UAV Formation Base Path Planning

Presented by Group 1

TEAM MEMBERS

Ankit Kholwad

Arsh Chauhan

Harshil Goel

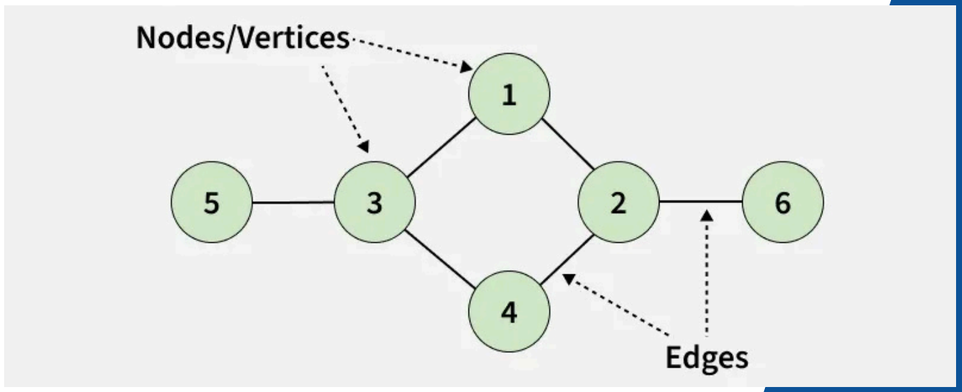
Vinayak Chandra Srivastava

WHAT IS A GRAPH?

- A graph models relationships between entities

- Two main components:
  a. Vertices (nodes): entities or states
  b. Edges: connections or transitions

- Graphs appear everywhere:
  a. Road networks
  b. Social networks
  c. Computer networks
  d. State spaces in AI and robotics



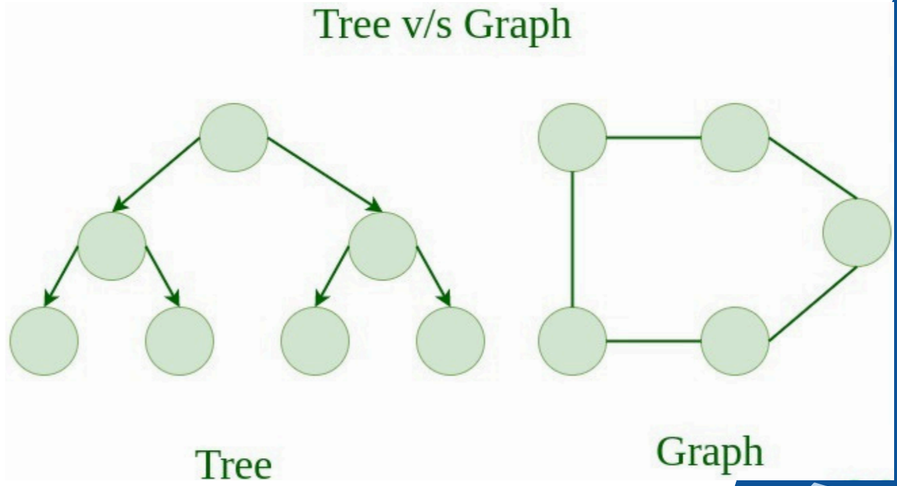
TYPES OF GRAPHS

- Directed vs Undirected
  - Undirected: edges work both ways (roads)
  - Directed: edges have direction (one-way streets, followers)
- Unweighted vs Weighted
  - Unweighted: all edges equal cost
  - Weighted: edges have costs (distance, time, energy)
- Sparse vs Dense (idea level)
  - Sparse: few edges per node
  - Dense: many edges per node

TREES VS GENERAL GRAPHS

- Tree:
  - No cycles
  - Exactly one unique path between any two nodes
  - N nodes → N–1 edges

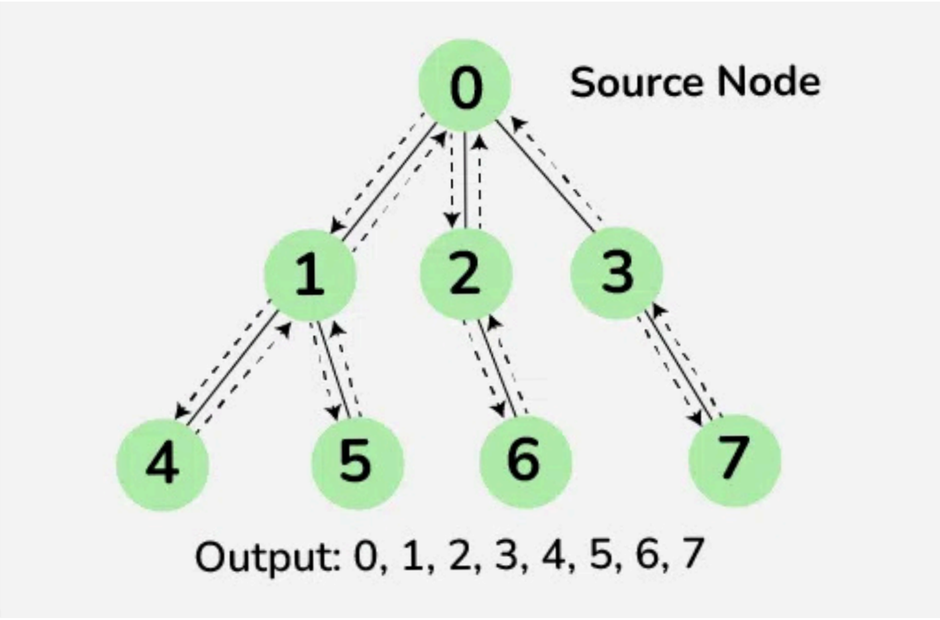
- General graph:
  - May contain cycles
  - Multiple paths between nodes



Why this matters:

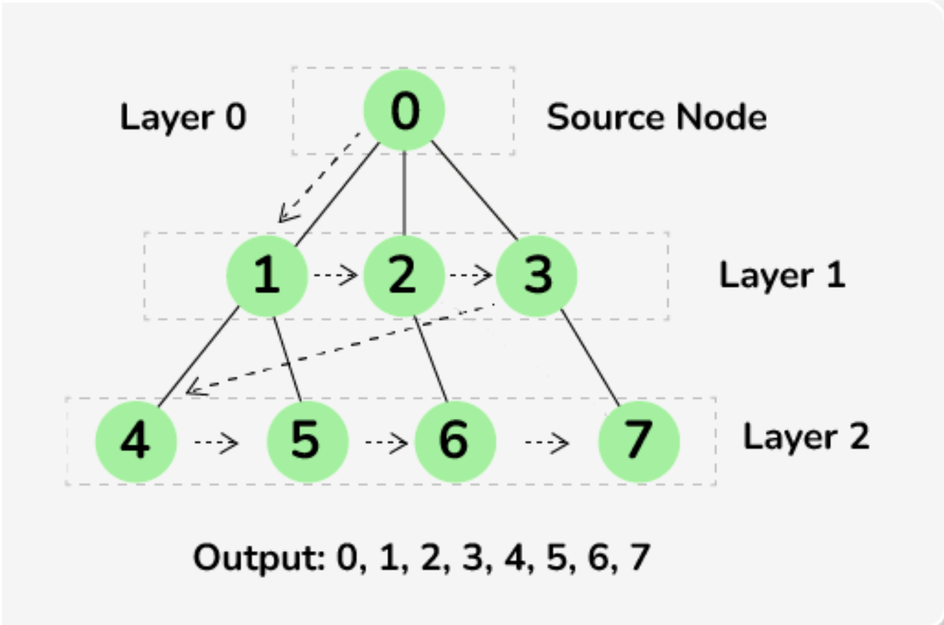
1. Pathfinding is harder when many paths exist
2. Need systematic traversal algorithms

DEPTH FIRST SEARCH (DFS)



- Explores as far as possible along each branch before backtracking.
- Uses a stack (or recursion).

BREADTH FIRST SEARCH (BFS)



- Explores all neighbors of a vertex before moving to the next level.
- Uses a queue.

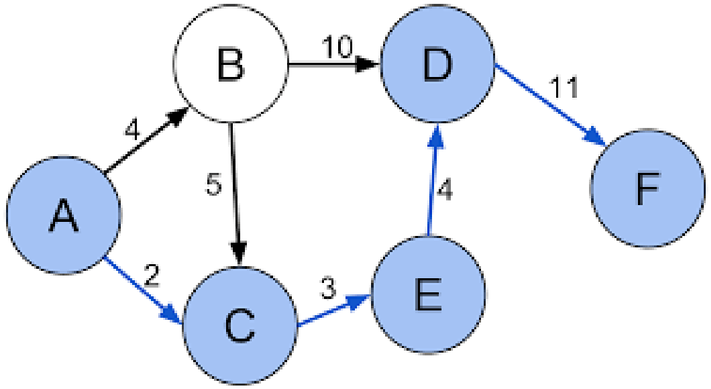
BFS VS DFS

- DFS:
  - Deep, narrow exploration
  - Memory efficient
  - Not optimal for paths
- BFS:
  - Wide, level-based exploration
  - More memory
  - Optimal only for unweighted graphs

Key insight: Real-world problems often have unequal costs.

NEED FOR WEIGHTED SHORTEST PATH ALGORITHMS

- In many problems, edges have different costs:
  - Distance
  - Time
  - Energy
- BFS treats all edges equally → incorrect results
- We need an algorithm that:
  - Handles non-uniform weights
  - Guarantees optimal paths



This leads to **Dijkstra's Algorithm**

DIJKSTRA'S ALGORITHM

1. Solves shortest paths in graphs with non-negative weights
2. Core idea:
   - Always expand the node with smallest known distance
3. Uses:
   - Priority queue (min-heap)
4. Guarantees optimality

A* ALGORITHM

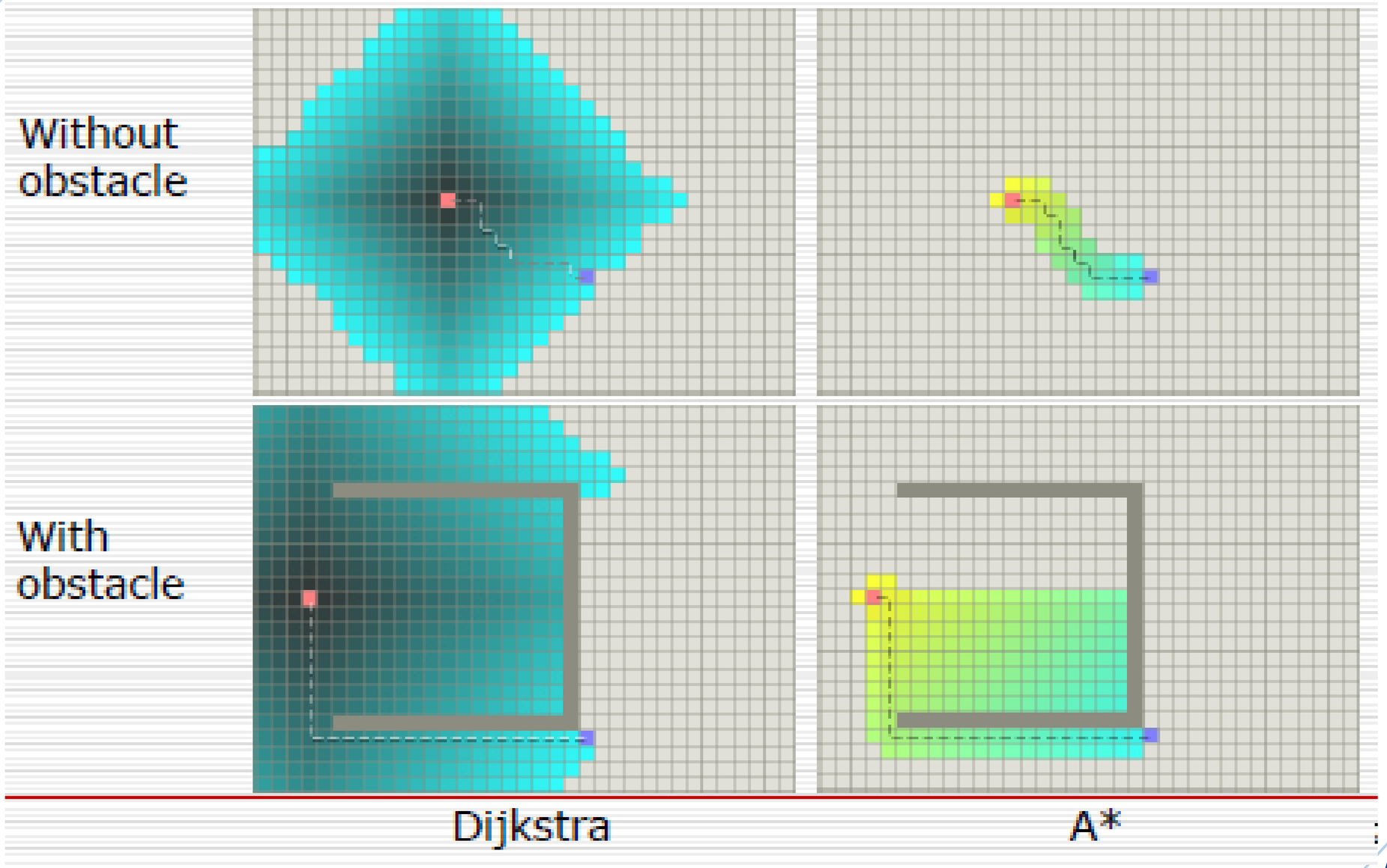
1. Solves shortest paths in graphs using cost + heuristic
2. Core idea:
3. Always expand the node with smallest f(n)=g(n)+h(n) (where g(n) = cost so far, h(n) = heuristic estimate to goal)
4. Uses: Priority queue (min-heap), heuristic function
5. Guarantees:

- Optimal if the heuristic is admissible (never overestimates)
- More efficient than Dijkstra when the heuristic is informative

WHY A* IMPROVES ON DIJKSTRA ?

- Dijkstra explores uniformly in all directions
- Often unnecessary when goal is known
- A* adds a heuristic:
  - Estimates distance to goal
  - Guides search efficiently

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


Without obstacle
With obstacle
Dijkstra
A*

KEY RESULTS & TAKEAWAYS

- BFS & DFS: foundational but limited
- Dijkstra: optimal for weighted graphs
- A*: optimal and efficient with good heuristics
- Grid planning is practical but approximate

Final message: Graph theory concepts directly power real-world path planning.

THANK YOU

For your attention