Lab01 - Search

Depth-first search (DFS)'

Idea

Depth-First Search (DFS) is a graph traversal algorithm that explores deeply before backtracking. It starts at a node, visits adjacent nodes recursively, and uses a stack.

Pros

- **Simplicity and Efficiency:** DFS is simple and efficient for exploring deep paths, suitable for problems like maze-solving or finding paths.

Cons

- **Risk of Infinite Loop:** There is a risk of falling into an infinite loop if not managed carefully.
- **Non-Optimal:** Does not guarantee finding the shortest path, as it may search through unnecessary routes.

Breadth-first search (BFS)

Idea

Breadth-First Search (BFS) is a graph traversal algorithm that explores nodes level by level. It starts at the initial node, visits all its neighbors before moving on to their neighbors, and uses a queue to manage exploration.

Pros

- **Optimality:** BFS ensures the shortest path is found in unweighted graphs.
- **No Risk of Infinite Loop:** Unlike DFS, BFS doesn't risk getting stuck in infinite loops.

Cons

- **Memory Usage:** May require more memory compared to DFS, especially in large graphs.
- **Less Suitable for Deep Paths:** In scenarios where deep exploration is needed, BFS might not be as efficient as DFS.

Uniform cost search (UCS)

Idea

Uniform Cost Search (UCS) is a graph traversal algorithm focused on finding the path with the lowest cost. It starts at the initial node and expands nodes based on the cumulative cost from the start node. UCS utilizes a priority queue to manage exploration.

Pros

- **Optimal Path:** UCS guarantees finding the optimal path in terms of the lowest cost.
- **Adaptability to Edge Weights:** Well-suited for graphs with varying edge weights.

Cons

- **Complexity:** UCS can be computationally intensive, especially in graphs with high branching factors.
- **Memory Usage:** May require significant memory, as it keeps track of the cost for each explored node.

Iterative deepening search (IDS)

Idea

Iterative Deepening Search (IDS) is an algorithm that combines the benefits of depth-first search and breadth-first search. It uses depth-first tree search as a core component but avoids loops by checking a new node against the current path. IDS gradually increases the depth limit until the goal is found.

Pros

- **Completeness:** IDS is complete, ensuring a solution is found if one exists.
- **Memory Efficiency:** Consumes less memory compared to traditional depth-first or breadth-first searches.

Cons

- **Redundant Exploration:** May redundantly explore nodes at different depths, affecting efficiency.
- **Limited to Solvable Problems:** Like depth-first search, IDS may not terminate for unsolvable problems or those with infinite paths.

Greedy Best First Search (GBFS) with h = edge weight

Idea

Greedy Best-First Search (GBFS) with heuristic h, where h is the edge weight, is a graph traversal algorithm that prioritizes nodes based on their estimated distance to the goal. It selects the node with the lowest heuristic value and explores its neighbors. This process continues until the goal is reached.

Pros

- **Heuristic Guidance:** GBFS uses heuristics to make informed decisions, often leading to efficient paths.
- **Fast in Practice:** Can be fast and effective, especially in scenarios where a good heuristic is available.

Cons

- **Lack of Optimality:** GBFS does not guarantee finding the optimal solution; it may settle for a locally optimal path.
- **Sensitivity to Heuristic Quality:** The quality of the heuristic significantly impacts the algorithm's performance.

A* with h = eclidean_distance(pos[current vertex], pos[Goal]))

Idea

A* (A-star) with heuristic h = Euclidean distance between the current vertex and the goal is a graph traversal algorithm designed to find the optimal path. It combines the advantages of both Dijkstra's algorithm and greedy best-first search by considering both the actual cost from the start and the heuristic estimate to the goal.

Pros

- **Optimality:** A* guarantees finding the optimal path by considering both actual and estimated costs.
- **Efficiency:** The heuristic helps guide the search towards promising paths, often leading to faster convergence.

Cons

- **Sensitivity to Heuristic Quality:** The effectiveness of A* depends on the quality of the heuristic.
- **Memory Usage:** A* can consume significant memory resources, especially in large graphs with many nodes.