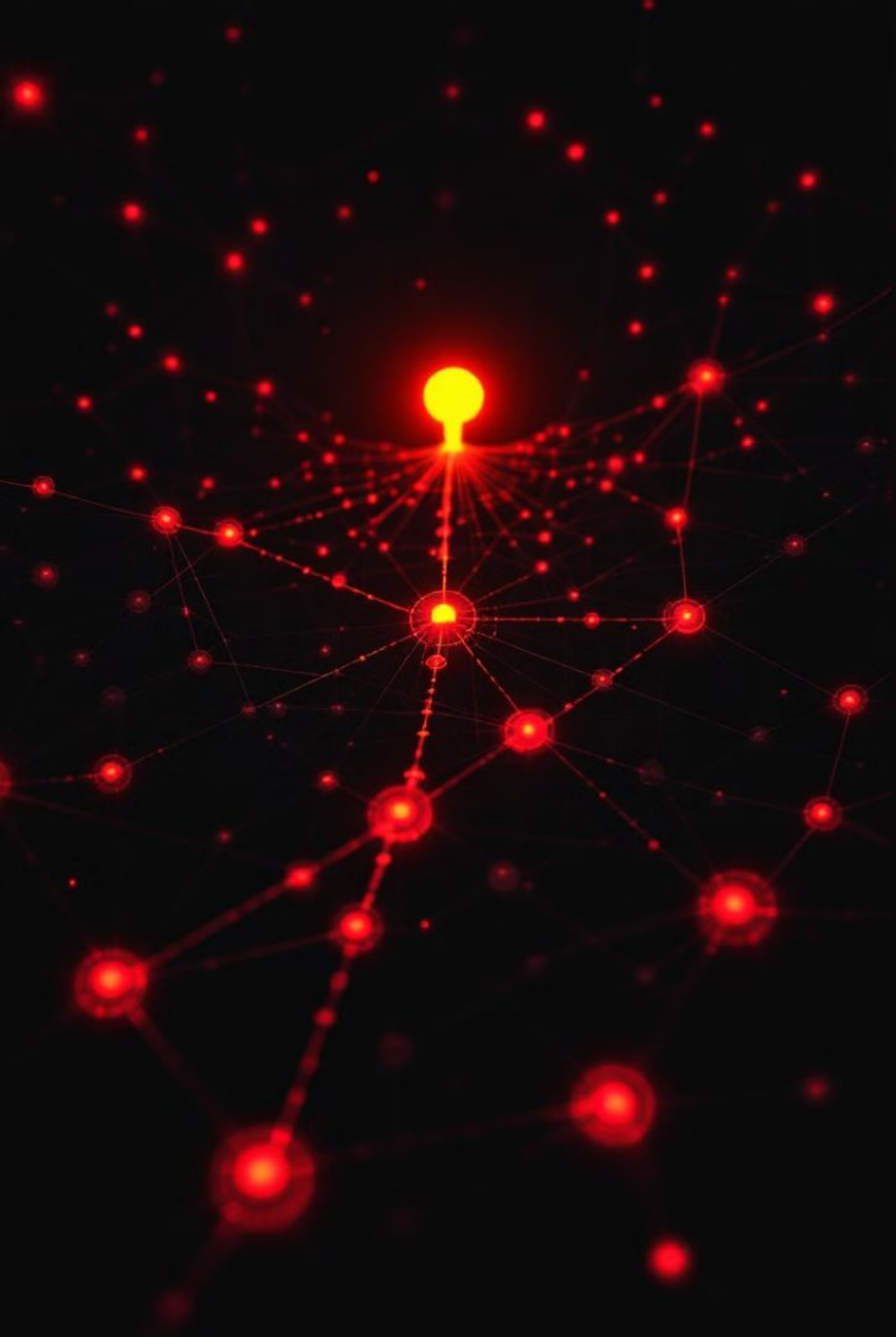


Exploring Informed Search Strategies in Artificial Intelligence

A Comprehensive Overview of Heuristic Approaches and Algorithms

DM Group B3

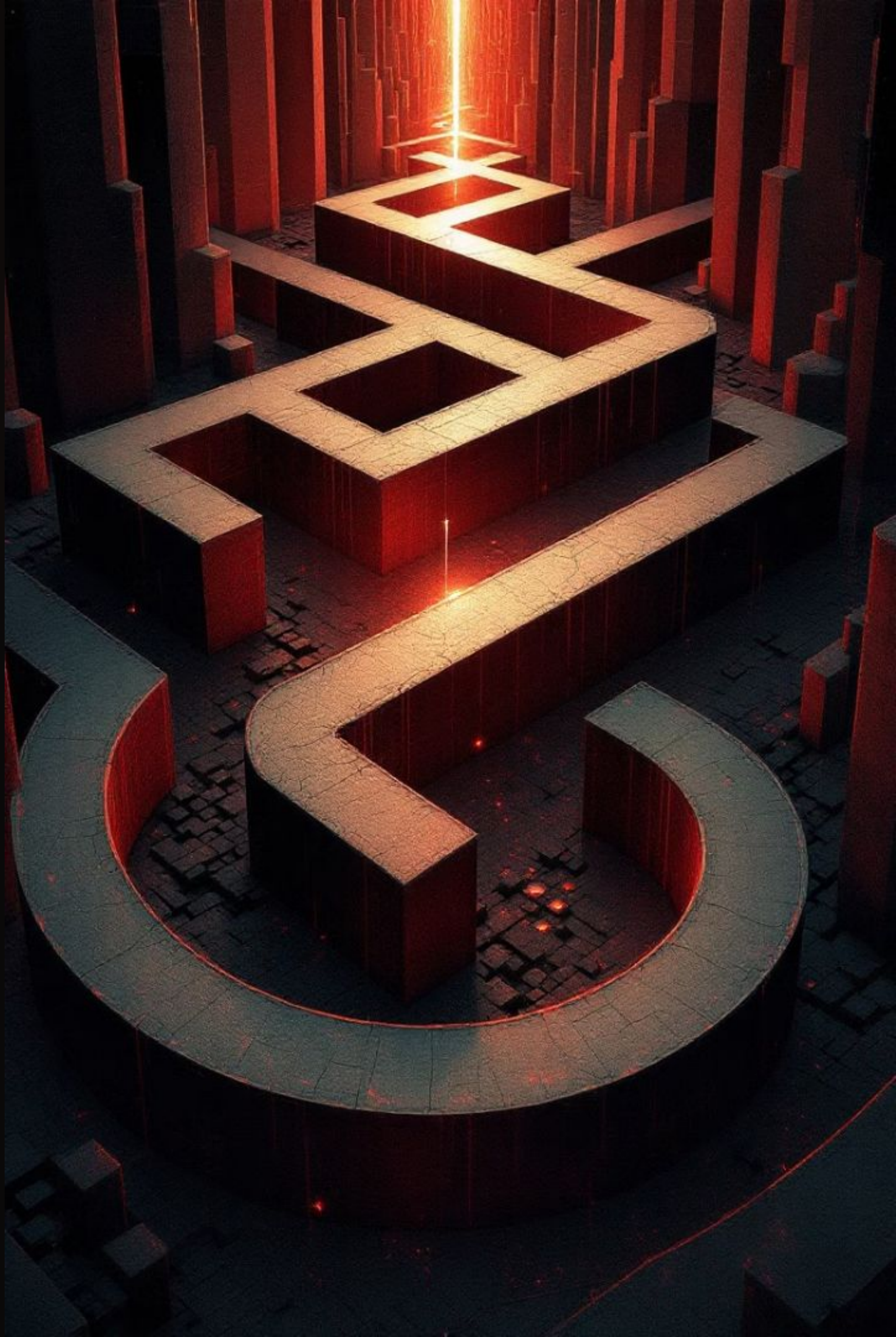
Introduction to Informed Search Strategies

1 **Heuristic Functions**

Informed search strategies use heuristic functions. These functions estimate the cost to reach a goal from a node, enhancing search efficiency over uninformed strategies.

2 **Essential for Optimization**

They are essential for optimizing problem-solving in many areas, such as route finding and game playing.



Heuristic Function in Search Algorithms

Role in Search Algorithms

- Functions that estimate the cost from a given state to the goal.
- Play a crucial role in guiding search algorithms towards more promising paths.
- Help improve the efficiency of search strategies by reducing the number of nodes explored.
- Facilitate faster problem-solving by prioritizing paths that are likely to lead to the goal.

Examples of Heuristics

- Straight-line distance in route-finding problems.
- Domain-specific heuristics tailored to particular problem types.

Main Types of Heuristic Functions

1. **Admissible Heuristics**
    - Never overestimate the actual cost to the goal.
    - Example: **Manhattan Distance** in the 8-puzzle.
2. **Consistent Heuristics**
    - Always satisfies h(n)≤c(n,n′)+h(n′)h(n) \leq c(n, n') + h(n')h(n)≤c(n,n′)+h(n′), ensuring efficiency in search algorithms.
3. **Straight-Line Distance Heuristic**
    - Used in pathfinding problems (e.g., Romania Map Problem).
    - Example: **Euclidean Distance** from a city to Bucharest.
4. **Domain-Specific Heuristics**
    - Custom heuristics tailored for specific problems.
    - Example: **Misplaced Tiles Heuristic** in puzzle-solving.

Greedy Best-First Search

Algorithm Overview

Greedy Best-First Search is a pathfinding algorithm that expands the node with the lowest heuristic value first. It aims to find a solution quickly by prioritizing nodes that appear closest to the goal.

Evaluation Function

The evaluation function for Greedy Best-First Search is defined as f(n) = h(n), where h(n) is the heuristic estimate of the cost to reach the goal from the current node. This function guides the search towards the most promising paths.

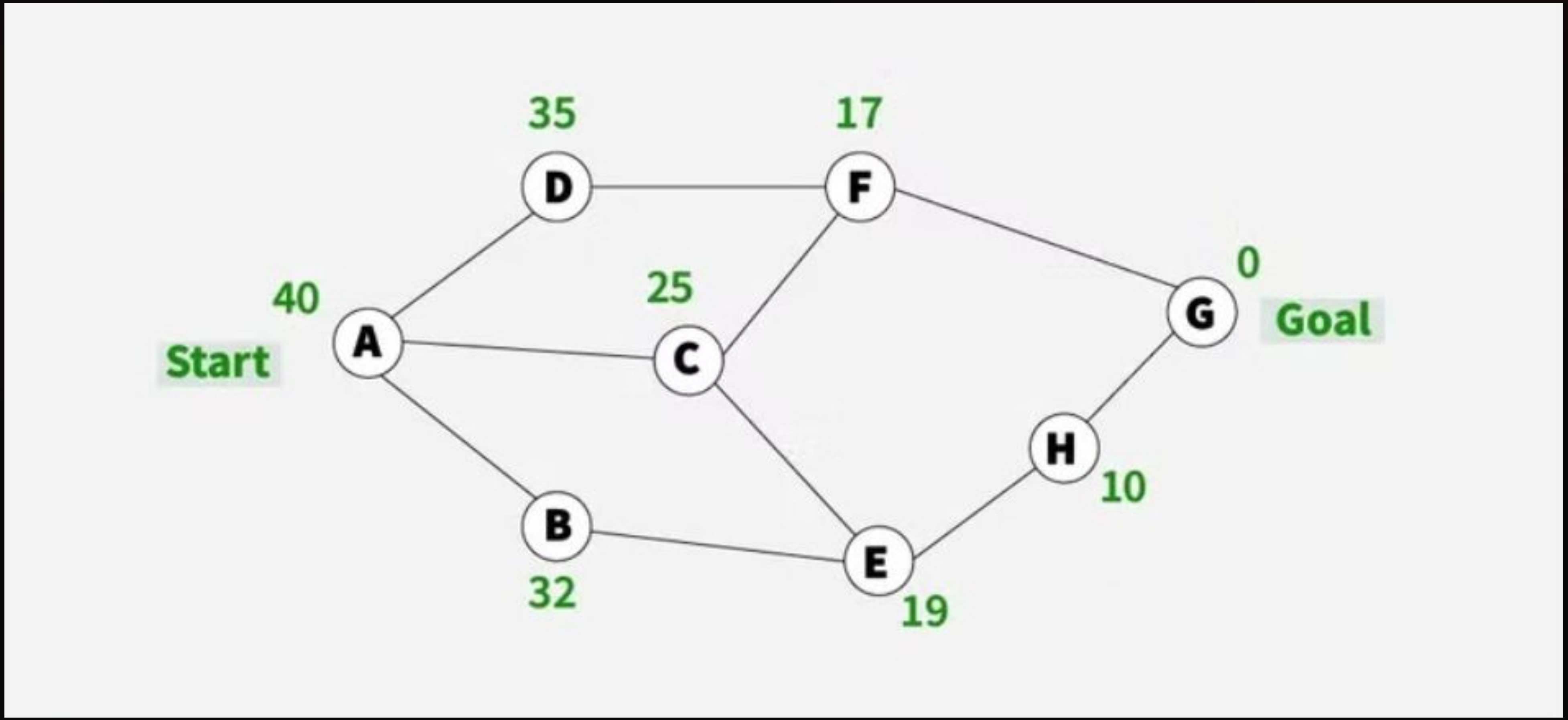
Example Problem

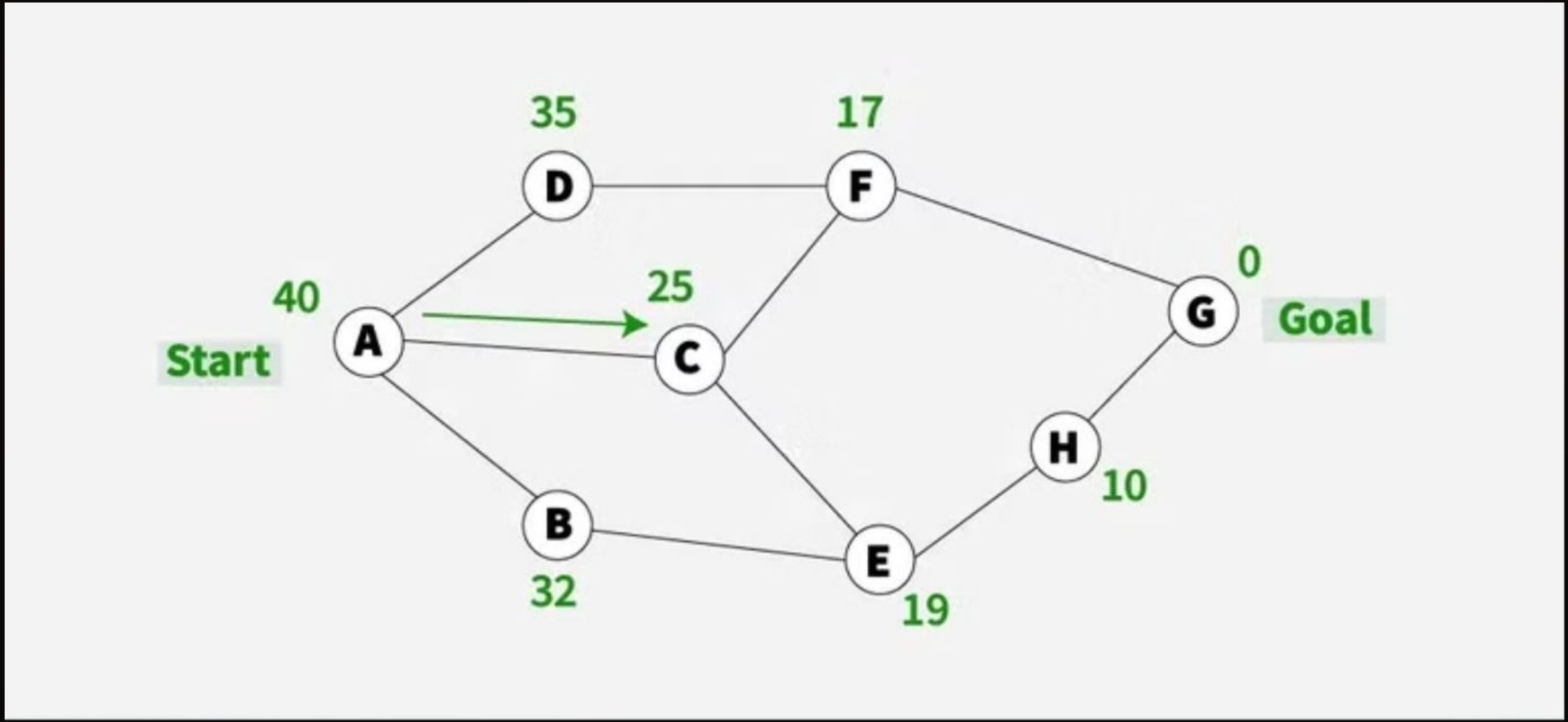
Consider a search problem where the goal is to navigate from Arad to Bucharest. The algorithm uses the straight-line distance as a heuristic to determine which neighboring city to explore next.

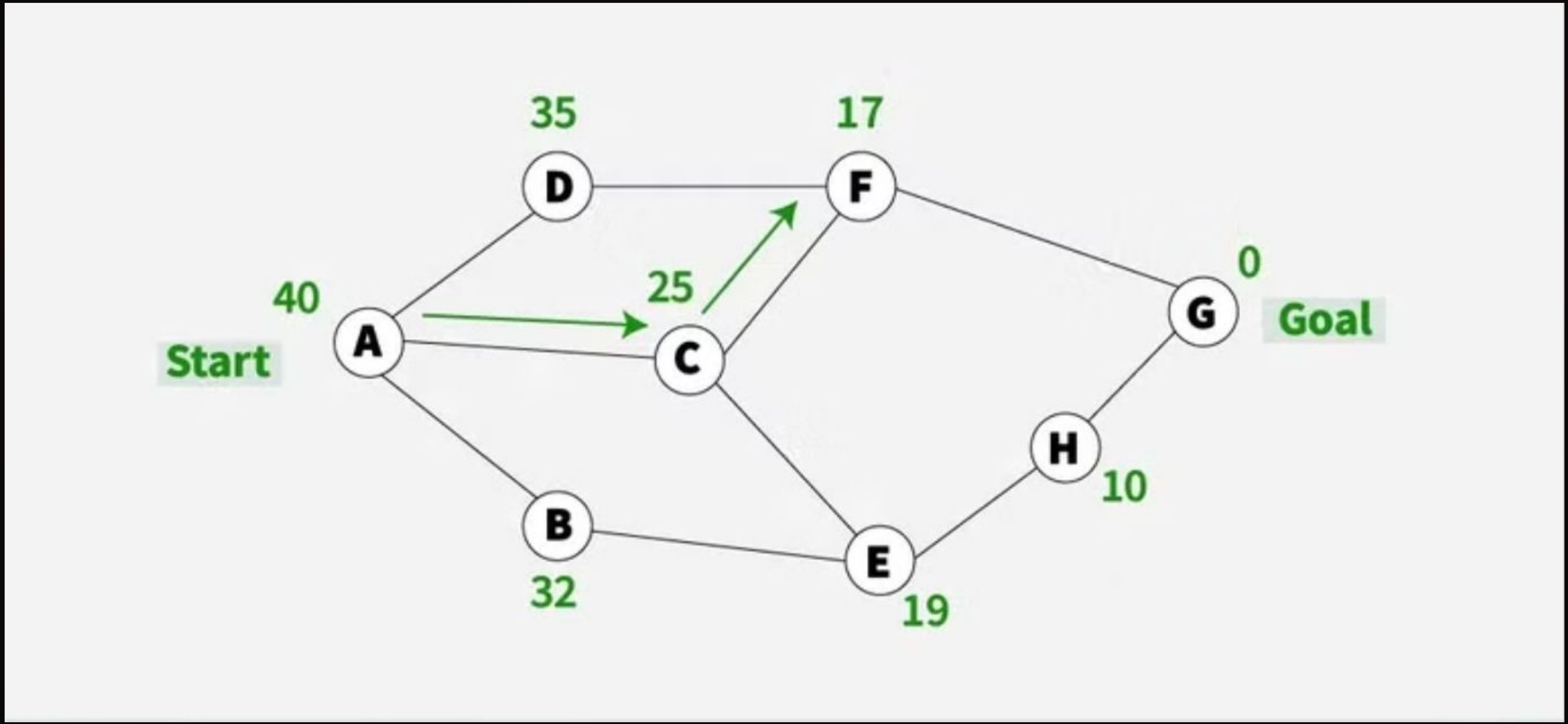
Heuristic Application

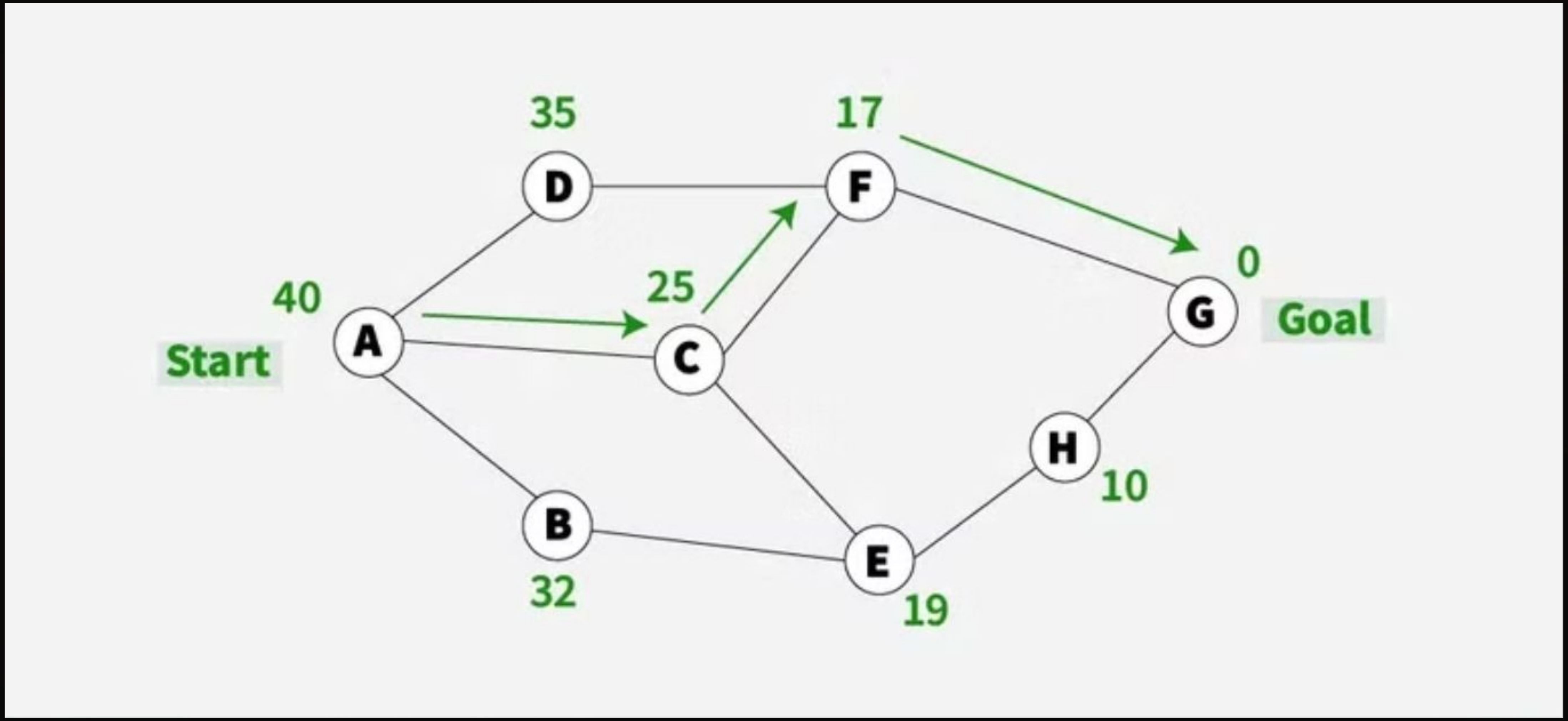
In this example, the algorithm evaluates each city based on its straight-line distance to Bucharest, expanding the city that is closest to the goal. This approach allows the algorithm to efficiently find a path to Bucharest.

An example of the best-first search algorithm is below graph, suppose we have to find the path from A to G




35
17
D
F
40
25
0
Start
A
C
G
Goal
H
10
B
E
32
19


35
17
D
F
40
25
0
Start
A
C
G
Goal
H
10
B
E
32
19


35
17
D
F
40
25
0
Start
A
C
G
Goal
H
10
B
E
32
19

A* Search Algorithm

1

Introduction to A* Search Algorithm

A* is a widely used informed search algorithm that combines features of Dijkstra's algorithm and greedy best-first search. It aims to find the most cost-effective path to a goal using a heuristic.

2

Evaluation Function

The evaluation function is defined as f(n) = g(n) + h(n). Here, g(n) represents the cost to reach the current node, and h(n) is the heuristic estimate of the cost to reach the goal.

3

Heuristic Types/ Characteristics

Admissible heuristics never overestimate the true cost to reach the goal, ensuring optimality. Consistent heuristics maintain that the estimated cost is always less than or equal to the cost from any adjacent node plus the step cost.

4

Node Expansion Process

A* expands nodes based on their f(n) values, prioritizing those with the lowest cost. This process involves generating child nodes from a parent node and evaluating their potential using the evaluation function.

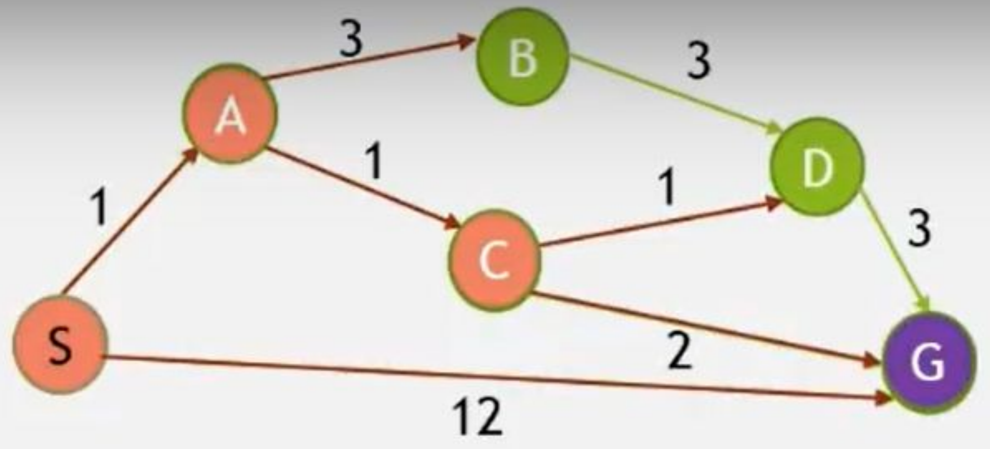
5

Example Problem: 8-Puzzle

In the 8-puzzle problem, the goal is to arrange tiles in a specific order using minimal moves. Heuristics like Manhattan distance can effectively guide the A* search to find the optimal solution.

g(n)=cost of getting to node from start state

h(n)=heuristic function



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

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

Queue={ [S->A->C,4] , [S->A->B,10] , [S->G,12] }

Queue={ [S->A->C->G,4] , [S->A->C->D,6] , [S->G,12] , [S->A->B,10] , [S->G,12] }

Final path : S->A->C->G with cost = 4

f(S)=g(S)+h(S)
=(0)+4
=4

f(A)=g(A)+h(A)
=(1)+2
=3

f(G)=g(G)+h(G)
=(12)+0
=12

f(C)=g(C)+h(C)
=(1+1)+2
=4

f(B)=g(B)+h(B)
=(1+3)+6
=10

f(D)=g(D)+h(D)
=(1+1+1)+3
=6

f(G)=g(G)+h(G)
=(1+1+2)+0
=4

Search Contours and Visualization

Visualizing Search Contours

Search contours represent the progression of a search algorithm. They illustrate how nodes are expanded based on their cost values.

Examples of Contour Maps

Contour maps resemble topographic maps, showing cost levels in a search space. For example, in pathfinding scenarios, they can depict costs from a starting point to various destinations.

Understanding the Purpose of Contours

Contours help visualize the efficiency of a search algorithm. They indicate how the algorithm prioritizes paths based on estimated costs.

Advanced Search Techniques

Weighted A* Search

Weighted A* search modifies the traditional A* algorithm by applying a heavier weight to the heuristic value, allowing for faster search times. The evaluation function is defined as $f(n) = g(n) + W \times h(n)$, where $W > 1$. While it may find solutions more rapidly, it may not always yield the optimal path.

Memory-Bounded Search Techniques

- Beam Search: Keeps only the top-scoring nodes in the search area.
- Iterative-Deepening A*: Combines the strengths of A* search with memory efficiency.
- Recursive Best-First Search: Operates like standard best-first search but uses linear memory.

Bidirectional Heuristic Search

Bidirectional search can significantly reduce the number of nodes expanded compared to unidirectional search. It combines costs from both forward and backward searches, often finding optimal paths more quickly, especially with average heuristics.

THE END