上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 3: Uninformed Search Algorithms

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

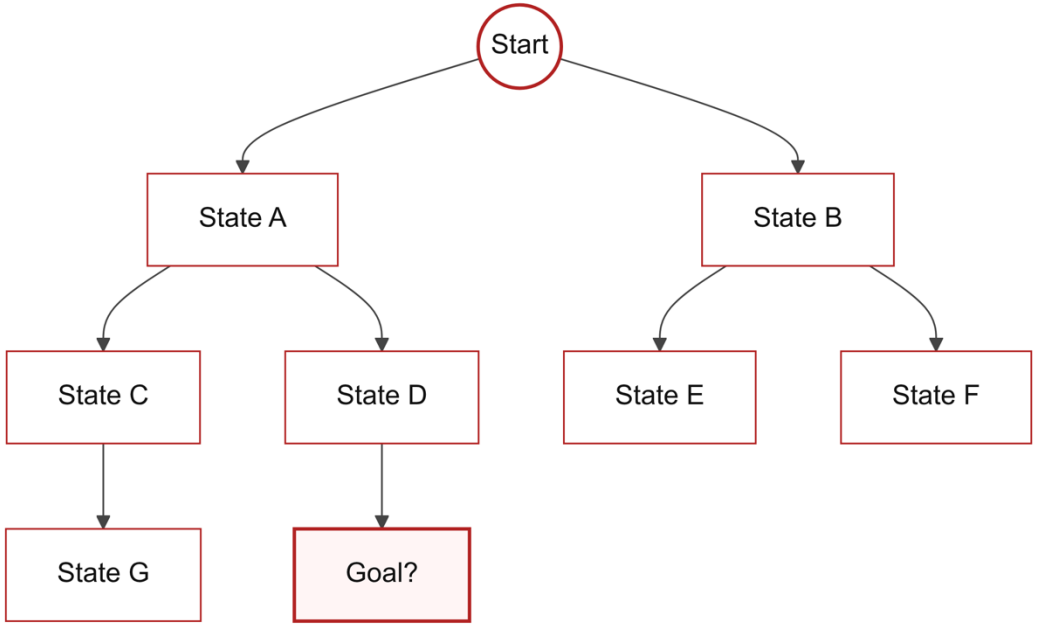
https://oc.sjtu.edu.cn/courses/89538

Review: Search Problems

A search problem defines:

- State space
- Initial state & Goal test
- Actions & Transition model
- Path cost

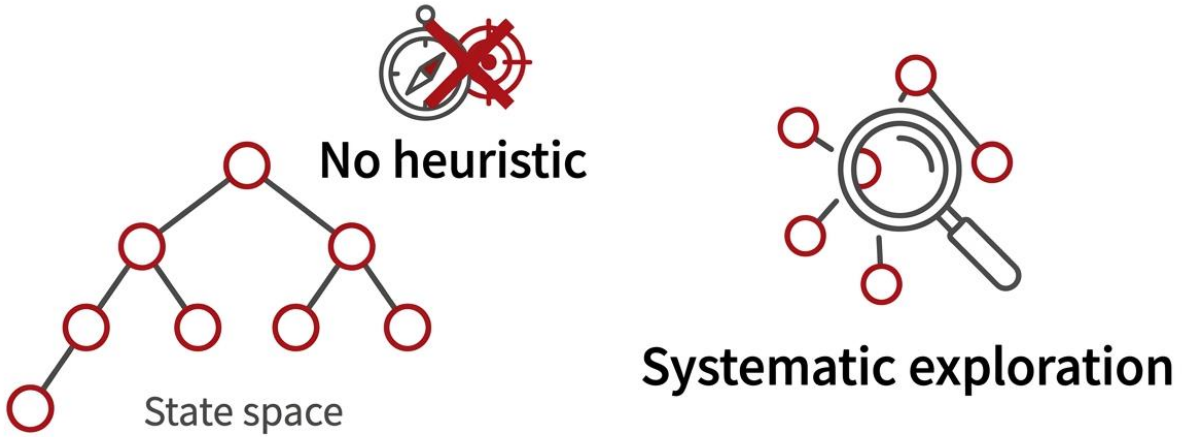
Search algorithms navigate the state space.



State-space exploration tree

Uninformed (Blind) Search

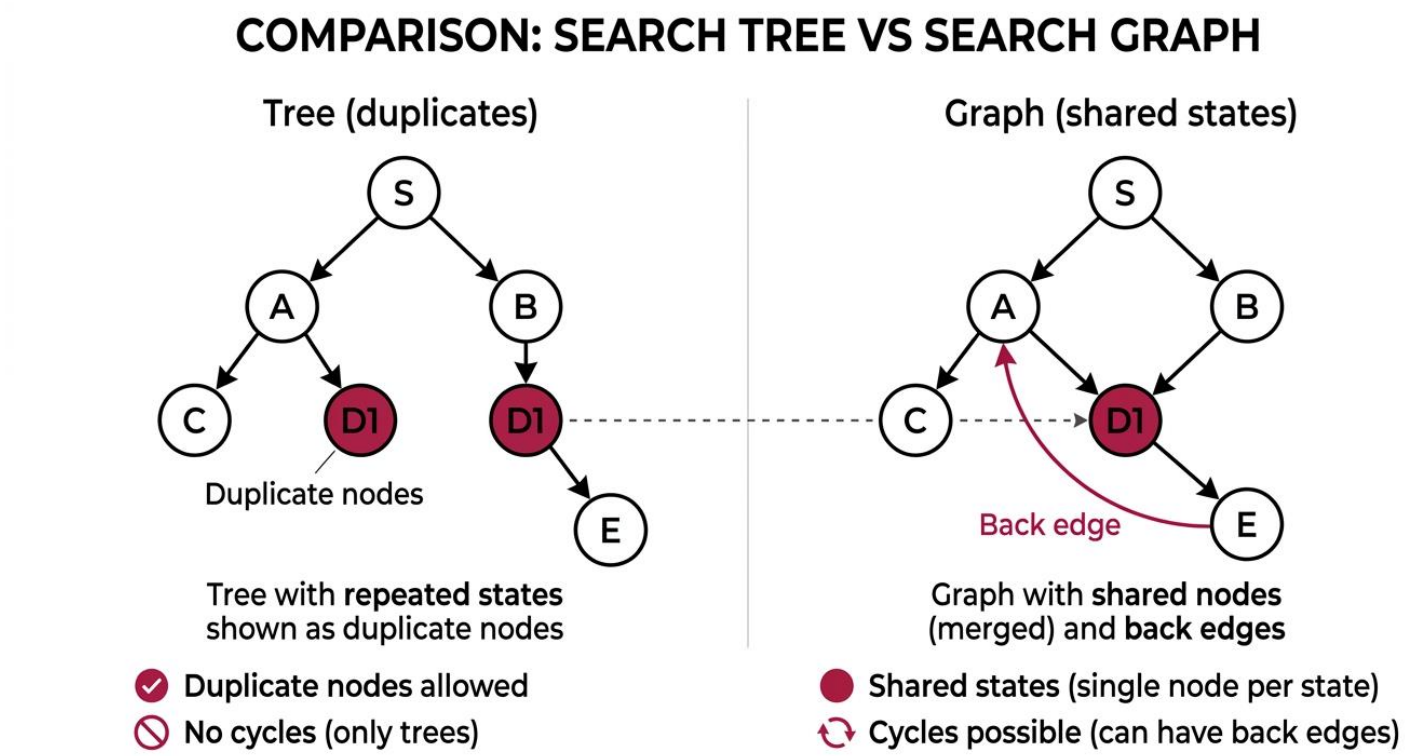
- **No domain knowledge:** Does not know how close a state is to the goal.
- **Algorithm only uses:** Actions, transition model, and goal test.
- **Blind exploration:** Systematic but undirected expansion of the state space.



Expands nodes without knowing which is closer to goal

Search Tree vs. Search Graph

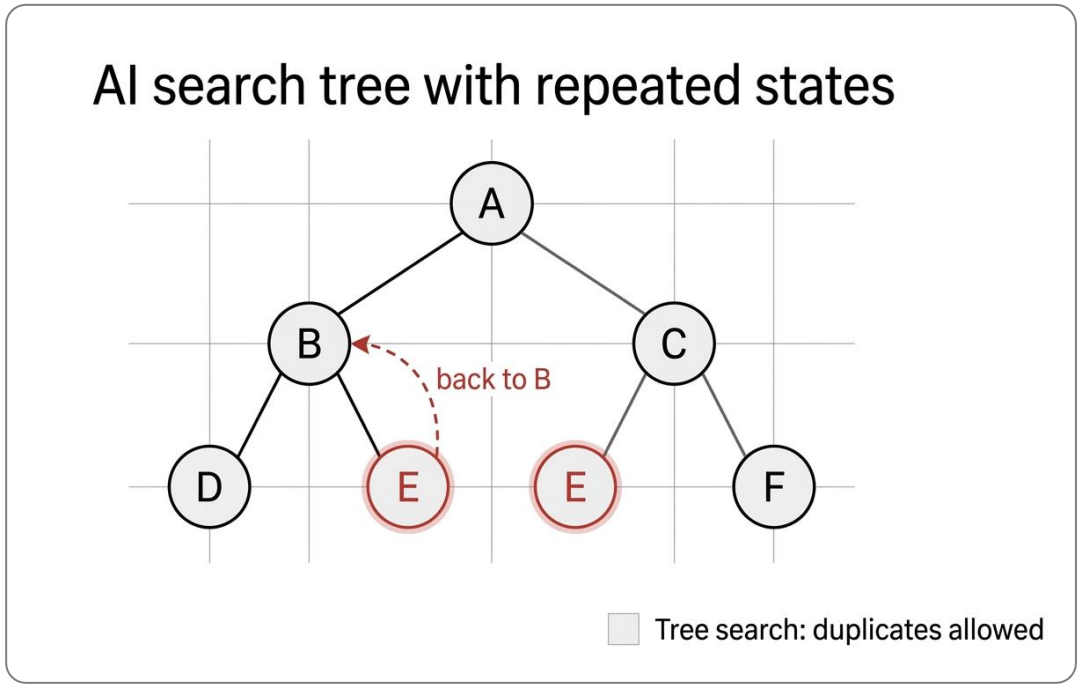
- **Tree:** Can have redundant states (loops/re-exploration)
- **Graph:** Keeps track of visited states (no redundancy)



Tree Search: Redudant States

- **Tree search** expands nodes without remembering visited states
- The **same state** may appear many times
- Can fall into **loops** (A→…→B→…→B→…)
- Wastes time + memory exploring duplicates

Key idea: The tree is about **paths**, not unique states.

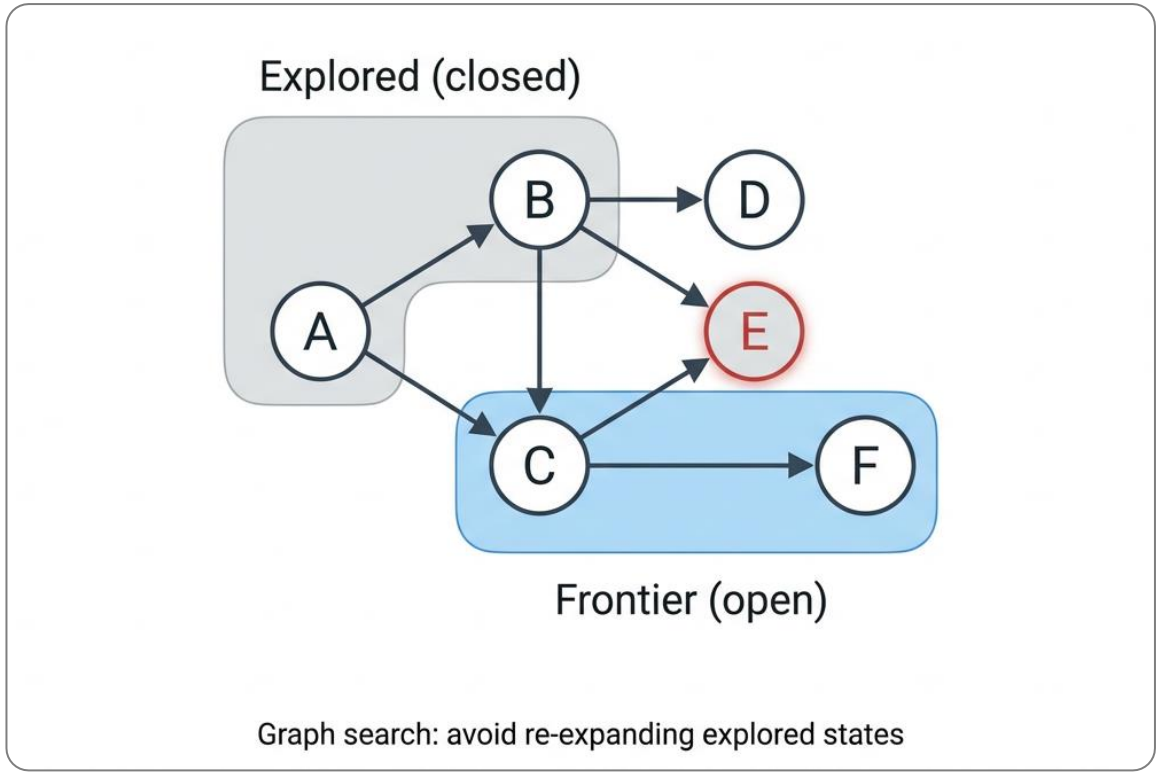


Graph Search: Frontier & Explored

Frontier (OPEN): generated but not expanded yet

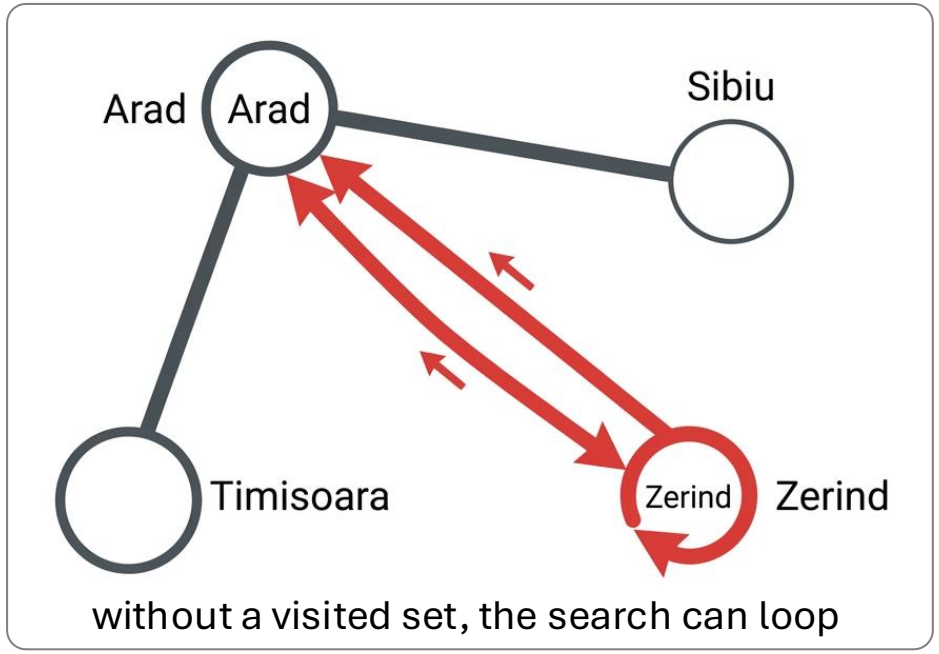
Explored (CLOSED): already expanded (don't expand again)

- If successor is in **closed** → ignore
- If successor is new → push to **frontier**
- Frontier ordering defines BFS / UCS / A*

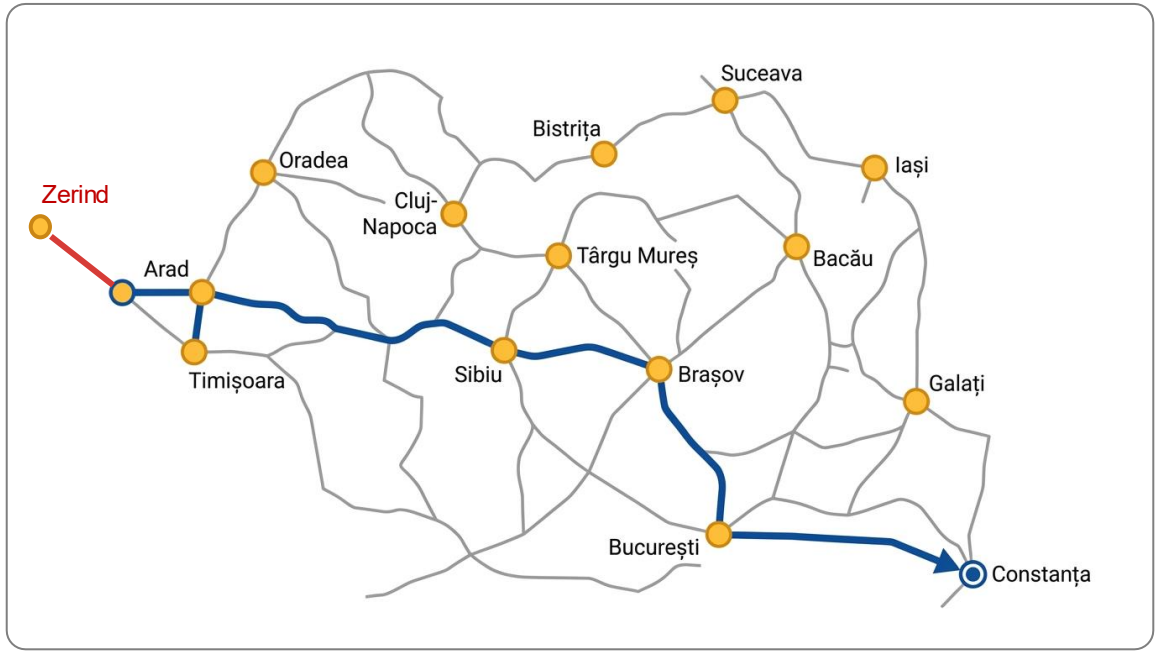


Graph search: avoid re-expanding explored states

Example: Loop on Romania Map



without a visited set, the search can loop

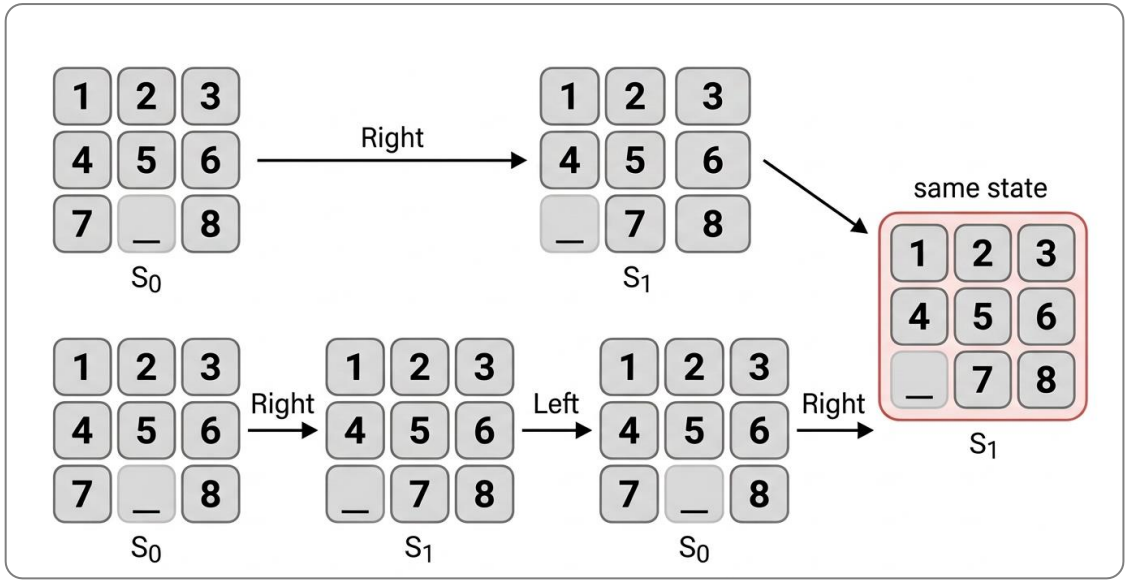


Tree search: may expand Arad repeatedly

Graph search: Arad ∈ explored → stop the loop

Example: Duplicate States in 8-Puzzle

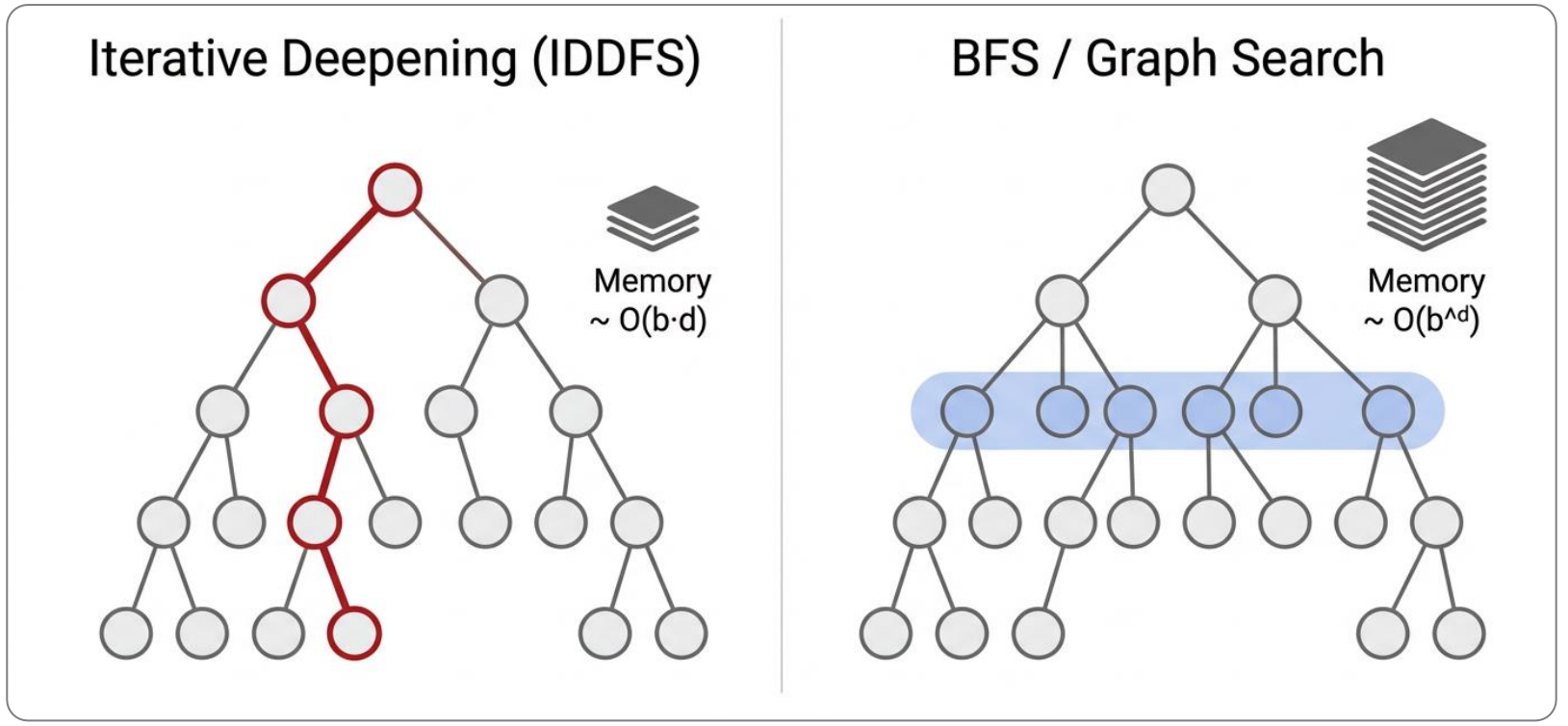
- Two different action sequences can reach the **same board**
- Tree search treats them as different nodes (different paths)
- Graph search stores board configs in a **closed set**
- Result: smaller frontier, fewer expansions



Duplicate states arise when multiple paths reach the same configuration

Heuristic note: duplicate detection helps even before we add heuristics.

Example: When Tree Search Wins



- **Scenario:** very deep search (large d), but RAM is limited
- **Approach:** depth-limited DFS / **iterative deepening (IDDFS)**
- **Benefit:** keeps only a path + small bookkeeping → avoids frontier explosion

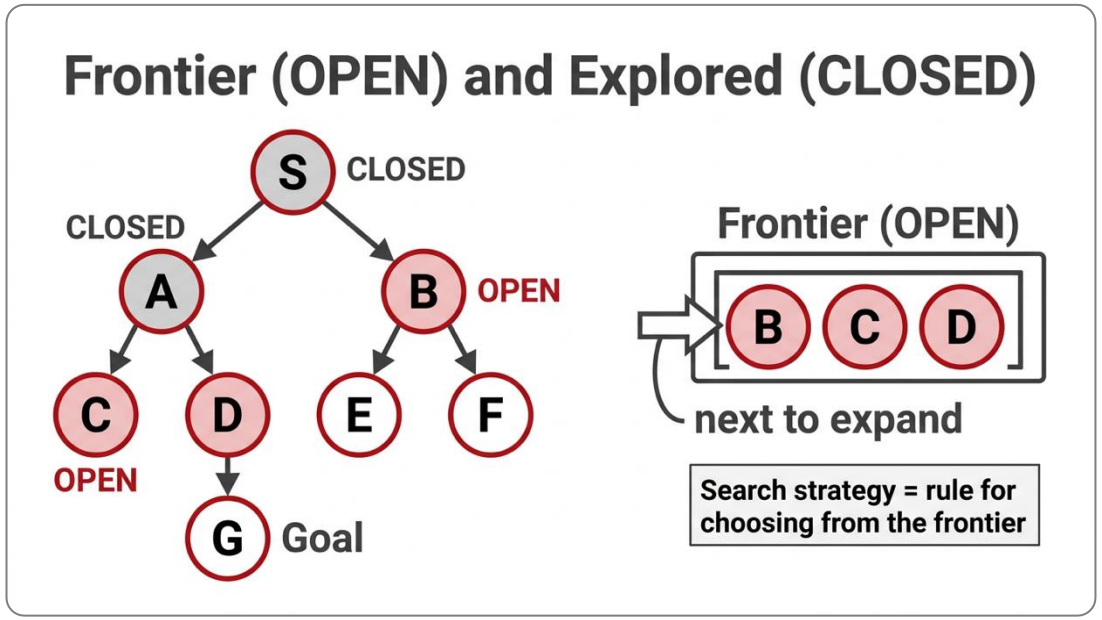
The Frontier

Definition: The set of all leaf nodes available for expansion at any given time.

The General Search Algorithm:

1. Choose node from frontier.
2. Expand the node.
3. Add resulting children to frontier.

Search strategy = how to pick the *next* node from the frontier.



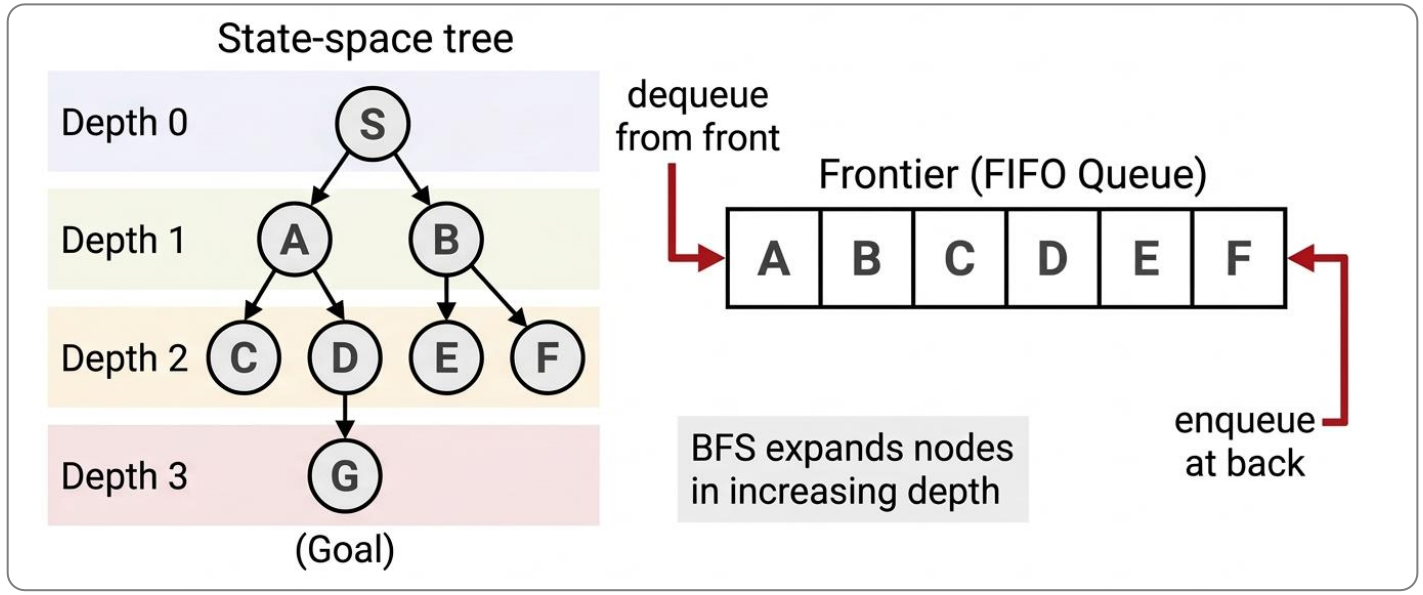
Breadth-First Search (BFS)

Strategy: Explore all nodes at a given depth before moving to the next level.

Frontier Management: FIFO Queue (First-In-First-Out).

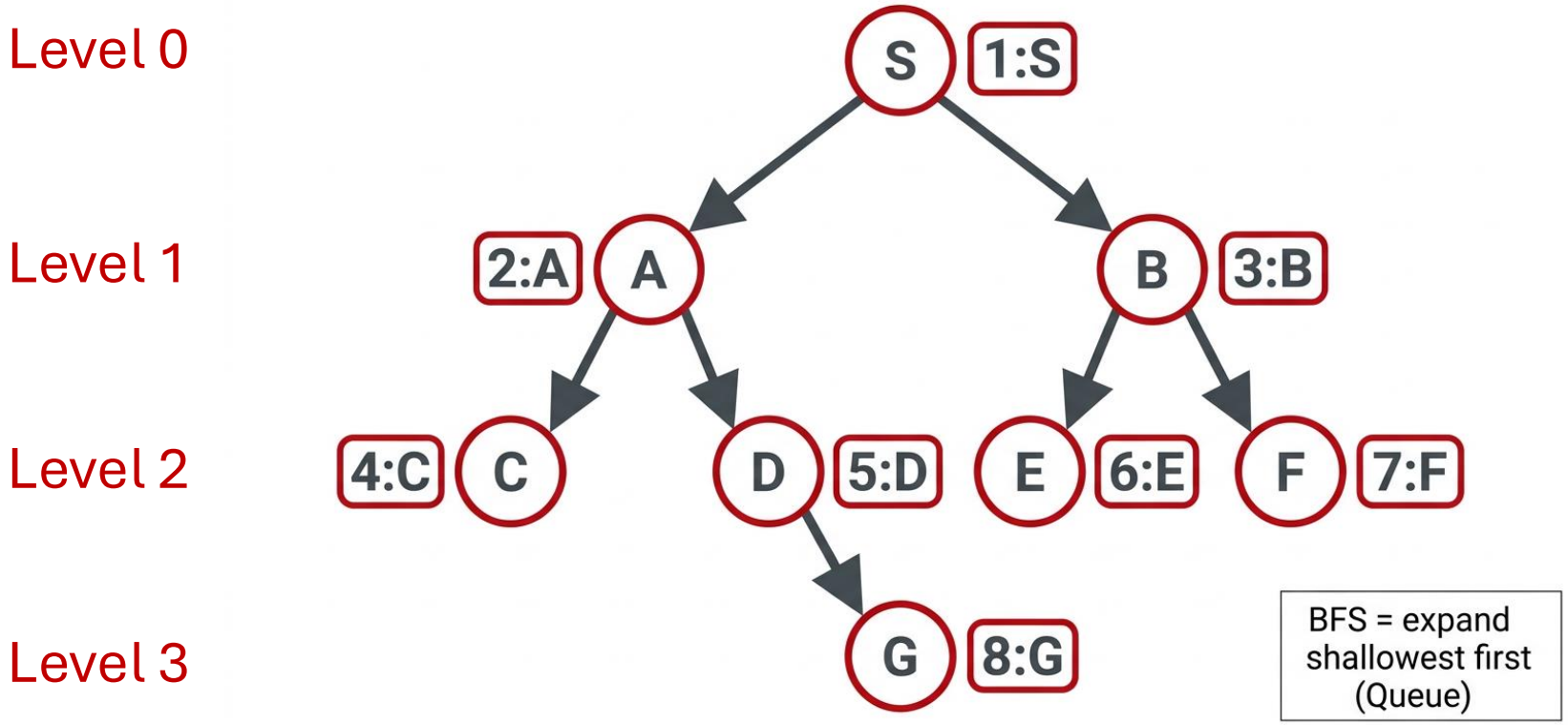
1. Put start node in queue.
2. Dequeue node, expand, and enqueue all children.

BFS always finds the shallowest goal first.



BFS Exploration

Expansion order: Level 0 → Level 1 → Level 2...

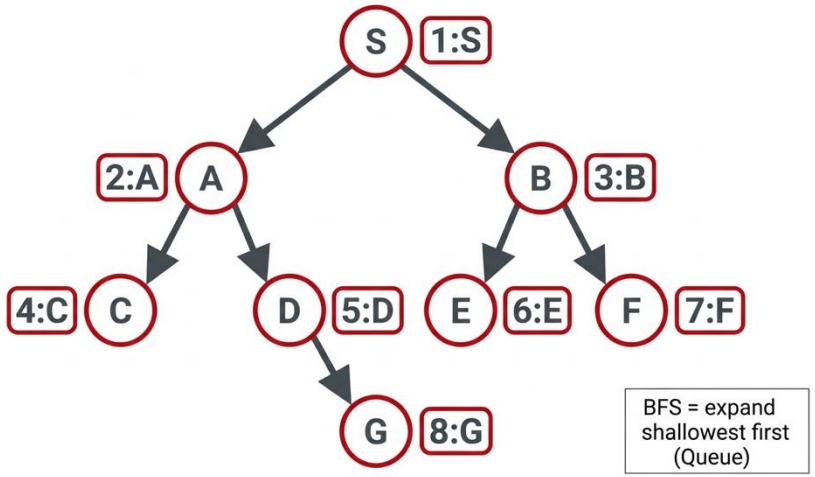


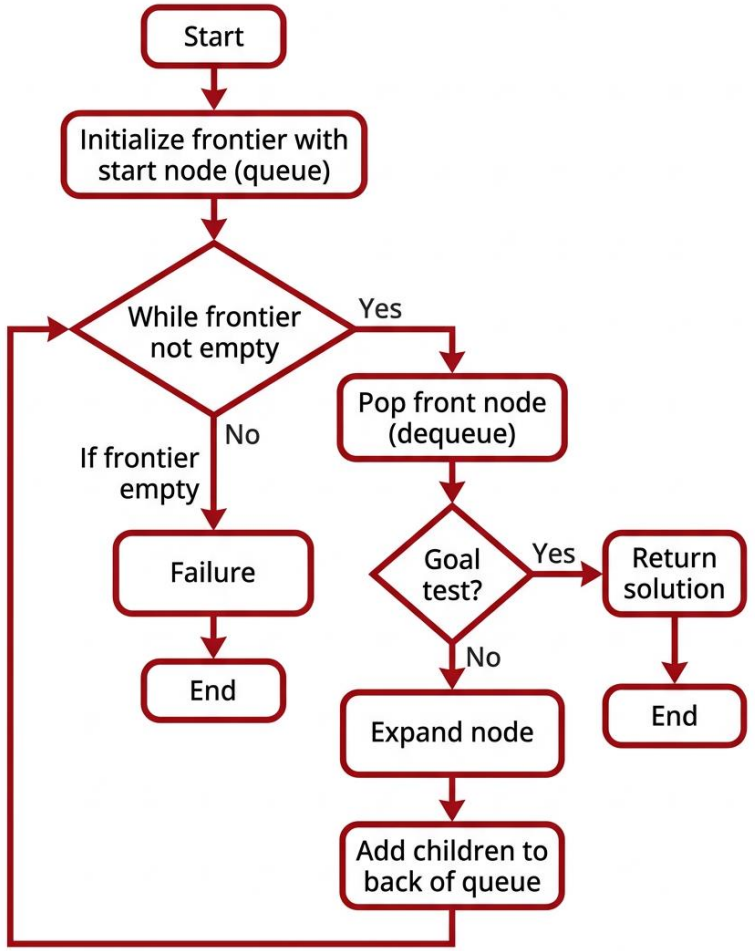
BFS visits nodes layer by layer, finishing one depth before the next.

BFS Algorithm

Standard Loop:

1. Add root to frontier (Queue).
2. Pop node from front.
3. Goal test? Return solution.
4. Expand and add neighbors to back.
5. Repeat until frontier empty.





Properties of BFS

- **Completeness:** Yes (if b is finite).
- **Optimality:** Yes (if costs are identical).
- **Time Complexity:** $O(b^d)$
- **Space Complexity:** $O(b^d)$

Critical Issue: Space is the bigger problem than time for BFS!

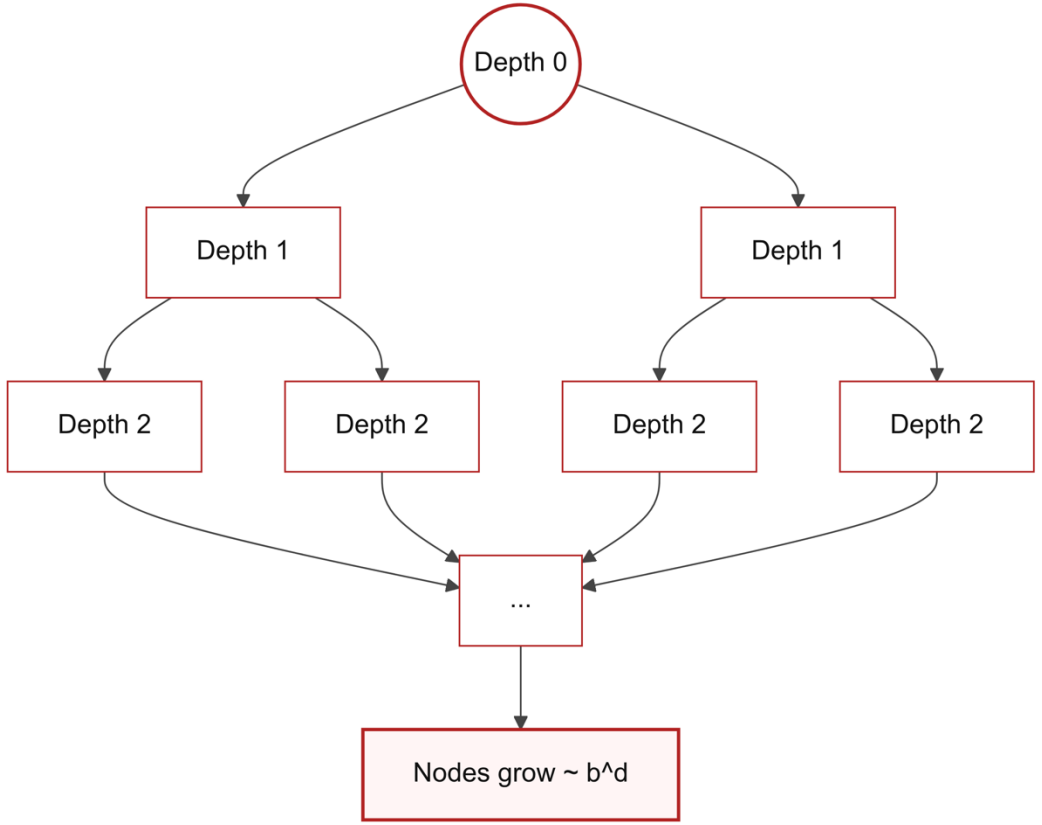
Memory Explosion

BFS stores all nodes in frontier.

Example:

- branching factor b = 10
- depth d = 6
- Nodes ≈ 1,000,000

Memory required increases exponentially with depth!

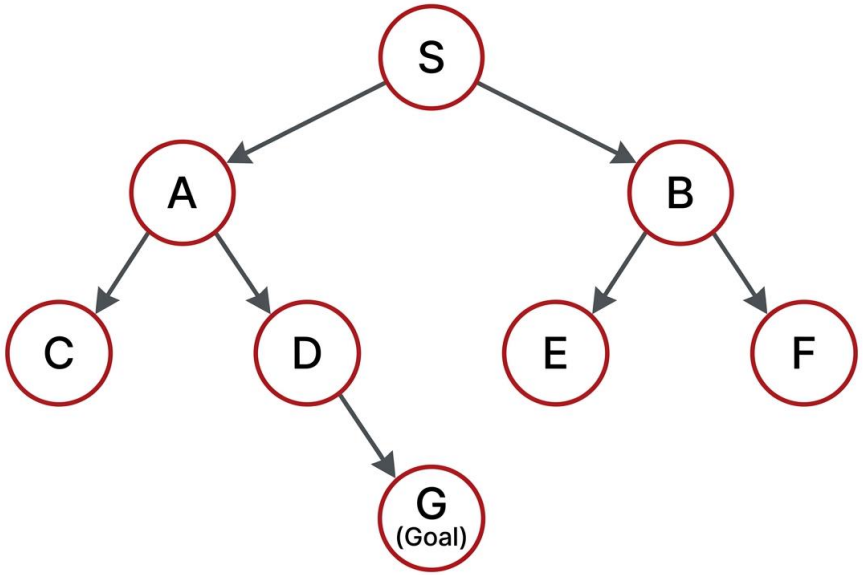


Depth-First Search (DFS)

Strategy: Explore a single path as deeply as possible before backtracking.

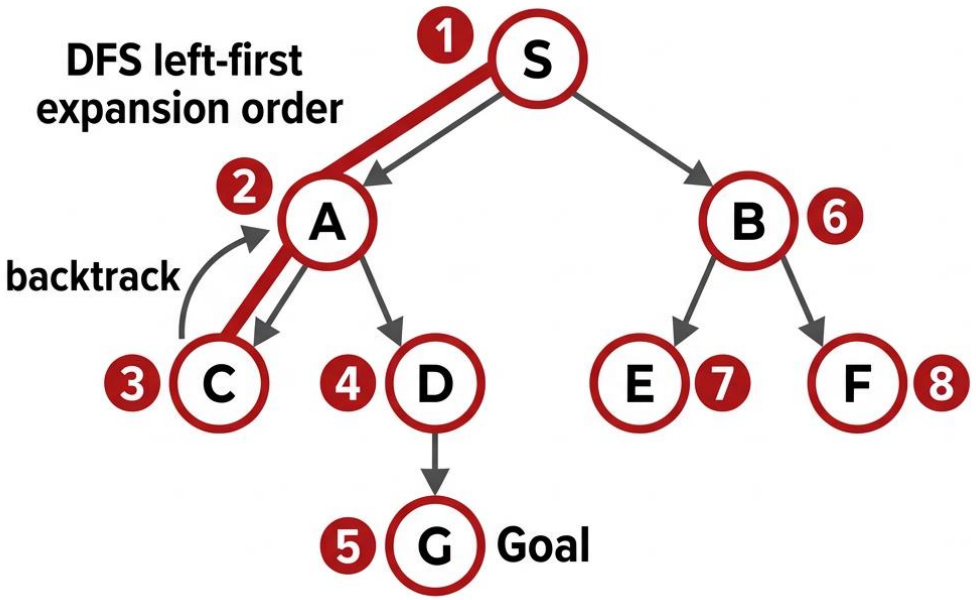
Frontier Management: LIFO Stack (Last-In-First-Out).

1. Put start node in stack.
2. Pop node, expand, and push children onto stack.



DFS Exploration

- Search follow one branch to the bottom.
- If no goal found, backtrack to the nearest unexpanded sibling.
- Efficient memory use but risky for infinite paths.



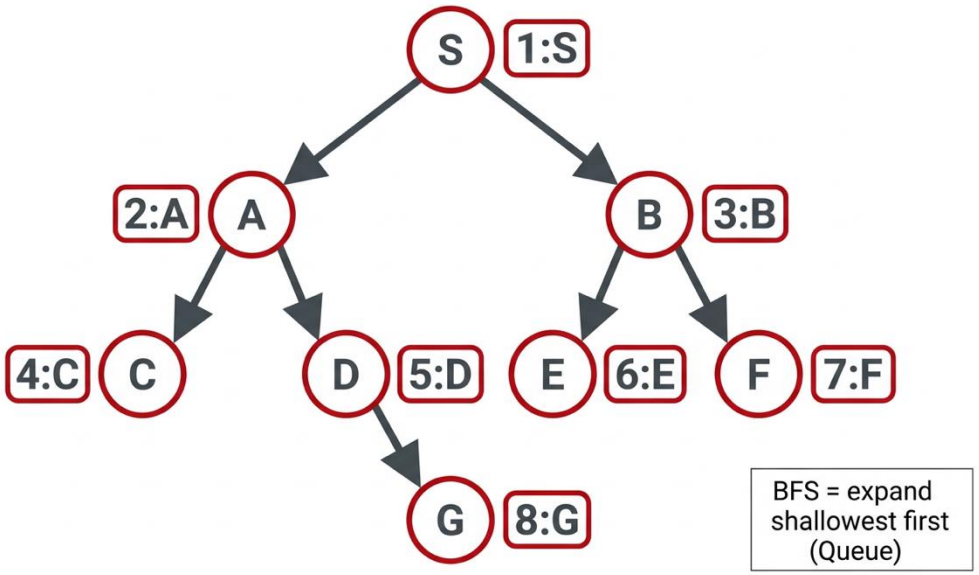
Properties of DFS

- **Space Complexity:** O(bm) — very memory efficient!
- **Completeness:** No (may get stuck in infinite paths).
- **Optimality:** No (finds "some" solution, not necessarily the best).

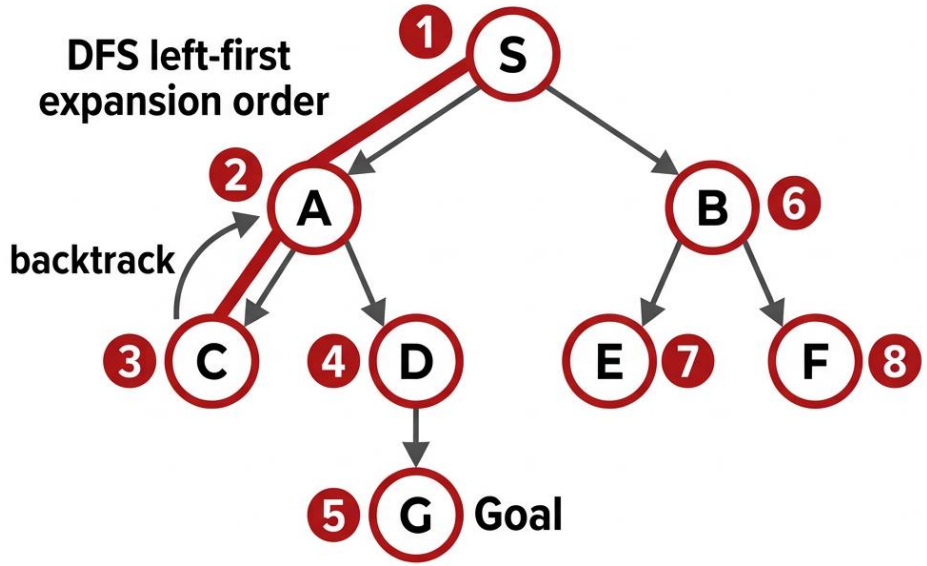
m is the maximum depth of the state space.

BFS vs. DFS

Algorithm	Strategy	Memory	Optimal
BFS	FIFO	O(b^d)	Yes
DFS	LIFO	O(bm)	No

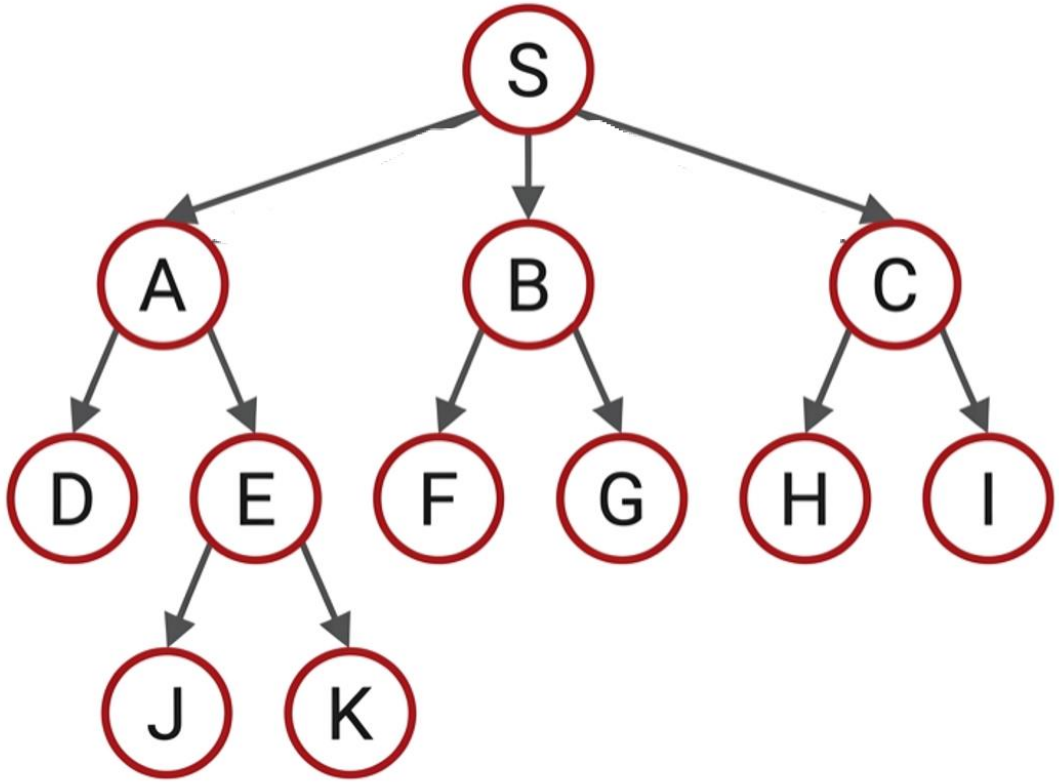


BFS: Layer-by-layer



DFS: Deepest-first

Exercise: Trace BFS & DFS



BFS expansion order:

___ → ___ → ___ …

DFS (left-to-right) expansion order:

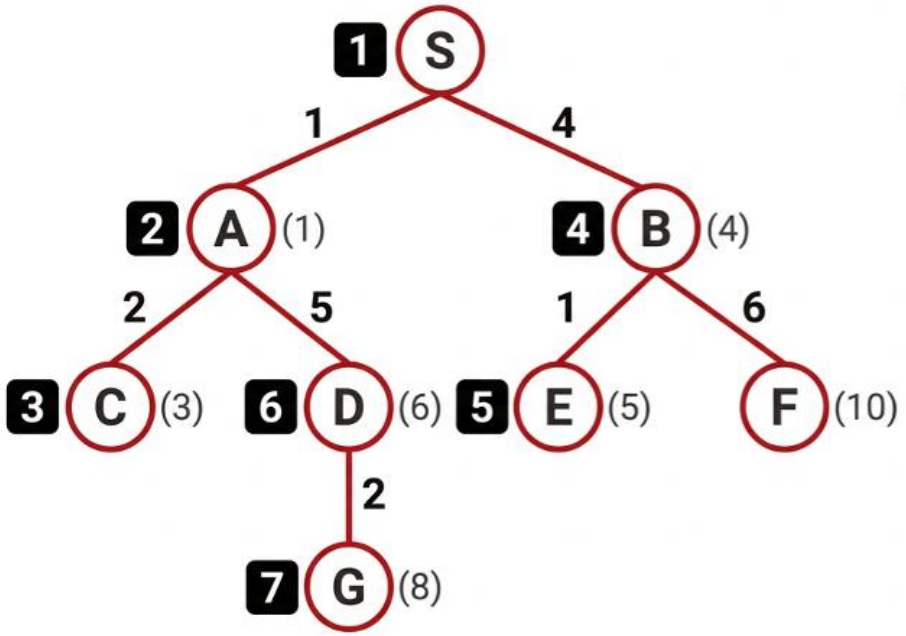
___ → ___ → ___ → …

Uniform Cost Search (UCS)

Challenge: Action costs are not equal (e.g., varying distances).

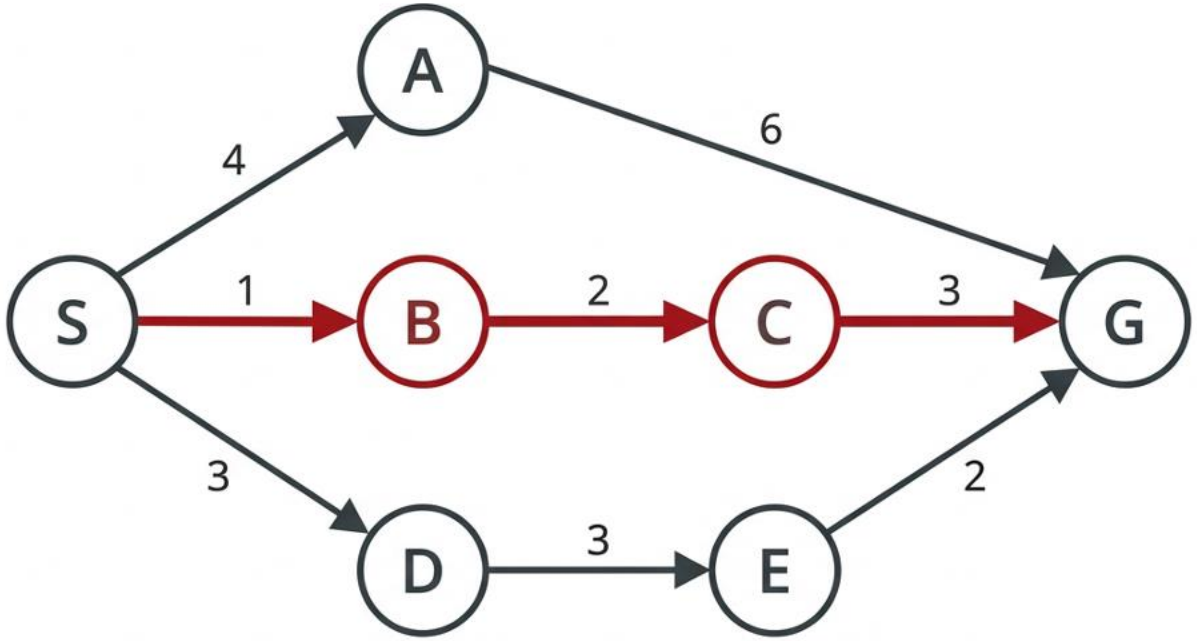
Strategy: Expand the node *n* with the lowest path cost **g(n)**.

Frontier Management: Priority Queue.



UCF Example: Optimal Path

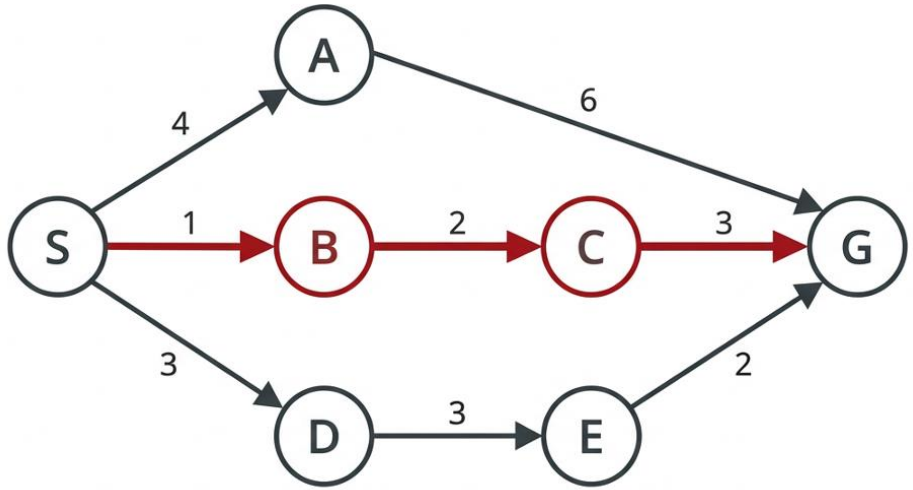
- UCS expands the cheapest partial path available, but still guarantees the optimal solution (lowest cost). **WHY?**



UCF Example: Optimal Path

Priority queue (frontier) states — ordered by cumulative cost g(n)

Expand	Frontier after expansion
S	B:1, D:3, A:4
B	D:3, C:3, A:4
D	C:3, A:4, E:6
C	A:4, E:6, G:6
A	E:6, G:6 (ignore new G:10)
E	G:6 (ignore new G:8)
G	**POP goal → stop. Optimal cost = 6**



UCS guarantees optimality because the first time G is popped, it has the smallest possible g(G).

Properties of UCS

- **Completeness:** Yes (if all step costs ≥ ε > 0).
- **Optimality:** Yes (guaranteed to find the cheapest path).
- **Complexity:** Depends on the cost values, not just depth.

If all costs are equal, UCS becomes BFS.

Comparison of Uninformed Search Algorithms

Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes*	Yes**	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	$O(bm)$
UCS	Yes***	Yes	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$

* if b is finite. ** if step costs are equal. *** if step costs ≥ ε > 0.

b: branching factor, d: depth of shallowest goal, m: max depth, C*: cost of optimal solution.

Summary

Algorithms covered:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Uniform Cost Search (UCS)

Uninformed algorithms explore "blindly" based only on the problem definition.

Next Lecture

Informed Search Algorithms

Greedy Search · A* Search