Smart Evacuation Route Selector Using Heuristic Search

1 Problem Description

Heuristic search algorithms are widely used in artificial intelligence to efficiently solve pathfinding and planning problems. Applications include robotics navigation, game AI, evacuation planning, and route optimization. These problems are commonly modeled as graphs, where nodes represent states or locations and edges represent actions with associated costs.

In this project, the objective is to compute an optimal or near-optimal evacuation route from a designated start location to an exit location within a building. The building is modeled as a weighted directed graph, and different search strategies determine which node should be expanded next. This mirrors real-world decision-making systems in which a priority queue (frontier) is continuously reordered according to a heuristic rule.

Students will implement and compare multiple heuristic search algorithms, observing how different evaluation functions affect the order of node expansion, the quality of the solution, and the overall efficiency of the search.

2 Search Algorithms

The system must support the following search strategies:

- **Uniform Cost Search (UCS)**: Expands the node with the lowest path cost from the start.
- **Greedy Best-First Search**: Expands the node with the lowest heuristic estimate to the goal.
- **A* Search**: Expands the node with the lowest evaluation value $f(n) = g(n) + h(n)$.

Here, $g(n)$ denotes the path cost from the start node to node n, and $h(n)$ denotes a heuristic estimate of the remaining cost from n to the goal.

3 Graph Model

The environment is modeled as a directed weighted graph:

- Nodes represent rooms or intersections.
- Directed edges represent traversable paths.
- Each edge has a non-negative integer cost.

A heuristic value $h(n)$ is provided for each node, representing an estimate of the distance from that node to the goal.

4 Input Specification

The input consists of a single line with the following format:

```
<algorithm> | <start> -> <goal> | E:<edge_list> | H:<heuristic_list>
```

Algorithm

One of the following strings:

- `ucs`
- `greedy`
- `astar`

Start and Goal

A pair of node identifiers formatted as:

```
<start> -> <goal>
```

Edge List

The edge list is prefixed by `E:` and consists of semicolon-separated triples:

```
u,v,w
```

where u is the source node, v is the destination node, and w is the edge cost.

Heuristic List

The heuristic list is prefixed by `H:` and consists of semicolon-separated pairs:

```
node:heuristic_value
```

Heuristic values are non-negative integers. For UCS, heuristic values are ignored but must still be parseable.

5 Output Specification

The output consists of a single line.

If a path exists from the start node to the goal node, output:

```
PATH:<node_1->node_2->...->goal> COST:<total_cost>
```

If no path exists, output:

```
NO_PATH
```

6 Example

Input

```
astar | A -> F | E:A,B,2;A,C,5;B,D,4;C,D,1;D,F,3;C,F,20 |
                    H:A:7;B:6;C:4;D:2;F:0
```

Output

```
PATH:A->C->D->F COST:9
```

7 Constraints

- The number of nodes is finite and at most 10^4.
- All edge costs are non-negative.
- Heuristic values are non-negative.
- Node identifiers are unique strings.

8 Learning Objectives

Upon completing this project, students will:

- Understand frontier-based search and priority queue behavior.
- Implement and compare UCS, Greedy, and A* search algorithms.
- Analyze the effect of heuristic quality on search performance.
- Observe conditions for optimality and completeness.
- Apply AI search techniques to real-world-inspired problems.