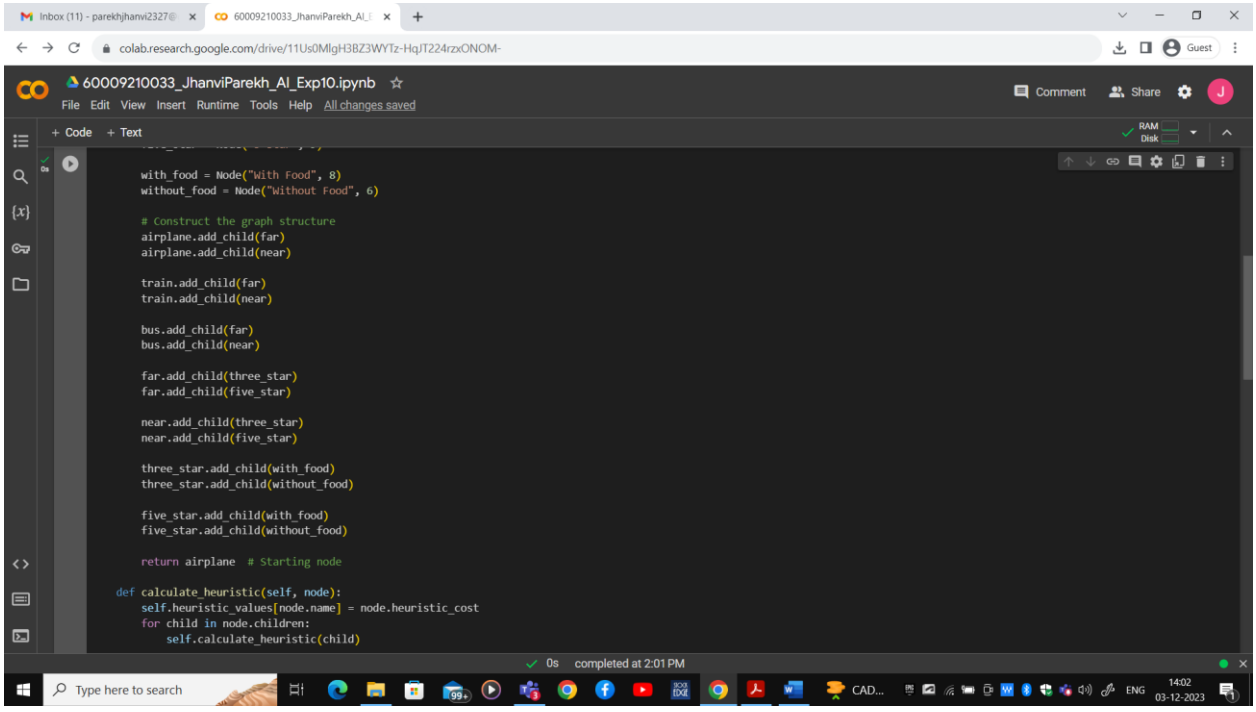
Consider the use case of a plan to travel from Mumbai to Goa to attend a wedding at Taj Aguada. The plan needs to be decided based on the cost. You can either travel by train or bus or flight and stay in a hotel near or far to the wedding venue. The three options of the venues are Westin, Kennel Worth and Maria Rica hotels. You can choose between a two days package for stay and meal together or separately. Other option for your travel and stay will be a vanity van. There you need to decide if you want to cook or eat outside.
Implement AO* to find the most suitable plan in terms of cost.

LINK: https://colab.research.google.com/drive/11Us0MlgH3BZ3WYTz-HqJT224rzxONOM-?usp=sharing

Department of Computer Science and Engineering (Data Science)

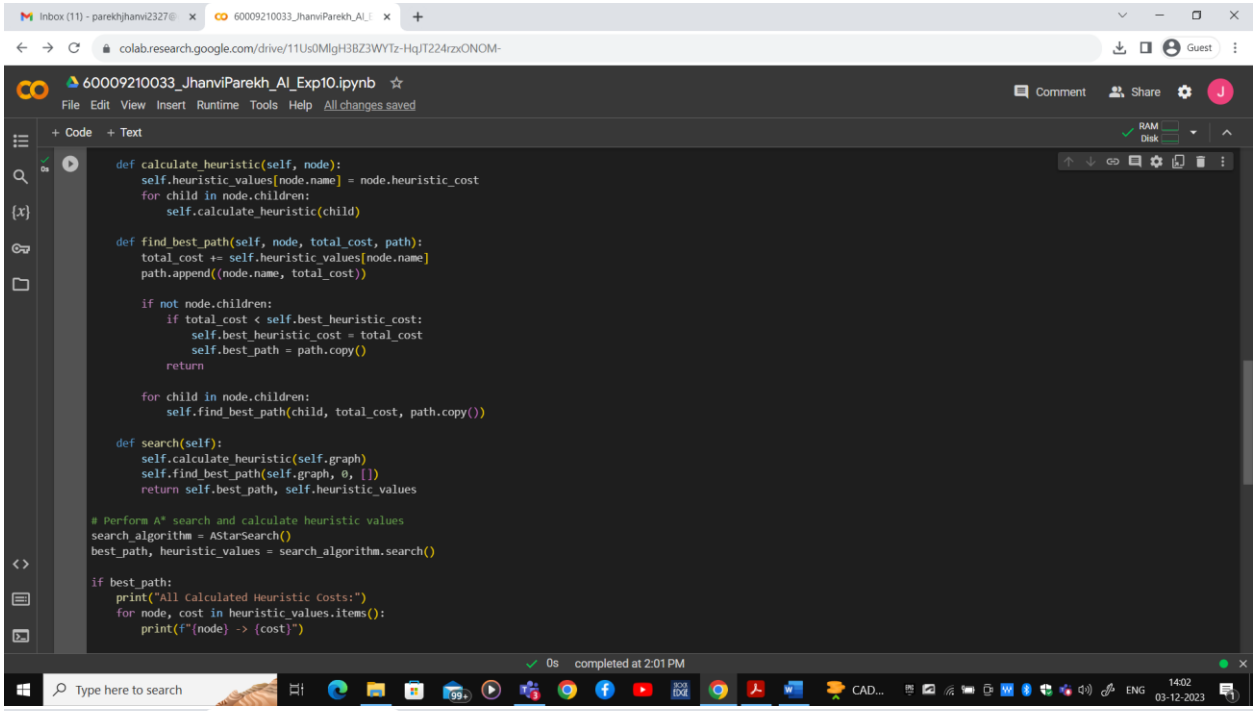


```
class Node:
    def __init__(self, name, heuristic_cost):
        self.name = name
        self.heuristic_cost = heuristic_cost
        self.children = []

    def add_child(self, child_node):
        self.children.append(child_node)

class AStarSearch:
    def __init__(self):
        self.graph = self.create_graph()
        self.heuristic_values = {}
        self.best_path = []
        self.best_heuristic_cost = float('inf')

    def create_graph(self):
        # Define nodes with heuristic values
        airplane = Node("Airplane", 9)
        train = Node("Train", 8)
        bus = Node("Bus", 6)

        far = Node("Far", 7)
        near = Node("Near", 5)

        three_star = Node("3-Star", 7)
        five_star = Node("5-Star", 9)

        with_food = Node("With Food", 8)
        without_food = Node("Without Food", 6)
```

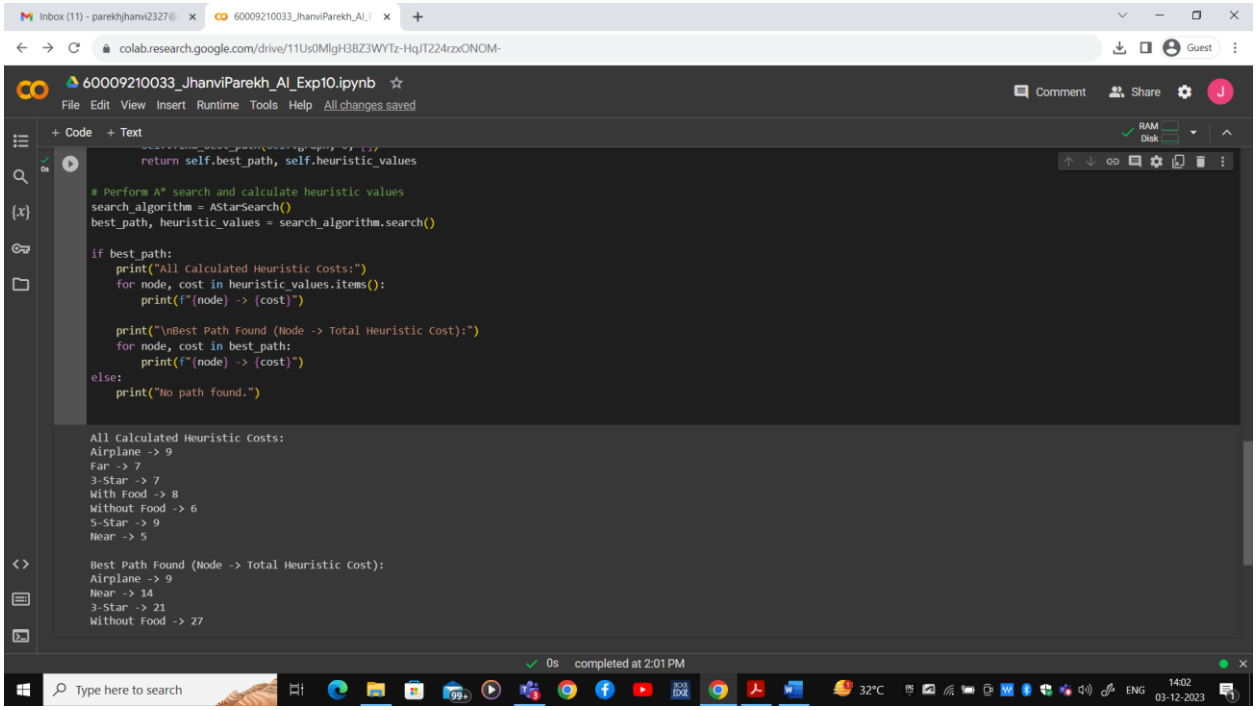


```
        with_food = Node("With Food", 8)
        without_food = Node("Without Food", 6)

        # Construct the graph structure
        airplane.add_child(far)
        airplane.add_child(near)

        train.add_child(far)
        train.add_child(near)

        bus.add_child(far)
        bus.add_child(near)

        far.add_child(three_star)
        far.add_child(five_star)

        near.add_child(three_star)
        near.add_child(five_star)

        three_star.add_child(with_food)
        three_star.add_child(without_food)

        five_star.add_child(with_food)
        five_star.add_child(without_food)

        return airplane  # Starting node

    def calculate_heuristic(self, node):
        self.heuristic_values[node.name] = node.heuristic_cost
        for child in node.children:
            self.calculate_heuristic(child)
```

Department of Computer Science and Engineering (Data Science)



```
    def calculate_heuristic(self, node):
        self.heuristic_values[node.name] = node.heuristic_cost
        for child in node.children:
            self.calculate_heuristic(child)

    def find_best_path(self, node, total_cost, path):
        total_cost += self.heuristic_values[node.name]
        path.append((node.name, total_cost))

        if not node.children:
            if total_cost < self.best_heuristic_cost:
                self.best_heuristic_cost = total_cost
                self.best_path = path.copy()
            return

        for child in node.children:
            self.find_best_path(child, total_cost, path.copy())

    def search(self):
        self.calculate_heuristic(self.graph)
        self.find_best_path(self.graph, 0, [])
        return self.best_path, self.heuristic_values

# Perform A* search and calculate heuristic values
search_algorithm = AStarSearch()
best_path, heuristic_values = search_algorithm.search()

if best_path:
    print("All Calculated Heuristic Costs:")
    for node, cost in heuristic_values.items():
        print(f"{node} -> {cost}")
```



```
        return self.best_path, self.heuristic_values

# Perform A* search and calculate heuristic values
search_algorithm = AStarSearch()
best_path, heuristic_values = search_algorithm.search()

if best_path:
    print("All Calculated Heuristic Costs:")
    for node, cost in heuristic_values.items():
        print(f"{node} -> {cost}")

    print("\nBest Path Found (Node -> Total Heuristic Cost):")
    for node, cost in best_path:
        print(f"{node} -> {cost}")
else:
    print("No path found.")
```

```
All Calculated Heuristic Costs:
Airplane -> 9
Far -> 7
3-Star -> 7
With Food -> 8
Without Food -> 6
5-Star -> 9
Near -> 5

Best Path Found (Node -> Total Heuristic Cost):
Airplane -> 9
Near -> 14
3-Star -> 21
Without Food -> 27
```