# 3.Types of search.

a. Uninformed(Blind)

Uninformed search algorithms have no additional knowledge about the goal's location except the definition of the problem. They explore blindly.

1. Breadth-First Search (BFS)

Description

* Explores all nodes at one level (depth) before moving to the next.
* Uses a queue (FIFO) structure to maintain nodes at each level.
* Guaranteed to find the shallowest (minimum-step) solution.
* Example: Finding the shortest path in a maze/grid where each move has the same cost.

Properties

* Complete (finds solution if exists)
* Optimal for uniform step cost.
* Time Complexity: O(b^d)
* Space Complexity: O(b^d)

Where b = branching factor, d = depth of shallowest goal

Advantages

* Guaranteed to find the shortest path
* Suitable for small search spaces

Disadvantages

* Memory-intensive; high space requirement
* Not suitable for deep or infinite trees

A diagram of a triangle with letters and numbers

AI-generated content may be incorrect.

1. Depth-First Search (DFS)

Description

* Explores as deep as possible along one branch before backtracking.
* Uses a stack (LIFO) structure or recursion.
* Example: Solving a maze by exploring one path to its deepest level, then trying other paths.

Properties

* Incomplete for infinite/deep spaces
* Not Optimal: If the solution exists beyond the chosen depth limit, DLS won't find it. It can find a solution within the limit, but not necessarily the optimal (shortest or least-cost) one.
* Time Complexity: O(b^d)
* Space Complexity: O(b\*d)

Advantages

* Requires less memory than BFS
* May quickly find deep solutions

Disadvantages

* Can get stuck in infinite loops
* May miss shallower, optimal solutions



1. Depth-Limited Search (DLS)

Description

* DFS with a predefined depth limit to avoid infinite exploration.
* Limits how deep the search can go.
* Example: Searching a chess game tree up to depth 5 to find potential winning moves.

Properties

* Complete if solution within depth limit
* Not Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.
* Time Complexity: O(b^l)
* Space Complexity: O(b\*l), where l = depth limit

Advantages

* Prevents infinite loops
* More space-efficient than BFS

Disadvantages

* May fail to find deeper solutions
* Not guaranteed to be optimal

A diagram of a level of knowledge

AI-generated content may be incorrect.

1. Iterative Deepening Depth-First Search (IDS)

Description

* Combines DFS's space efficiency with BFS's completeness.
* Repeatedly applies DLS, increasing depth limit each time.
* Example: Solving puzzles like the 8-puzzle, progressively searching deeper levels.

Properties

* Complete when the branching factor b is finite.
* Optimal for uniform step cost
* Time Complexity: O(b^d)
* Space Complexity: O(b\*d)

Advantages

* Optimal and complete like BFS
* Space-efficient like DFS
* Finds shallowest solution

Disadvantages

* Repeats nodes multiple times
* Overhead from repeated shallow searches

1

/ \

2 3

/ \ / \

4 5 6 7

Depth limit 0: 1

Depth limit 1: 1 → 2 → 3

Depth limit 2: 1 → 2 → 4 → 5 → 3 → 6 → 7

1. Bidirectional Search

Description

* Performs two simultaneous searches: one from the start, one from the goal.
* Stops when the two search frontiers meet.
* Example: Finding the shortest path between two cities using forward and backward searches.

Properties

* Complete (with BFS in both directions)
* Optimal (for uniform step costs)
* Time Complexity: O(b^(d/2))
* Space Complexity: O(b^(d/2))

Advantages

* Reduces time complexity compared to BFS/DFS
* Effective when both start and goal states are known

Disadvantages

* Complex to implement
* Requires knowledge of goal state