1. Breadth-First Search (BFS)

- **Behavior**:
  Explores the graph level by level from the start node. It guarantees finding the shortest path in unweighted graphs.
- **Computational Complexity**:
  - **Time**: O(V + E)
    (V = number of vertices, E = number of edges)
  - **Space**: O(V) for the queue and visited set.
- **Suitability**:
  - Best for unweighted graphs where the shortest path (fewest edges) is needed.
  - Performs well on sparse and moderately dense graphs.
  - Can be slow or memory-heavy on large graphs due to broad exploration.

---

2. Depth-First Search (DFS)

- **Behavior**:
  Explores as far as possible along each branch before backtracking. Does **not** guarantee the shortest path.
- **Computational Complexity**:
  - **Time**: O(V + E)
  - **Space**: O(V) for the recursion stack or explicit stack.
- **Suitability**:
  - Good for searching through all possible paths or detecting cycles.
  - Not recommended when you strictly need the shortest path.
  - May perform better in deep, narrow graphs but worse in wide graphs.

---

3. Bidirectional Search

- **Behavior**:
  Runs two simultaneous searches — one forward from the start and one backward from the goal — meeting in the middle.
- **Computational Complexity**:
  - **Time**: O(b^(d/2)) where b = branching factor, d = depth of solution.
  - **Space**: O(b^(d/2)) for the two frontiers.
- **Suitability**:
  - Excellent for large, undirected, and unweighted graphs where start and goal are far apart.
  - Dramatically reduces the search space compared to BFS.
  - More complex to implement and not useful if the goal node is unknown or the graph is directed.

Documentation Summary

Project Description

This project visualizes three pathfinding algorithms (BFS, DFS, Bidirectional Search) on unweighted graphs, providing insights into their runtime behavior, path length, and nodes visited.

Code Highlights

- **BFSVisualizer, DFSVisualizer, BidirectionalVisualizer**:
  Classes responsible for running and visualizing each algorithm.
- **GraphGenerator**:
  Creates random graphs or grid graphs for testing using Matplotlib
- **Performance Measurement**:
  Tracks nodes visited, execution time, and path length.
- **Visualization**:
  Uses Matplotlib to animate search progress.