

Breadth-first and depth-first search

Lesson 71 · Module 5 — Graph Theory: Foundations · Discrete Mathematics · dailymaths.uk/discrete

Practice problems

1. Given a graph with vertices $V = \{1, 2, 3, 4, 5\}$ and edges $E = \{(1, 2), (1, 3), (2, 4), (3, 4), (3, 5)\}$, what is the order of discovery using Breadth-First Search starting from node 1? Assume neighbors are visited in increasing numerical order.
2. Using the same graph $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (1, 3), (2, 4), (3, 4), (3, 5)\}$, what is the order of discovery using Depth-First Search starting from node 1? Assume neighbors are visited in increasing numerical order.
3. In a graph with 100 nodes, if you are looking for a target node that is known to be very deep in the graph (far from the start), which algorithm is generally more memory-efficient, and why?

Solutions

1. Given a graph with vertices $V = \{1, 2, 3, 4, 5\}$ and edges $E = \{(1, 2), (1, 3), (2, 4), (3, 4), (3, 5)\}$, what is the order of discovery using Breadth-First Search starting from node 1? Assume neighbors are visited in increasing numerical order.

- Start at node 1. Mark 1 as visited. Queue: [1].
- Pop 1. Neighbors are 2 and 3. Add them to queue. Queue: [2, 3]. Visited: {1, 2, 3}.
- Pop 2. Neighbor is 4. Add to queue. Queue: [3, 4]. Visited: {1, 2, 3, 4}.
- Pop 3. Neighbors are 4 and 5. 4 is visited, so add 5. Queue: [4, 5]. Visited: {1, 2, 3, 4, 5}.
- Pop 4. No unvisited neighbors.
- Pop 5. No unvisited neighbors.

Answer: 1, 2, 3, 4, 5

2. Using the same graph $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (1, 3), (2, 4), (3, 4), (3, 5)\}$, what is the order of discovery using Depth-First Search starting from node 1? Assume neighbors are visited in increasing numerical order.

- Start at node 1. Mark 1 as visited.
- Visit neighbor 2. Mark 2 as visited.
- From 2, visit neighbor 4. Mark 4 as visited.
- From 4, visit neighbor 3. Mark 3 as visited.
- From 3, visit neighbor 5. Mark 5 as visited.
- No more unvisited neighbors; backtrack to 3, then 4, then 2, then 1.

Answer: 1, 2, 4, 3, 5

3. In a graph with 100 nodes, if you are looking for a target node that is known to be very deep in the graph (far from the start), which algorithm is generally more memory-efficient, and why?

- BFS must store the entire 'frontier' of the search. In a graph with a high branching factor, the number of nodes in the queue grows exponentially with depth.
- DFS only needs to store the current path from the start node to the current node, plus the untried neighbors of nodes on that path.
- Since the target is deep, BFS would likely store a massive number of nodes at intermediate levels before reaching the target.
- DFS only stores a linear amount of data relative to the maximum depth of the graph.

Answer: DFS, because it only stores the current path (linear space) rather than the entire frontier (exponential space).