Foundations of Artificial Intelligence

Search Algorithms

Prepared by: Mrs. AMRANE Leila

3rd Year Engineering

2025–2026

Definitions and Terminology

- A **search problem** consists of exploring a state space to reach a goal (i.e., search in the state space).

Examples: pathfinding (GPS), solving the 8-puzzle, planning a sequence of actions for a robot.

- A **search algorithm** is an algorithm that solves a search problem. Solving the problem means finding a path in the graph.

A **search** is a set of steps carried out to solve a problem and involves the following elements:

1. **Initial state**: the state from which the search begins.
2. **Search space**: the set of all possible states the system can be in.
3. **Goal function**: a function that tests whether the current state is a goal state.
4. **Action(s)**: an operation that allows moving from one state to another during the search.
5. **Successor function**: indicates the possible successor states from a given state.
6. **Solution**: a sequence of actions that leads from the initial state to the goal state.
7. **Cost function**: assigns a cost to each action or sequence of actions.
8. **Optimal solution**: a solution with the lowest cost among all possible solutions.

Algorithm Properties

- **Completeness**: guarantees that the algorithm will find a solution if one exists.
- **Optimality**: ensures that the solution found is the best (lowest-cost) one.
- **Time complexity**: the amount of time required to find a solution.
- **Space complexity**: the amount of memory needed during the search.

Search Methods

1. **Blind (Uninformed) Methods**: Depth-First Search (DFS), Breadth-First Search (BFS)
2. **Informed (Heuristic) Methods**: A*, Greedy Best-First Search, Monte Carlo, etc.

Blind Search Methods – Analogy

A brave explorer is searching for a hidden treasure in a large forest. There are many possible paths, but the treasure is hidden at only one location. The explorer has two strategies:

Depth-First Search (DFS)

Advantage: The explorer may find the treasure quickly if it lies at the end of one of the first paths explored.

Disadvantage: If the chosen path is very long and leads nowhere, the explorer wastes a lot of time before backtracking to try another path.

Breadth-First Search (BFS)

Advantage: The explorer will never miss a short path to the treasure.

Disadvantage: It must explore many paths at each level—even useless ones—which can be slow if the treasure is far away.

Conclusion:

If you're in a hurry and don't mind backtracking, DFS might be faster.
If you want to guarantee finding the closest treasure, BFS is safer—but sometimes slower.

How to Keep Track of the Path?

DFS Algorithm Steps

1. Define a **stack** with size equal to the total number of vertices in the graph.
2. Choose any vertex as the starting point, visit it, and push it onto the stack.
3. Visit an adjacent, unvisited vertex of the vertex on top of the stack, and push it onto the stack.
4. Repeat step 3 until no new adjacent vertices are available from the top vertex.
5. When no new vertices can be visited, **backtrack** by popping a vertex from the stack.
6. Repeat steps 3–5 until the stack is empty.
7. When the stack is empty, generate the final spanning tree by removing unused edges.

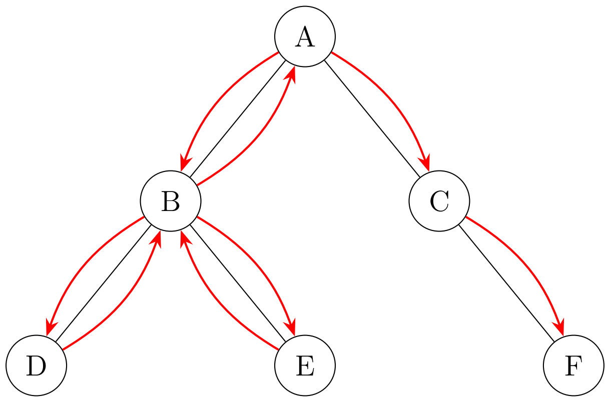

Figure 1: DFS Traversal (Red Arrows)

BFS Algorithm Steps

1. Define a **queue** with size equal to the total number of vertices.
2. Choose any vertex as the start, visit it, and enqueue it.
3. Visit **all** unvisited adjacent vertices of the vertex at the front of the queue, and enqueue them.
4. When no new adjacent vertices remain, dequeue the current vertex.
5. Repeat steps 3–4 until the queue is empty.
6. When the queue is empty, produce the final spanning tree by removing unused edges.

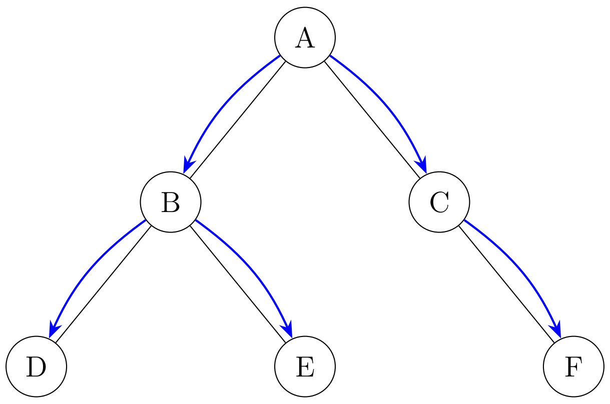

Figure 2: BFS Traversal (Blue Arrows)

.

End of Document