*Route Finding System Using A and UCS Algorithm**

1. Introduction

This program implements a route-finding system that utilizes **A* search** and **Uniform Cost Search (UCS)** algorithms. It is designed to help users find the most optimal route between two cities based on distance data. The system also allows the use of different heuristic functions to guide the A* algorithm for faster route finding.

The core functionality is built on two algorithms:

- **A* Search (A-star)**: An informed search algorithm that uses both the actual travel cost and a heuristic estimate of the remaining cost to prioritize the most promising paths.
- **Uniform Cost Search (UCS)**: An uninformed search algorithm that focuses only on the actual travel cost, ensuring the shortest path but without any heuristic assistance.

2. System Overview

The system relies on **road distance data** between cities, which is loaded from a CSV file. The data is structured such that the first column contains city names, while the other columns represent the distance between the cities.

The program supports the following features:

- **A* Search Algorithm** with user-defined heuristics (Admissible or Inadmissible).
- **Uniform Cost Search (UCS)** which finds the optimal route based purely on actual distances.
- **Heuristic Function Options**:
    - **Admissible Heuristic**: The heuristic function that returns the exact distance between cities.
    - **Inadmissible Heuristic**: An overestimating heuristic where the distance is multiplied by 2.

The system allows users to enter two cities, the search algorithm, and the heuristic, and then returns the optimal route along with the total distance.

3. Algorithm Description

3.1 A* Search Algorithm

The A* algorithm searches through the graph by maintaining two scores:

- **g_score**: The actual cost to reach a city from the start.
- **f_score**: The estimated total cost to reach the destination (f_score = g_score + heuristic).

The A* algorithm prioritizes cities with the least f_score to explore. It uses a **min-heap** (priority queue) to always expand the least-cost option. The `admissible_heuristic` guarantees optimal paths since it never overestimates the cost, while the `inadmissible_heuristic` may provide quicker solutions but can overestimate distances, leading to less optimal paths.

3.2 Uniform Cost Search (UCS)

Uniform Cost Search (UCS) is a simpler, uninformed algorithm that searches for the shortest path based purely on actual travel costs, i.e., the `g_score`. The algorithm also uses a priority queue to explore cities in the order of increasing total travel cost.

3.3 Heuristic Functions

- **Admissible Heuristic**: The heuristic function in A* that provides the exact road distance between the current city and the goal city. It guarantees that the algorithm finds the shortest possible path.
- **Inadmissible Heuristic**: A heuristic that doubles the distance between two cities, overestimating the cost. This heuristic may lead to faster solutions but could provide suboptimal paths.

4. Data Input and Structure

The program reads from a CSV file named `Road_Distance.csv`, which should contain the following format:

City	City1	City2	City3	...
CityA	0	50	120	...
CityB	50	0	80	...
CityC	120	80	0	...
...	...	...	...	...

Each row corresponds to a city, with distances to other cities listed in the same order as the header row.

5. Functions Breakdown

- **`load_data(file_path)`**:
  - Reads the CSV file, extracts the city names, and stores the distances between cities in a dictionary format for easy lookup.

- **`admissible_heuristic(city, goal, data)`**:
  - Returns the exact road distance between the current city and the goal city. This function is used in the A* algorithm.
- **`inadmissible_heuristic(city, goal, data)`**:
  - Returns the road distance multiplied by a factor of 2, making it an inadmissible heuristic.
- **`astar_search(graph, start, end, heuristic_func, data)`**:
  - Implements the A* algorithm, using both the actual cost and heuristic to find the shortest route between two cities.
- **`uniform_cost_search(graph, start, end)`**:
  - Implements the UCS algorithm, finding the shortest path by considering only the actual cost.
- **`reconstruct_path(came_from, current)`**:
  - Reconstructs the path taken from the start city to the end city by backtracking from the destination to the start city using the `came_from` dictionary.
- **`calculate_total_distance(data, path)`**:
  - Calculates the total distance of the route by summing the distances between each consecutive city in the path.
- **`find_route(data, start_city, end_city, search_algorithm, heuristic_func)`**:
  - A main function that allows the user to select the search algorithm (A* or UCS) and the heuristic, then finds the optimal route and returns it along with the total distance.

6. User Interaction

The program provides an interactive interface where users can:

1. **Enter the starting city**: The city from which the journey begins.
2. **Enter the destination city**: The city to which the user wants to travel.
3. **Select the search algorithm**: Choose either **A* or UCS**.
4. **Choose the heuristic**: Select either **admissible** or **inadmissible**.

After receiving the user inputs, the program calculates and displays the optimal route and total distance. It also asks if the user wants to find another route.

7. Sample Output

mathematica
Copy

```
Enter the starting city: CityA
Enter the destination city: CityB
Enter the search algorithm (A* or UCS): A*
Enter the heuristic (admissible or inadmissible): admissible
Route: CityA -> CityC -> CityB
Total Distance: 120 km
```

```
Do you want to find another route? (yes/no): no
```

8. Performance Considerations

- **Memory Usage**: The system stores road distance data in memory, which may become inefficient if the number of cities or the size of the data grows significantly.
- **Time Complexity**:
    - *A Search**: Typically O(E log V), where E is the number of edges (roads) and V is the number of cities.
    - **UCS**: O(E log V) as well, but without heuristic guidance, potentially requiring more exploration than A*.

9. Possible Enhancements

- **Error Handling**: Ensure that invalid city names or missing data in the CSV file are handled gracefully.
- **Improved Heuristics**: Experiment with different heuristics to improve performance and guarantee optimal paths.
- **Graphical User Interface (GUI)**: For better user experience, consider implementing a GUI to visualize the cities and the paths found by the algorithms.

10. Conclusion

This route-finding system is a simple yet effective implementation of **A* Search** and **Uniform Cost Search** algorithms. By utilizing different heuristics, the system offers flexibility in how routes are calculated. It can be expanded to handle larger data sets and incorporate more advanced features like path visualization.