ARTIFICIAL INTELLIGENCE

Uninformed Search Strategies

- Uninformed/blind search strategies use only the information available in the problem definition

- Generate successors and distinguish a goal state from a non-goal state

Uninformed Search Strategies

Uninformed (blind) strategies use only the information available in the problem definition.

These strategies order nodes without using any domain specific information

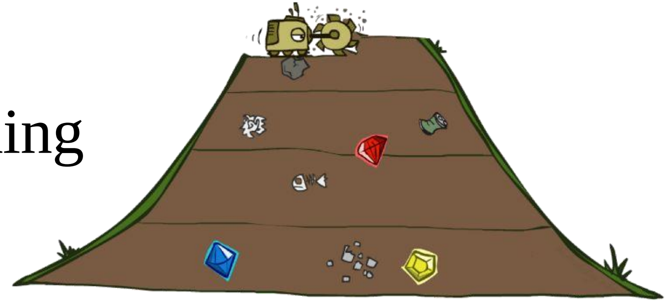
- Breadth-first search
- Depth-first search
- Uniform Cost Search
- Depth-limited search
- Iterative deepening search

Performance Evaluation

- A search strategy is defined by picking the order of node expansion
- Strategies are **evaluated** along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - Time complexity: How long does it take to find a solution
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost/optimal solution?

Breadth First Search (BFS)

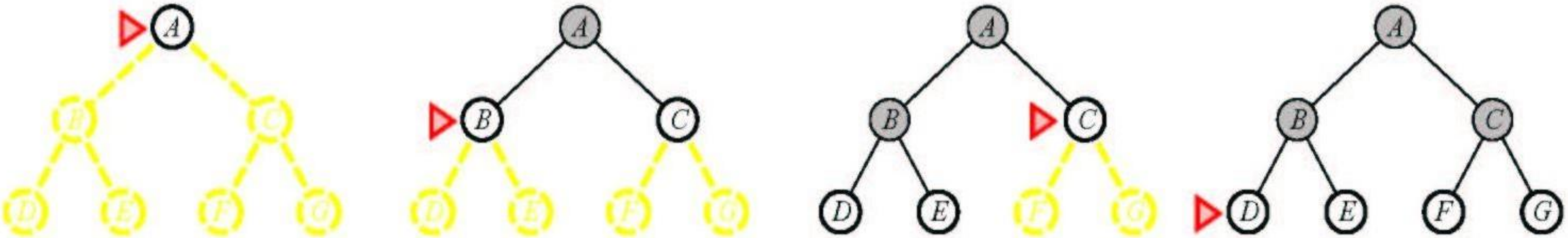
Expand all nodes at depth (i) before expanding nodes at depth (i + 1)
Level-order Traversal.



Implementation:

- Use of a First-In-First-Out queue (FIFO).
- Nodes visited first are expanded first.
- Enqueue nodes in FIFO (first-in, first-out) order.

Breadth First Search (BFS)

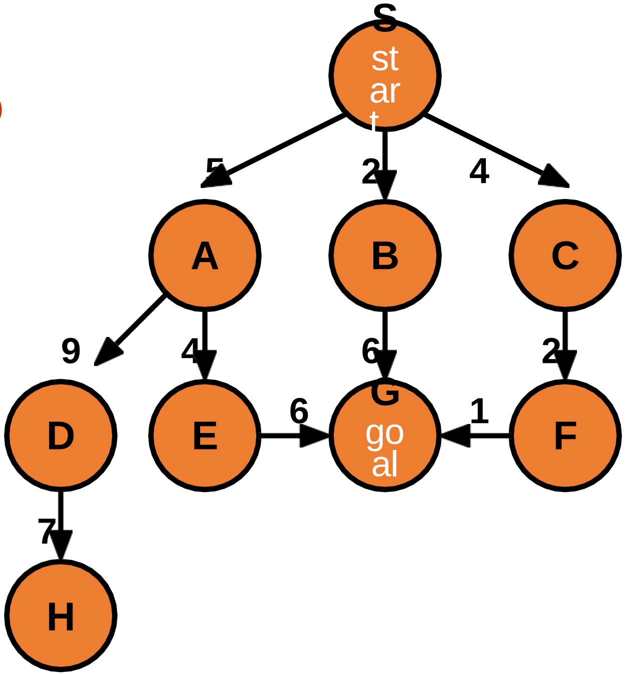


Breadth First Search (BFS)

`generalSearch(problem, queue)`

\# of nodes tested: 0, expanded: 0

expanded node	Frontier list
	{S}

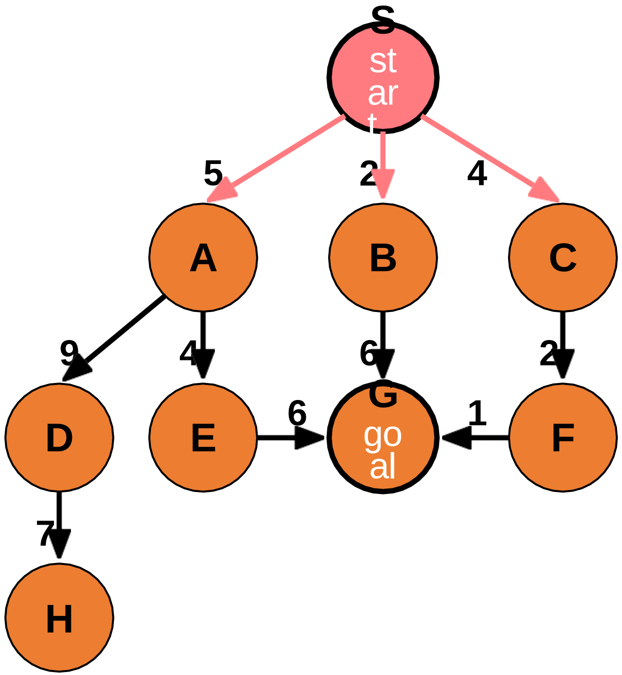


Breadth First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 1, expanded: 1

expanded node	Frontier list
	{S}
S not goal	{A,B,C}

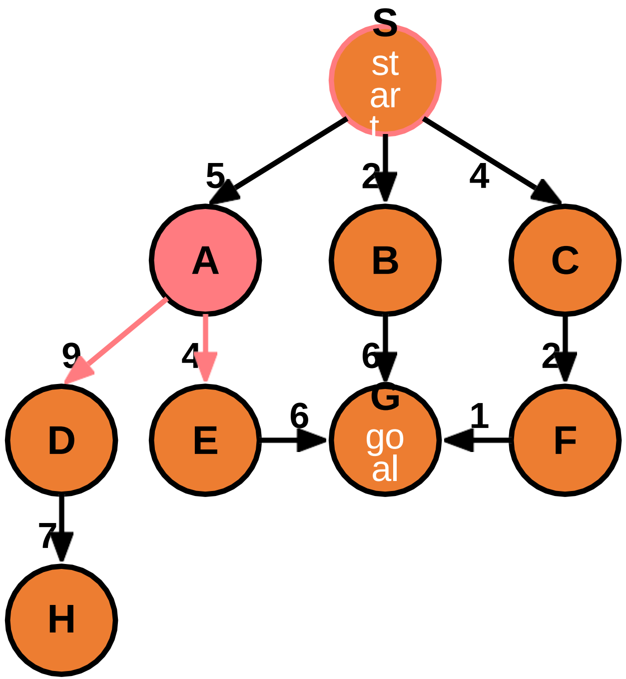


Breadth First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 2, expanded: 2

Expanded node	**Frontier list**
	{S}
S	{A,B,C}
A not goal	{B,C,D,E}

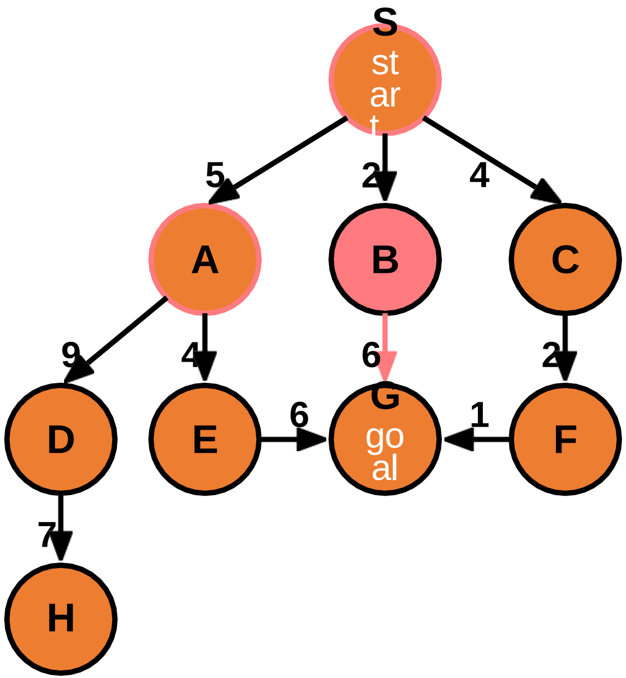


Breadth First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 3, expanded: 3

Expanded node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B not goal	{C,D,E,G}

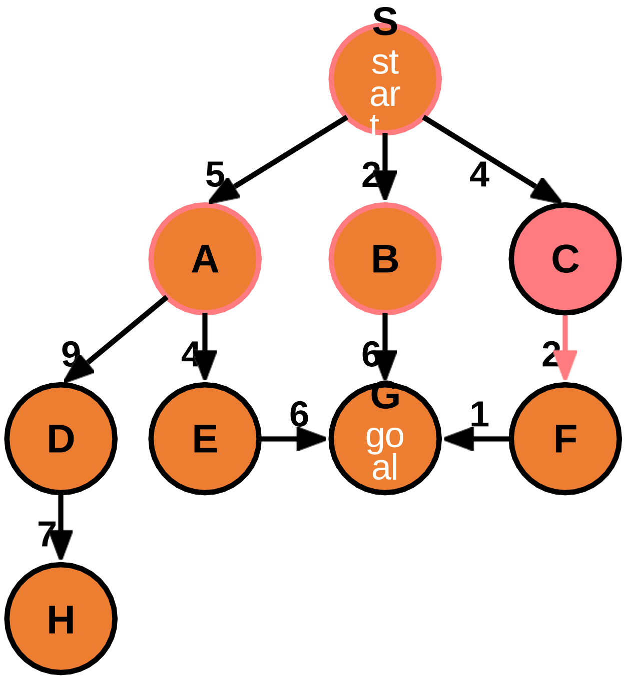


Breadth First Search (BFS)

generalSearch(problem, queue)

# of nodes tested: 4, expanded: 4

Expanded node	**Frontier list**
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C not goal	{D,E,G,F}

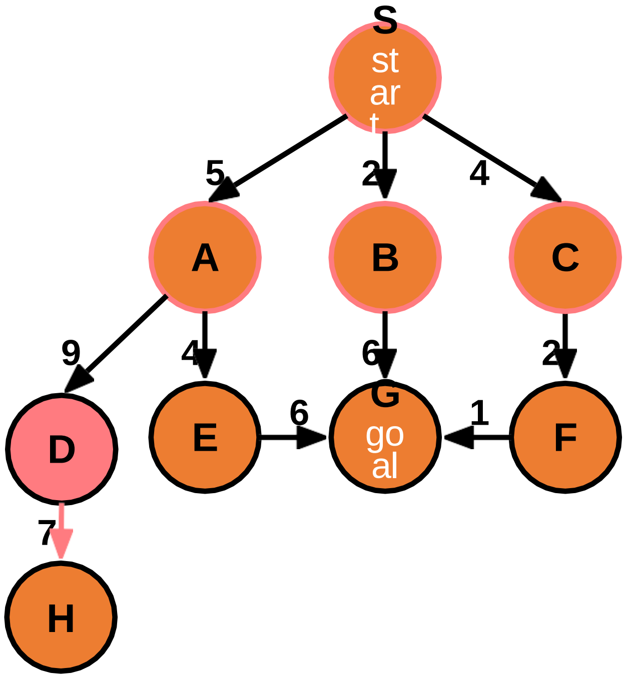


Breadth First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 5, expanded: 5

Expanded node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D not goal	{E,G,F,H}

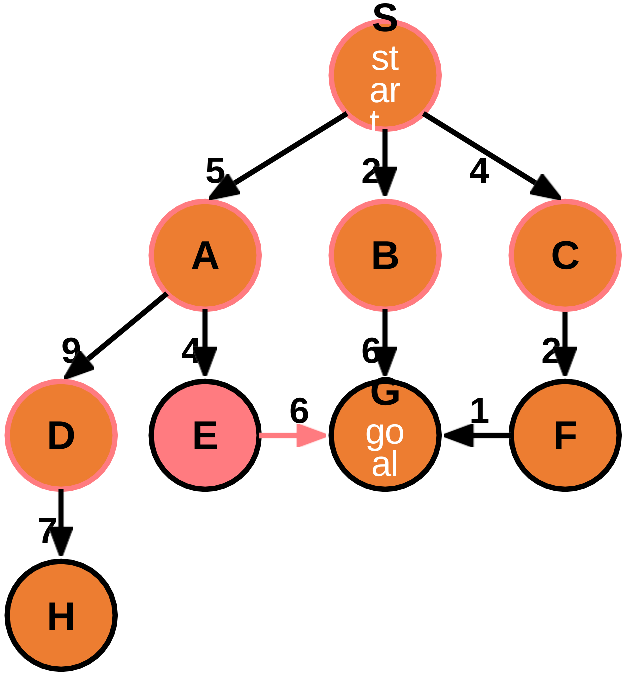


Breadth First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 6 , expanded: 6

Expanded node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E not goal	{G,F,H,G}

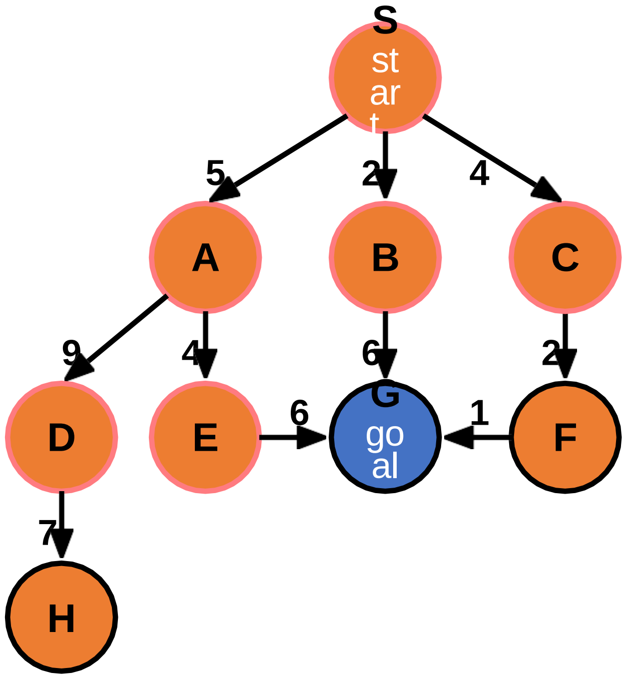


Breadth First Search (BFS)

`generalSearch(problem, queue)`

\# of nodes tested: 7 , expanded: 6

Expanded node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E	{G,F,H,G}
G goal	{F,H,G} no expand

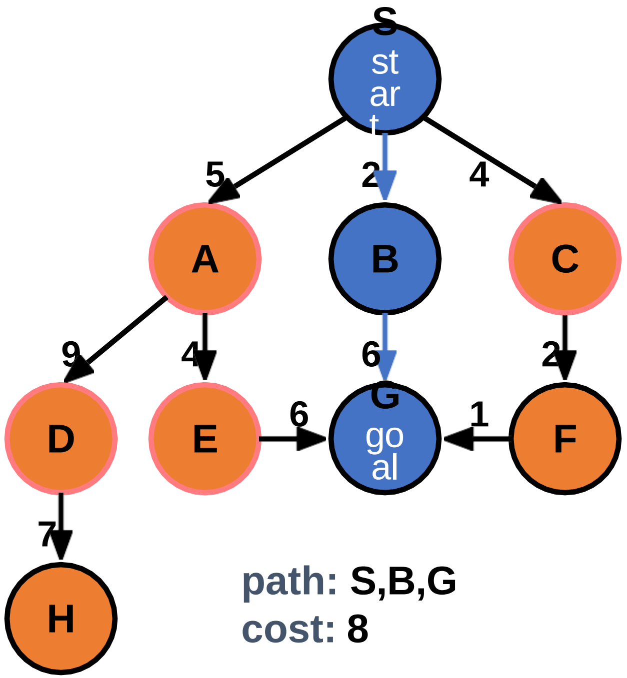


Breadth First Search (BFS)

generalSearch(problem, queue)

\# of nodes tested: 7 , expanded: 6

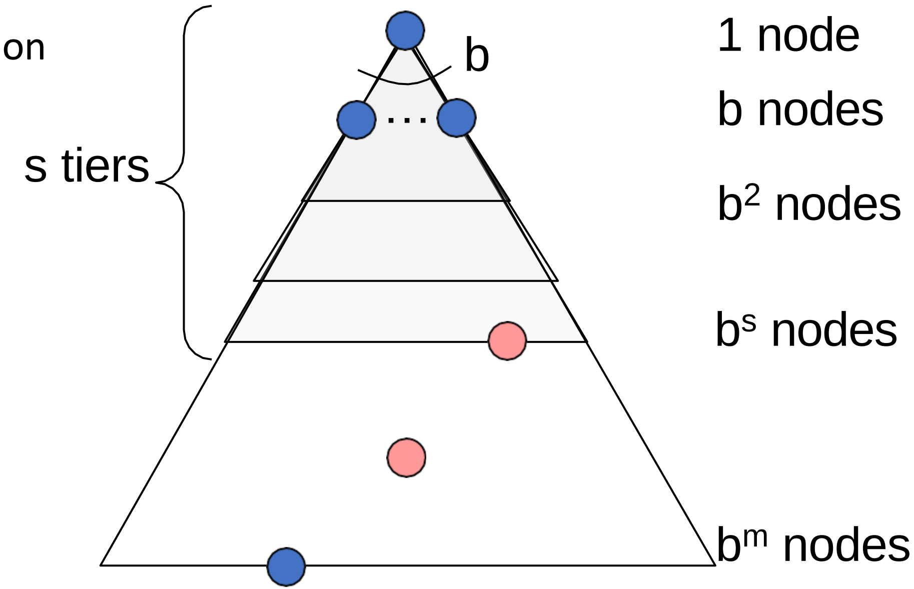
Expanded node	Frontier list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E	{G,F,H,G}
G goal	{F,H,G} no expand



path: S,B,G
cost: 8

BFS Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



BFS Evaluation

- Two lessons:
  - Memory requirements are a bigger problem
  - Time requirements are major factor. Takes too much time to go at depth

Uniform Cost Search (UCS)

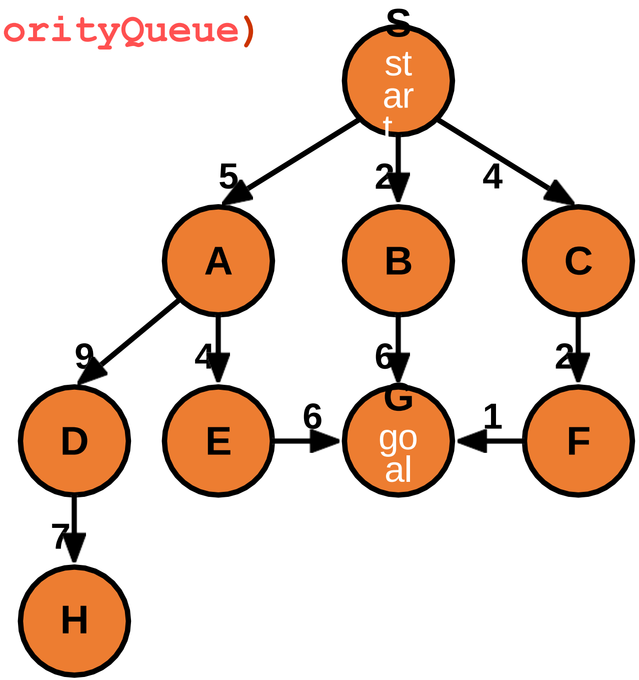
- Extension of BFS:
  - Expand node with *lowest path cost*
- Implementation: *fringe* = queue ordered by path cost.
- UCS is the same as BFS when all step-costs are equal.
- UCS does not care about the *number* of steps a path has, but only about their total cost

Uniform Cost Search (UCS)

generalSearch(problem, priorityQueue)

# of nodes tested: 0, expanded: 0

expnd. node	Frontier list
	{S}

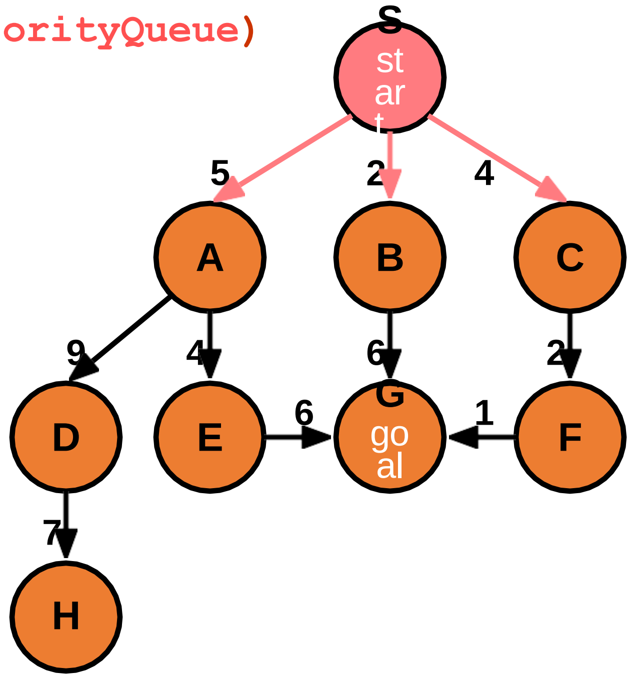


Uniform Cost Search (UCS)

```
generalSearch(problem, priorityQueue)
```

# of nodes tested: 1, expanded: 1

expnd. node	Frontier list
	{S:0}
S not goal	{B:2,C:4,A:5}

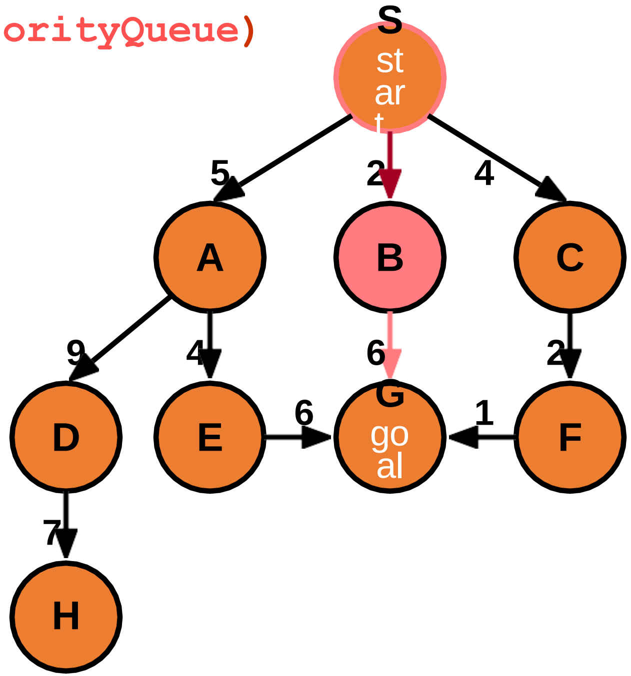


Uniform Cost Search (UCS)

```
generalSearch(problem, priorityQueue)
```

# of nodes tested: 2, expanded: 2

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B not goal	{C:4,A:5,G:2+6}

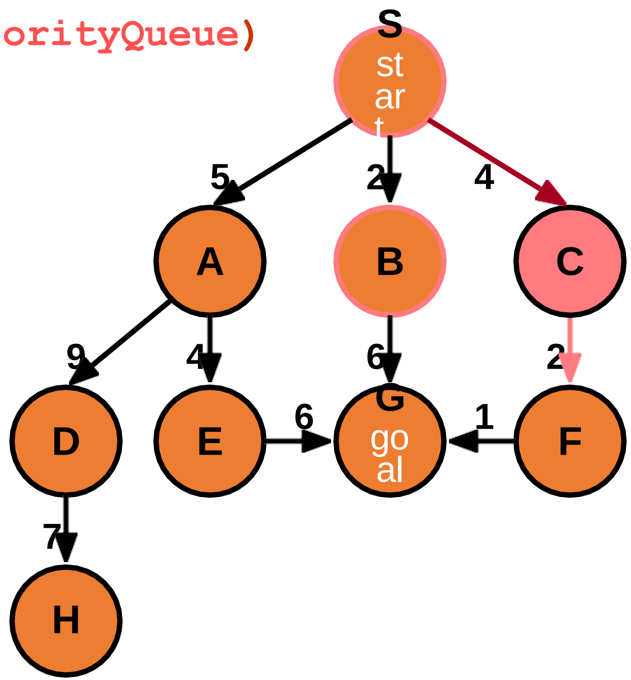


Uniform Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

# of nodes tested: 3, expanded: 3

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C not goal	{A:5,F:4+2,G:8}

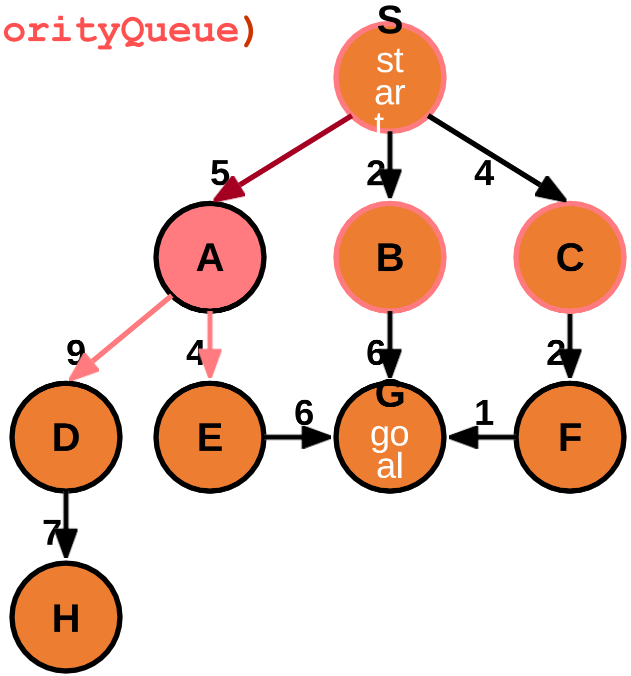


Uniform Cost Search (UCS)

generalSearch(problem, priorityQueue)

# of nodes tested: 4, expanded: 4

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A not goal	{F:6,G:8,E:5+4, D:5+9}

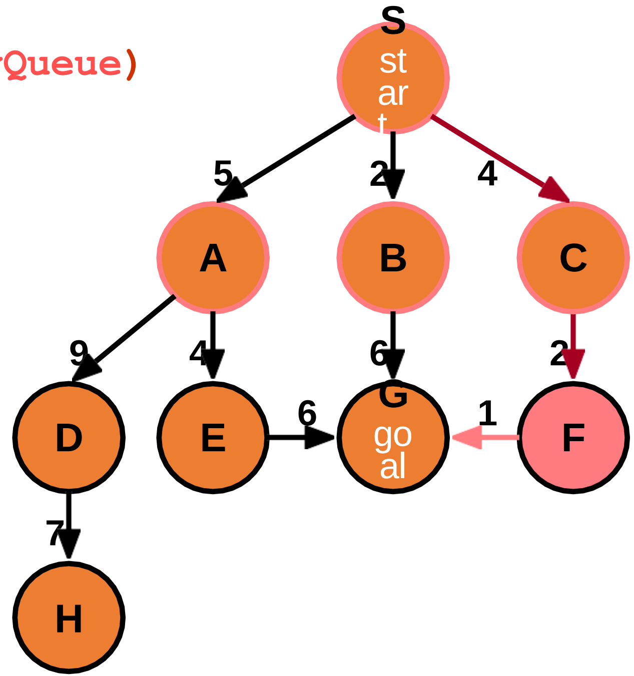


Uniform Cost Search (UCS)

generalSearch(problem, priorityQueue)

\# of nodes tested: 5, expanded: 5

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A	{F:6,G:8,E:9,D:14 }
F not goal	{G:4+2+1,G:8,E:9 ,D:14}

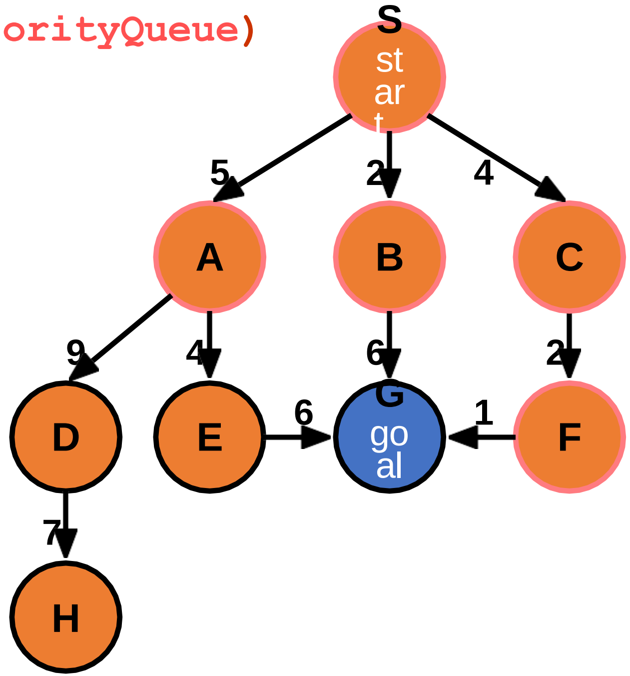


Uniform Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

\# of nodes tested: 6, expanded: 5

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A	{F:6,G:8,E:9,D:14 }
F	{G:7,G:8,E:9,D:14 }
G goal	{G:8,E:9,D:14} no expand

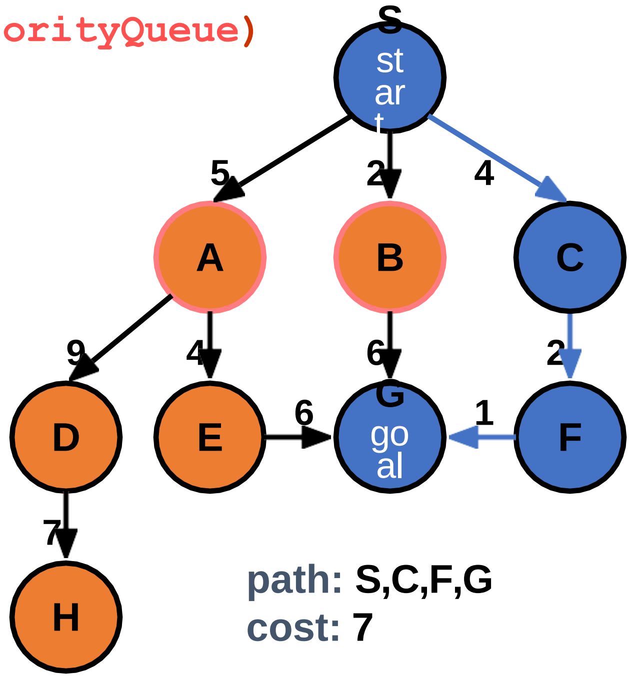


Uniform Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

# of nodes tested: 6, expanded: 5

expnd. node	Frontier list
	{S}
S	{B:2,C:4,A:5}
B	{C:4,A:5,G:8}
C	{A:5,F:6,G:8}
A	{F:6,G:8,E:9,D:14}
F	{G:7,G:8,E:9,D:14}
G	{G:8,E:9,D:14}



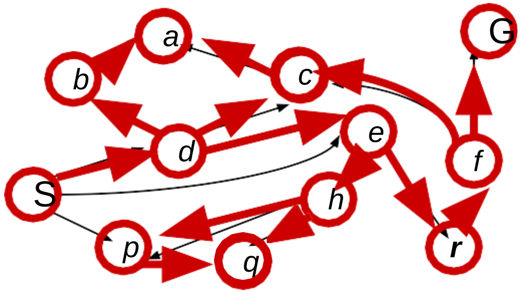
path: S,C,F,G
cost: 7

UCS Evaluation

- **Optimality:** Yes
  - nodes expanded in order of increasing path cost.
  - Therefore, the first goal node selected for expansion is the optimal solution.
- **Completeness:** YES
- **Time/Space complexity:**
  - Assume C* the cost of the optimal solution.
  - Assume that every action costs at least ε
  - Worst-case:

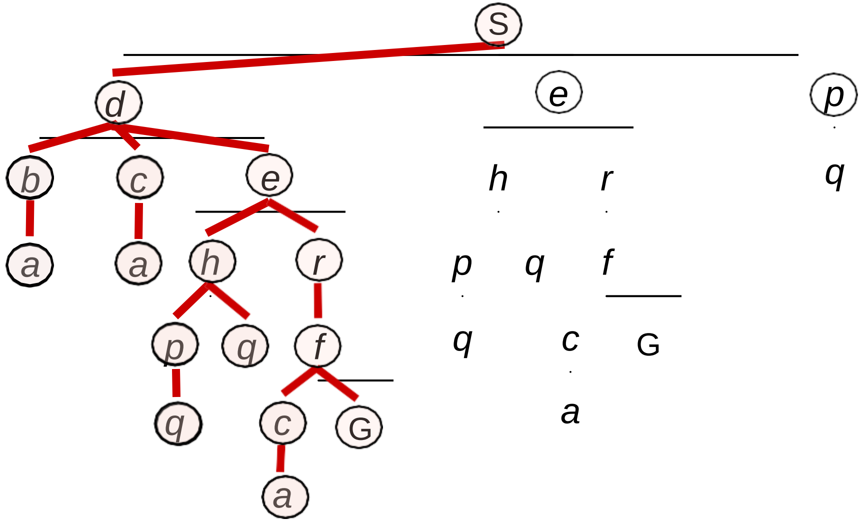
$$O(b^{C^*/\varepsilon})$$

Depth-First Search



Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

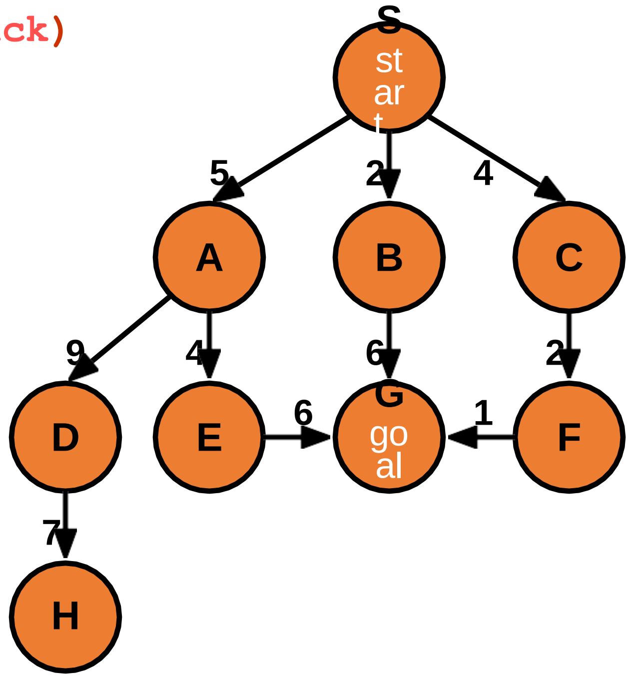


Depth-First Search (DFS)

generalSearch(problem, stack)

# of nodes tested: 0, expanded: 0

expnd. node	Frontier
	{S}

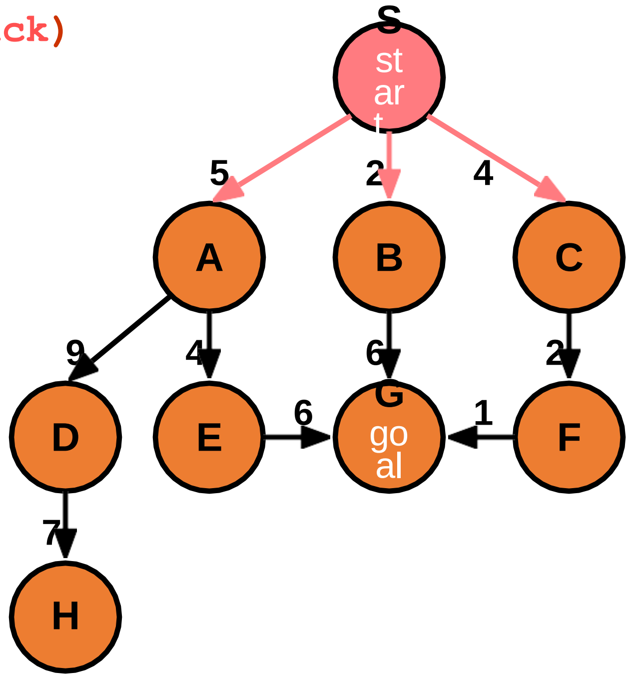


Depth-First Search (DFS)

`generalSearch(problem, stack)`

# of nodes tested: 1, expanded: 1

expnd. node	Frontier
	{S}
S not goal	{A,B,C}

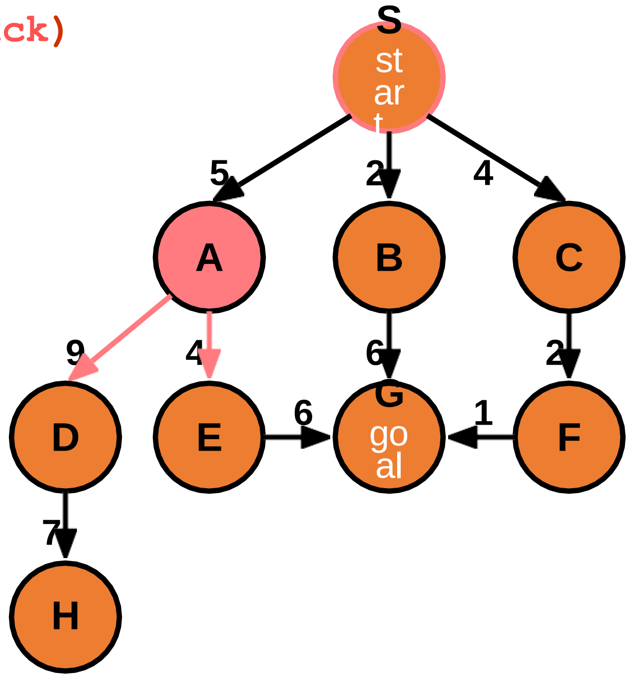


Depth-First Search (DFS)

generalSearch(problem, stack)

# of nodes tested: 2, expanded: 2

expnd. node	Frontier
	{S}
S	{A,B,C}
A not goal	{D,E,B,C}

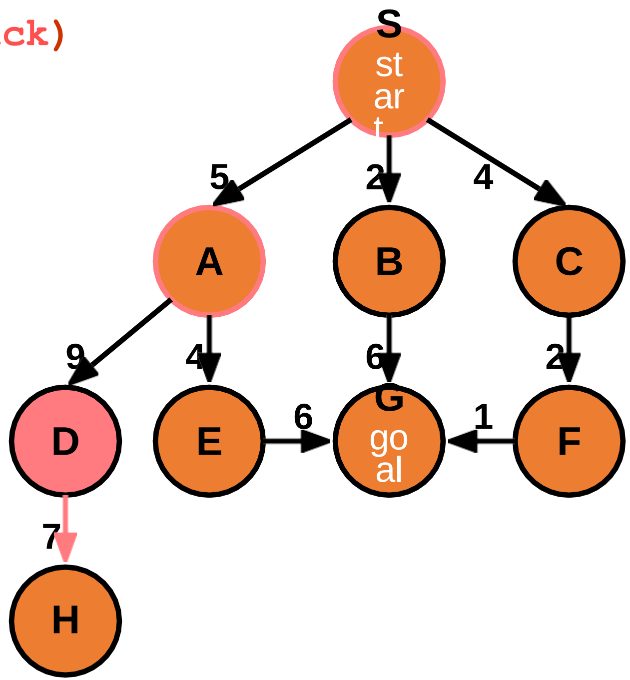


Depth-First Search (DFS)

`generalSearch(problem, stack)`

# of nodes tested: 3, expanded: 3

expnd. node	**Frontier**
	{S}
S	{A,B,C}
A	{D,E,B,C}
D not goal	{H,E,B,C}

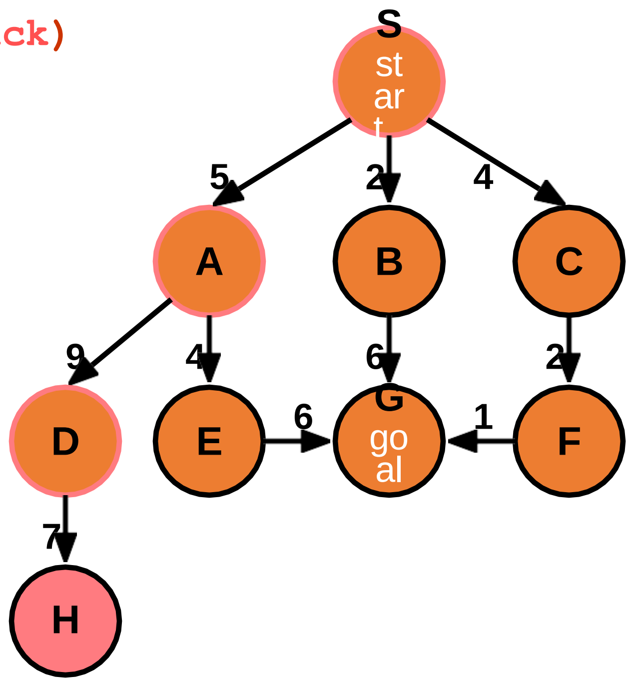


Depth-First Search (DFS)

generalSearch(problem, stack)

# of nodes tested: 4, expanded: 4

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H not goal	{E,B,C}

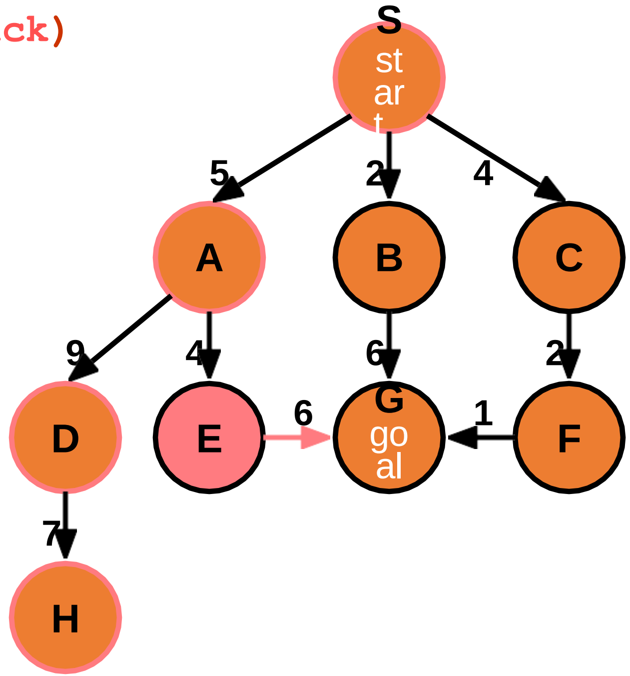


Depth-First Search (DFS)

`generalSearch(problem, stack)`

# of nodes tested: 5, expanded: 5

expnd. node	**Frontier**
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E not goal	{G,B,C}

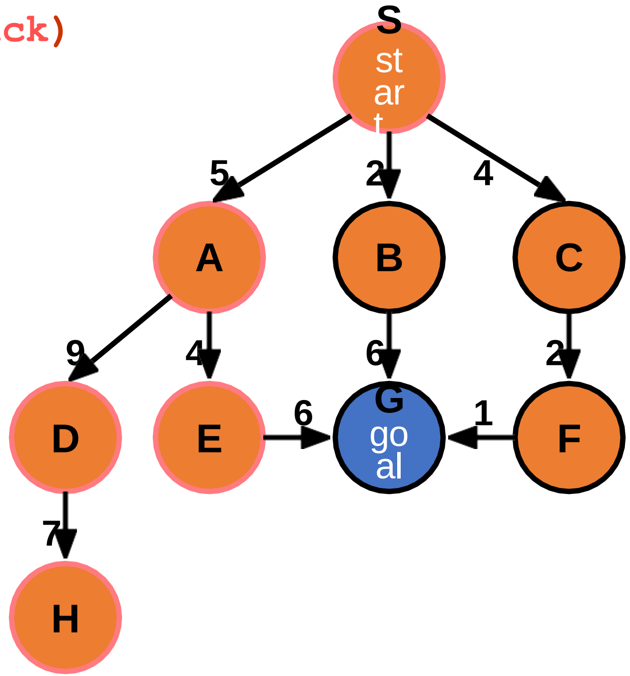


Depth-First Search (DFS)

generalSearch(problem, stack)

# of nodes tested: 6, expanded: 5

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G goal	{B,C} no expand

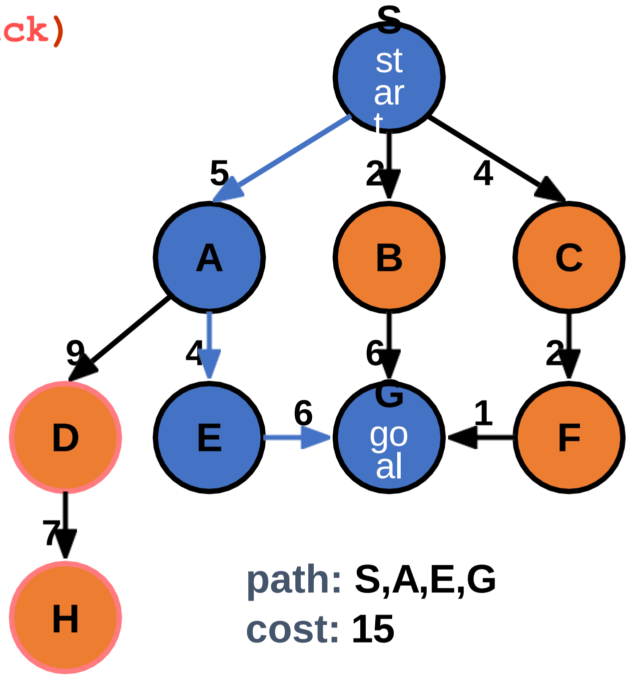


Depth-First Search (DFS)

`generalSearch(problem, stack)`

\# of nodes tested: 6, expanded: 5

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G	{B,C}

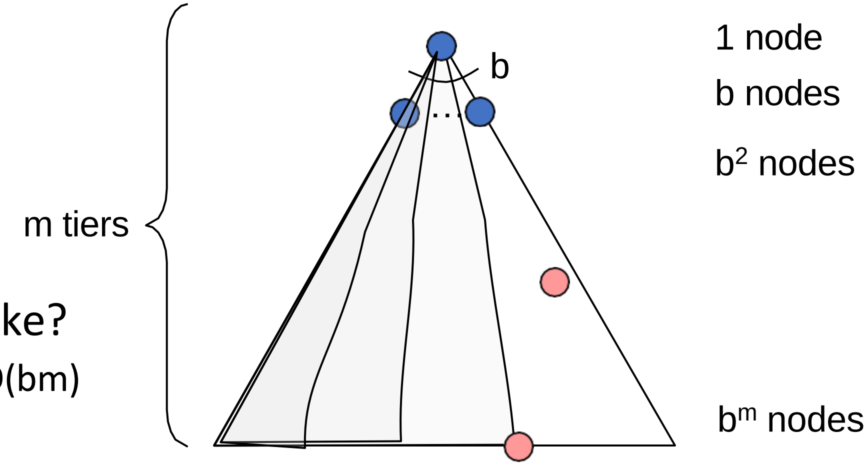


path: **S,A,E,G**

cost: **15**

Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost



Depth First Search (DFS)

Main idea: Expand node at the deepest level (breaking ties left to right).

Implementation: use of a Last-In-First-Out queue or stack(LIFO). Enqueue nodes in LIFO (last-in, first-out) order.

- ❑ Complete? No (Yes on finite trees, with no loops).
- ❑ Optimal? No
- ❑ Time Complexity: $O(b^m)$, where m is the maximum depth.
- ❑ Space Complexity: O(bm), where m is the maximum depth.

DFS Issue

- The drawback of DFS is that It can make a wrong choice and *get stuck* going down a very long path when a different choice would lead to a solution near the root of the search tree.

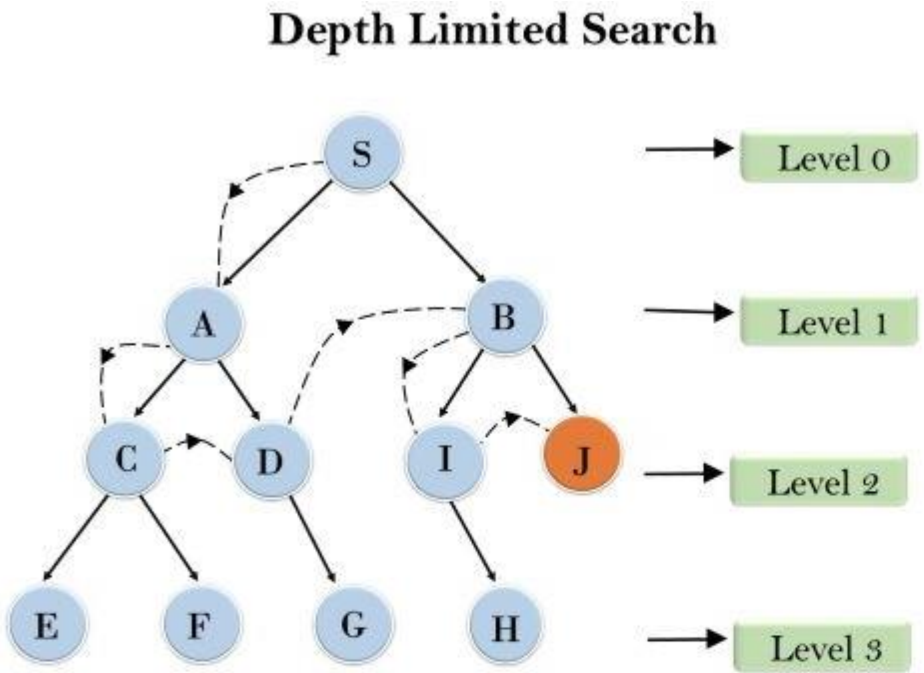
Depth-Limited Search (DLS)

It is simply DFS with a depth bound.

Searching is not permitted beyond the depth bound.

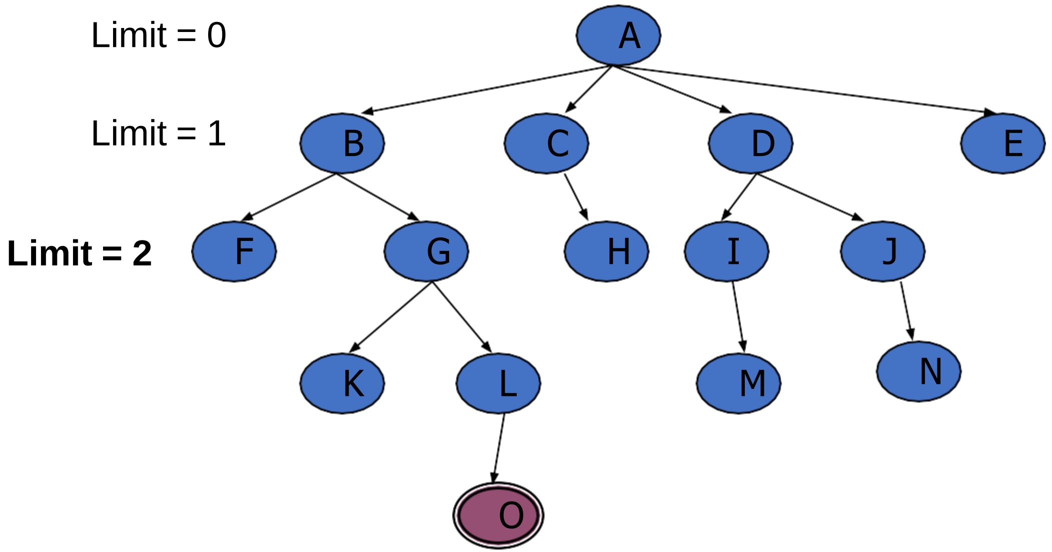
- Works well if we know what the depth of the solution is.
- Termination is guaranteed.
- If the solution is beneath the depth bound, the search cannot find the goal (hence this search algorithm is incomplete).
- Otherwise use Iterative deepening search (IDS).

Depth-Limited Search (DLS)



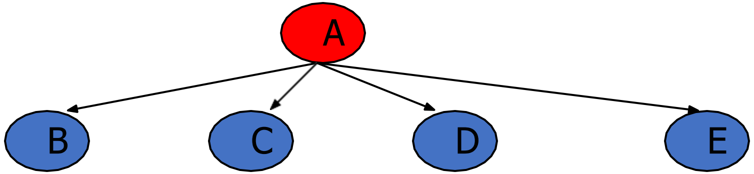
Depth-Limited Search (DLS)

- Given the following state space (tree search), give the sequence of visited nodes when using DLS (Limit = 2):



Depth-Limited Search (DLS)

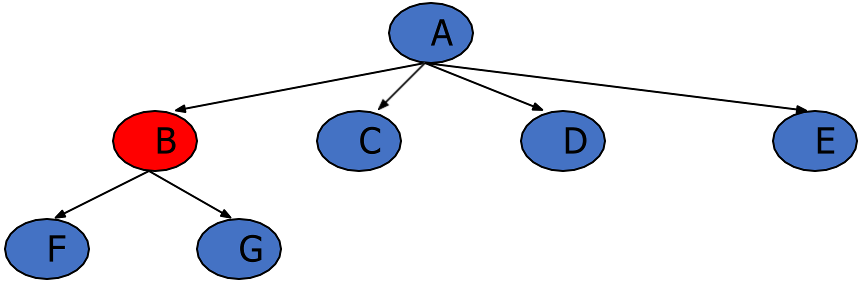
- A,



Limit = 2

Depth-Limited Search (DLS)

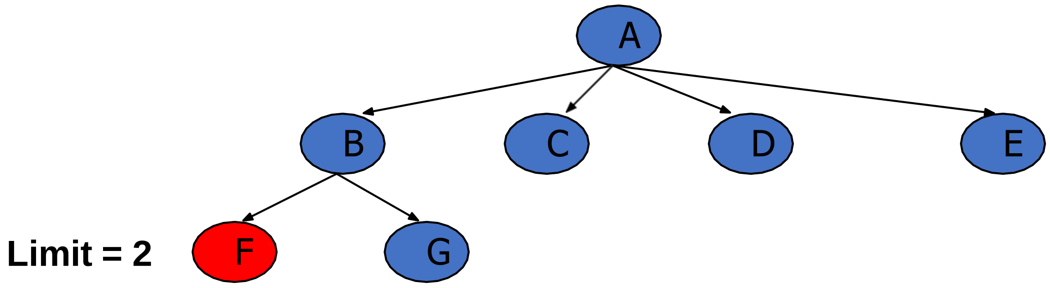
- A,B,



Limit = 2

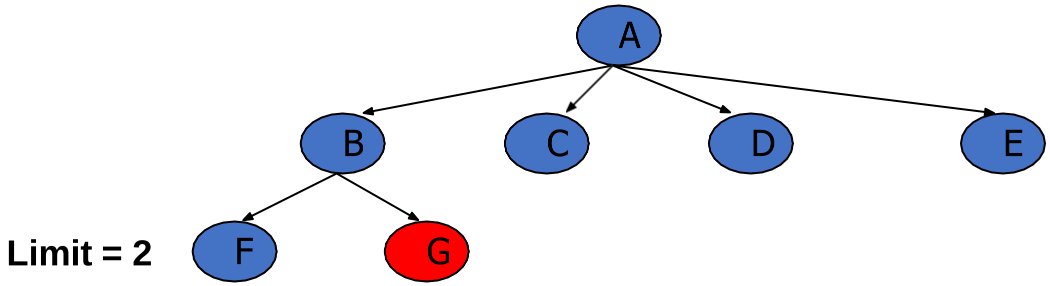
Depth-Limited Search (DLS)

- A,B,F,



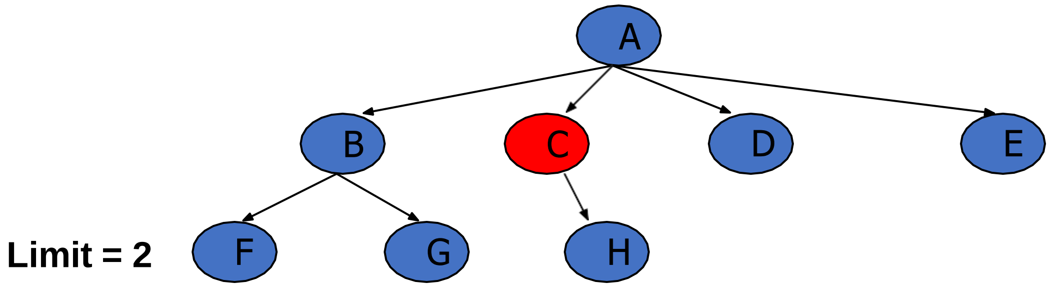
Depth-Limited Search (DLS)

- A,B,F,
- G,



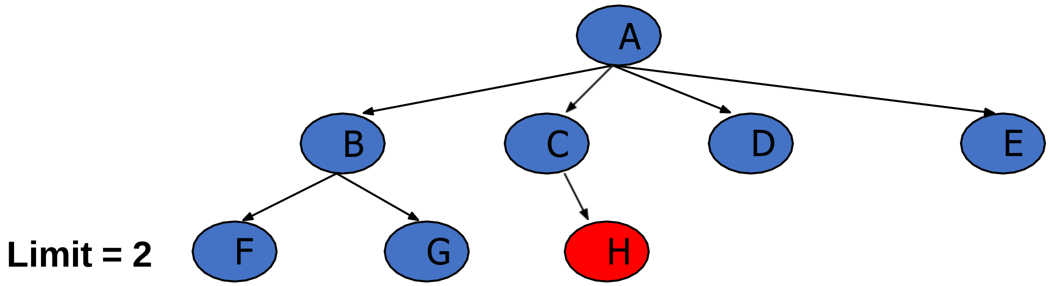
Depth-Limited Search (DLS)

- A,B,F,
- G,
- C,



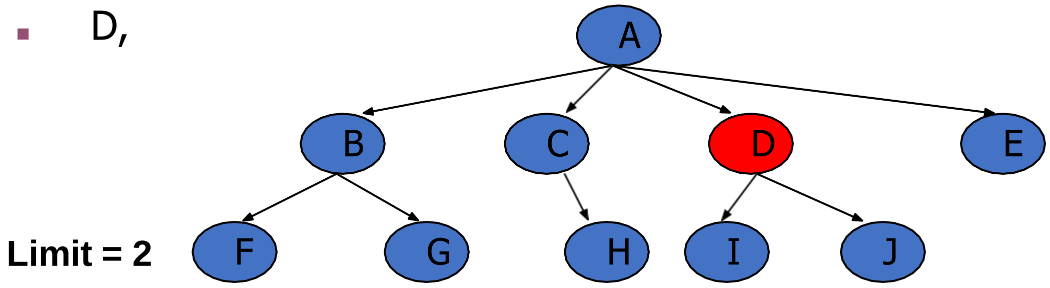
Depth-Limited Search (DLS)

- A,B,F,
- G,
- C,H,



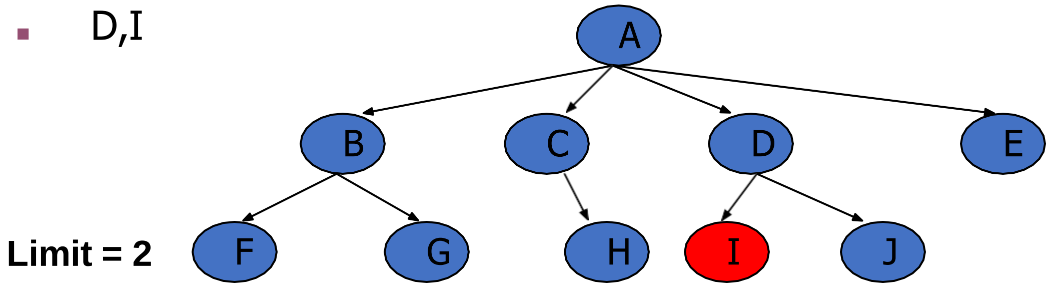
Depth-Limited Search (DLS)

- A,B,F,
- G,
- C,H,
- D,



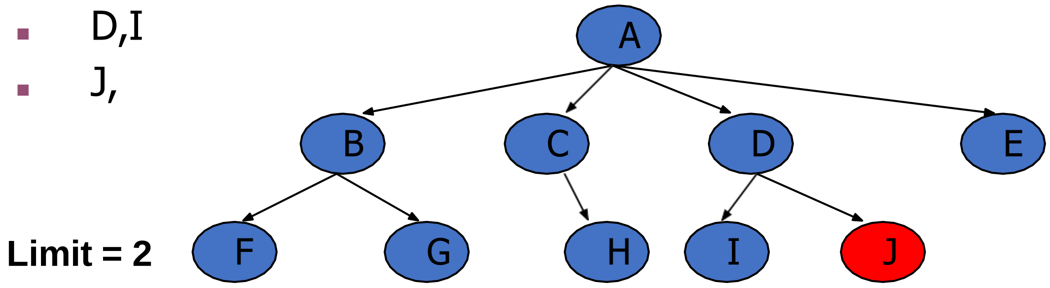
Depth-Limited Search (DLS)

- A,B,F,
- G,
- C,H,
- D,I



Depth-Limited Search (DLS)

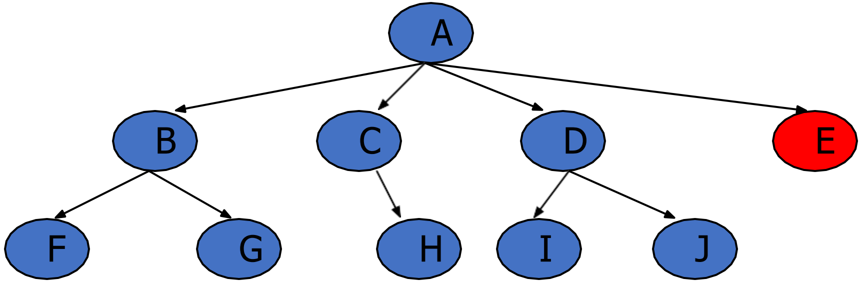
- A,B,F,
- G,
- C,H,
- D,I
- J,



Depth-Limited Search (DLS)

- A,B,F,
- G,
- C,H,
- D,I
- J,
- E

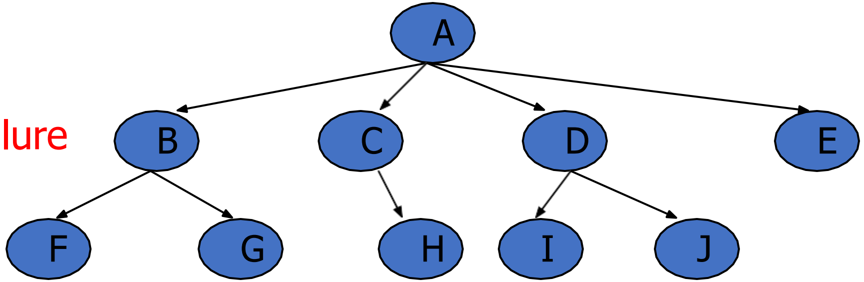
Limit = 2



Depth-Limited Search (DLS)

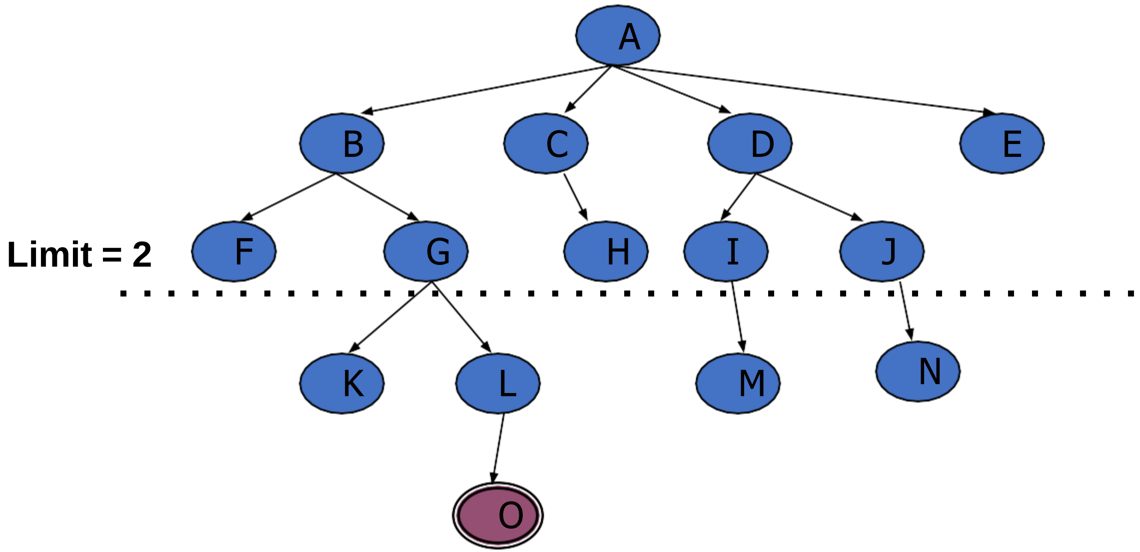
- A,B,F,
- G,
- C,H,
- D,I
- J,
- E, Failure

Limit = 2



Depth-Limited Search (DLS)

- DLS algorithm returns Failure (no solution)
- The reason is that the goal is beyond the limit (Limit =2): the goal depth is (d=4)



Depth-Limited Search (DLS)

Main idea: Expand node at the deepest level, but limit depth to L.

Implementation: Enqueue nodes in LIFO (last-in, first-out) order. But limit depth to L

- Complete? Yes if there is a goal state at a depth less than L else No
- Optimal? No
- Time Complexity: $O(b^L)$, where L is the cutoff.
- Space Complexity: O(bL), where L is the cutoff.

Iterative-Deepening Search (IDS)

- IDS is a general strategy often used with depth first search, that *finds the best depth limit.*
- Gradually increases the limit, first 0, then 1, then 2 and so on until a goal is found.
- This occurs when the depth limit reaches d, the depth of shallowest goal node.

Iterative-Deepening Search (IDS)

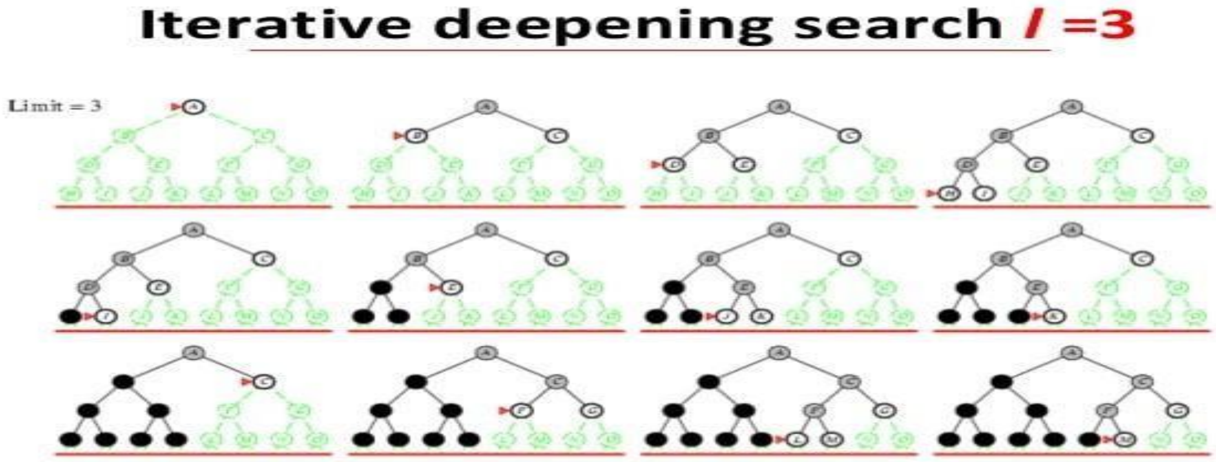
- IDS combines benefits of both DFS and BFS.

- Like depth-first search, its **memory** requirements are very modest: O(bd) to be precise.

- Like breadth-first search, it is **complete** when the branching factor is finite and **optimal** when the path cost is a nondecreasing function of the depth of the node.

Iterative-Deepening Search (IDS)

- *In general, iterative deepening is the* ***preferred uninformed search method*** *when there is a large search space and the depth of the solution is not known*

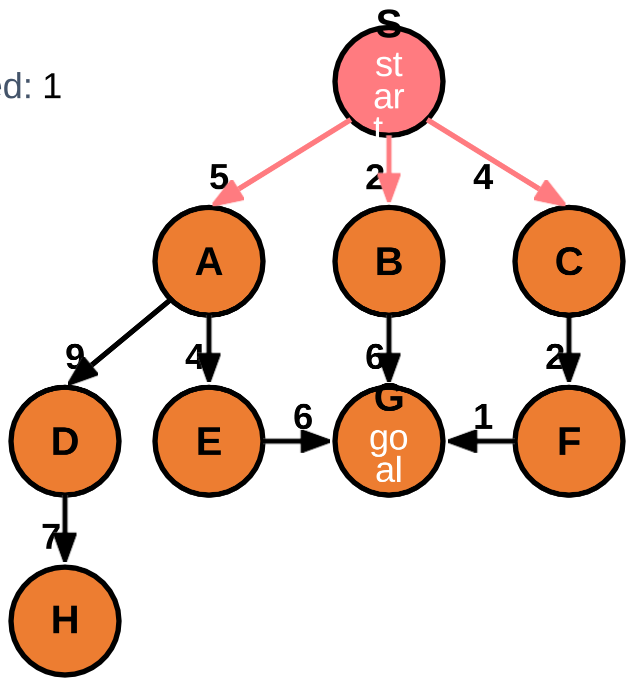


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 1, # of nodes tested: 1, expanded: 1

expnd. node	Frontier
	{S}
S not goal	{A,B,C}

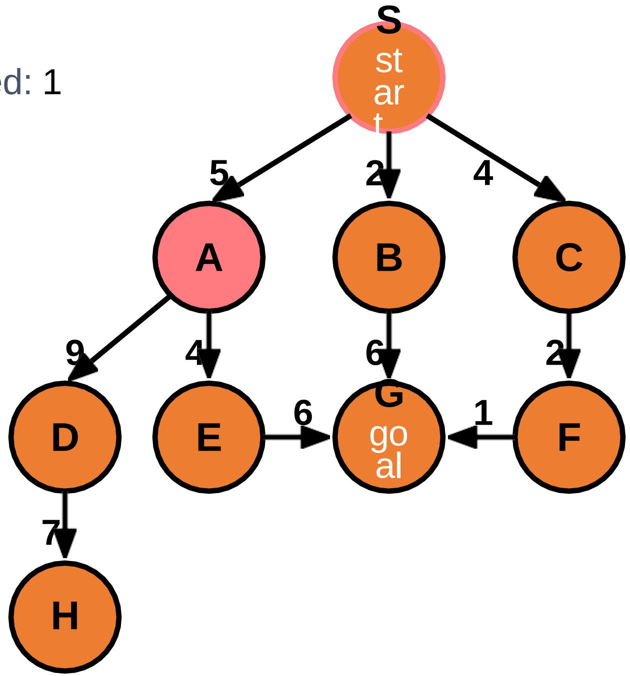


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 1, # of nodes tested: 2, expanded: 1

expnd. node	Frontier
	{S}
S	{A,B,C}
A not goal	{B,C} no expand

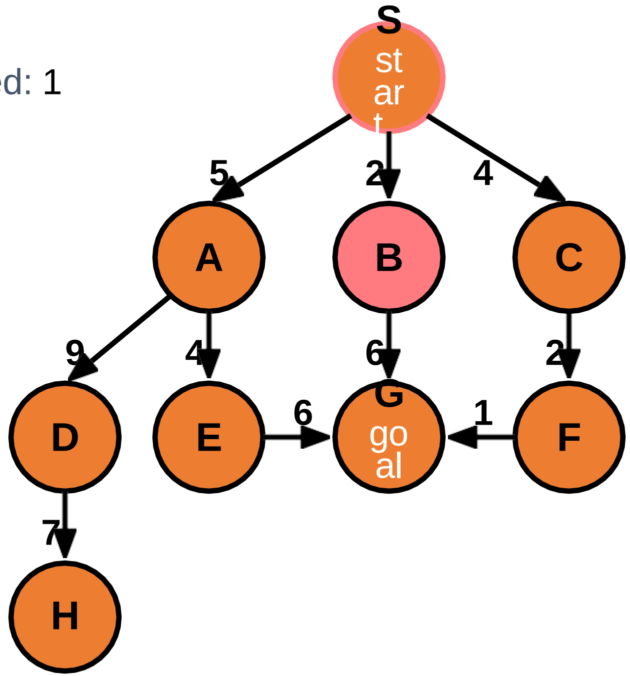


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 1, # of nodes tested: 3, expanded: 1

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{B,C}
B not goal	{C} no expand

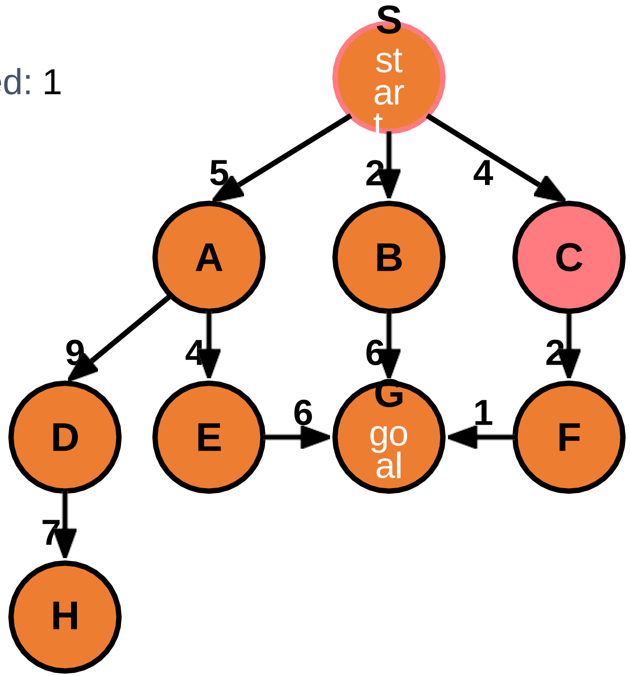


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 1, # of nodes tested: 4, expanded: 1

expnd. node	**Frontier**
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C not goal	{ } no expand-FAIL

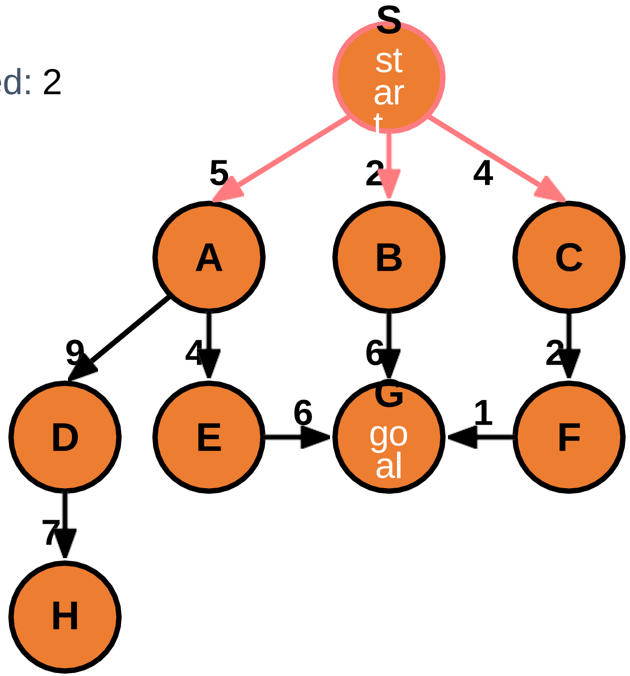


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 4, expanded: 2

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C	{ }
S no test	{A,B,C}

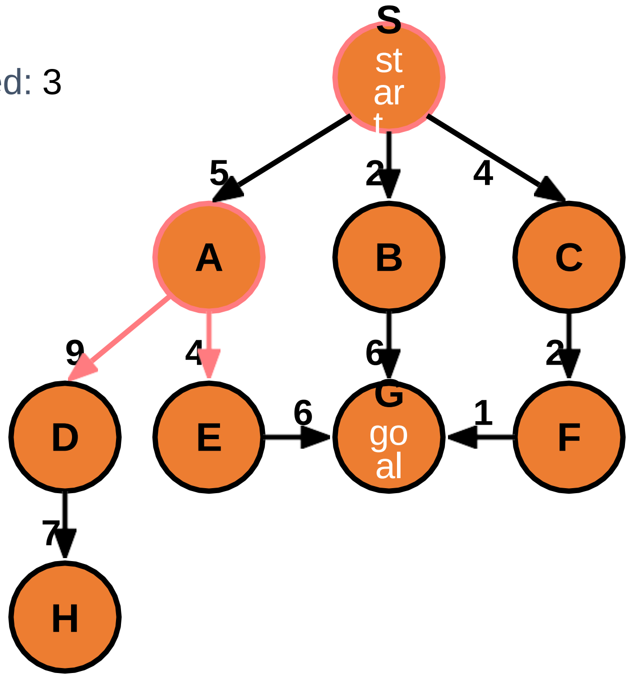


Iterative-Deepening Search (IDS)

`deepeningSearch(problem)`

depth: 2, # of nodes tested: 4, expanded: 3

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C	{ }
S	{A,B,C}
A no test	{D,E,B,C}

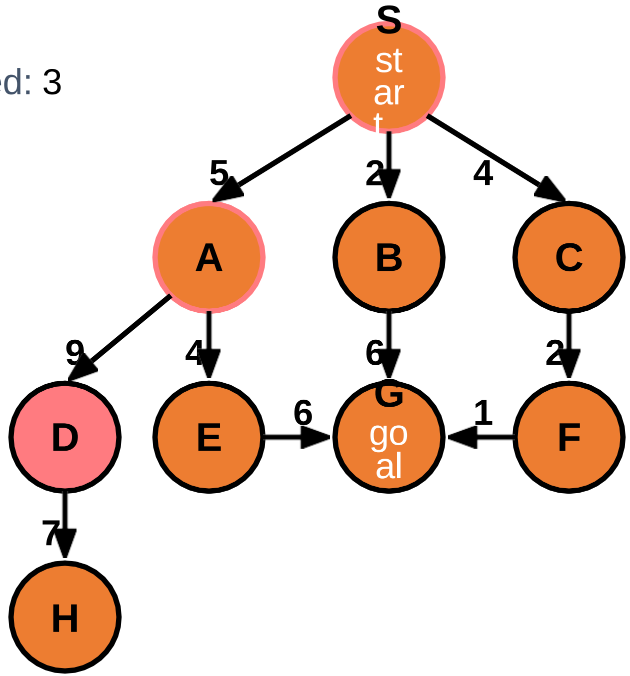


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 5, expanded: 3

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C	{ }
S	{A,B,C}
A	{D,E,B,C}
D not goal	{E,B,C} no expand

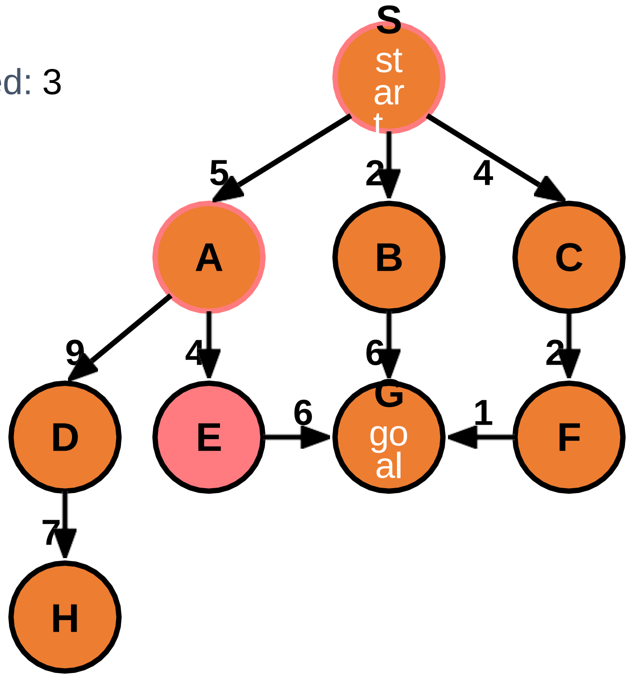


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 6, expanded: 3

expnd. node	**Frontier**
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C	{ }
S	{A,B,C}
A	{D,E,B,C}
D	{E,B,C}
E not goal	{B,C} no expand

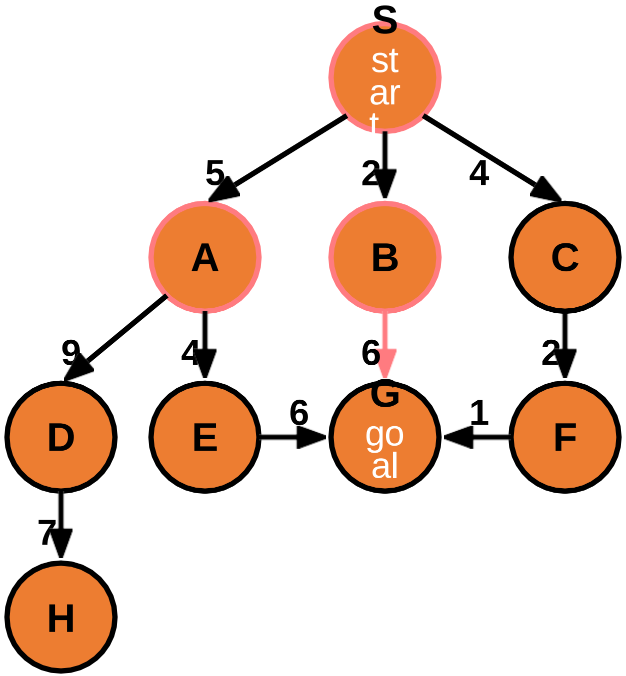


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 6, expanded: 4

expnd. node	**Frontier**
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C	{ }
S	{A,B,C}
A	{D,E,B,C}
D	{E,B,C}
E	{B,C}
B no test	{G,C}

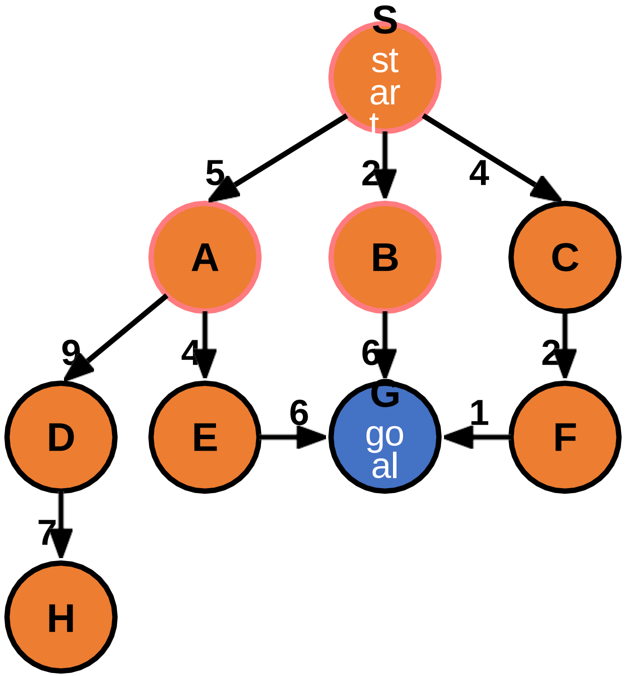


Iterative-Deepening Search (IDS)

`deepeningSearch(problem)`

depth: 2, # of nodes tested: 7, expanded: 4

expnd. node	**Frontier**
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C	{ }
S	{A,B,C}
A	{D,E,B,C}
D	{E,B,C}
E	{B,C}
B	{G,C}
G goal	{C} no expand

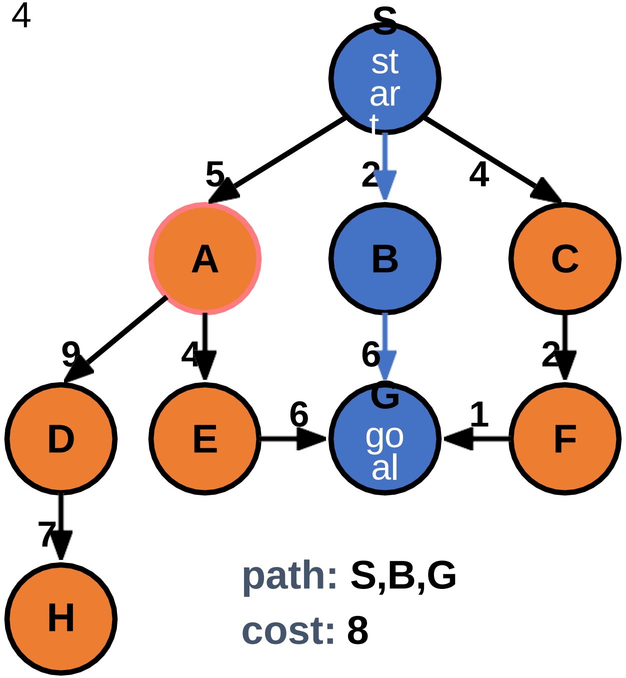


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 7, expanded: 4

expnd. node	Frontier
	{S}
S	{A,B,C}
A	{B,C}
B	{C}
C	{ }
S	{A,B,C}
A	{D,E,B,C}
D	{E,B,C}
E	{B,C}
B	{G,C}
G	{C}



path: S,B,G

cost: 8

Properties of iterative deepening search

- Complete? Yes
- Time? $O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

Bidirectional search

- Run two simultaneous searches
  - One forwards, from the initial state
  - Other backward from the goal.
- Stops when the two searches meet in the middle
- **Breadth First Search** is applied at both sides of searches

Bidirectional search

- Finds shortest path from initial state to goal
- Reduce the **search time** by searching forward from the start and backward from the goal simultaneously

Bidirectional search

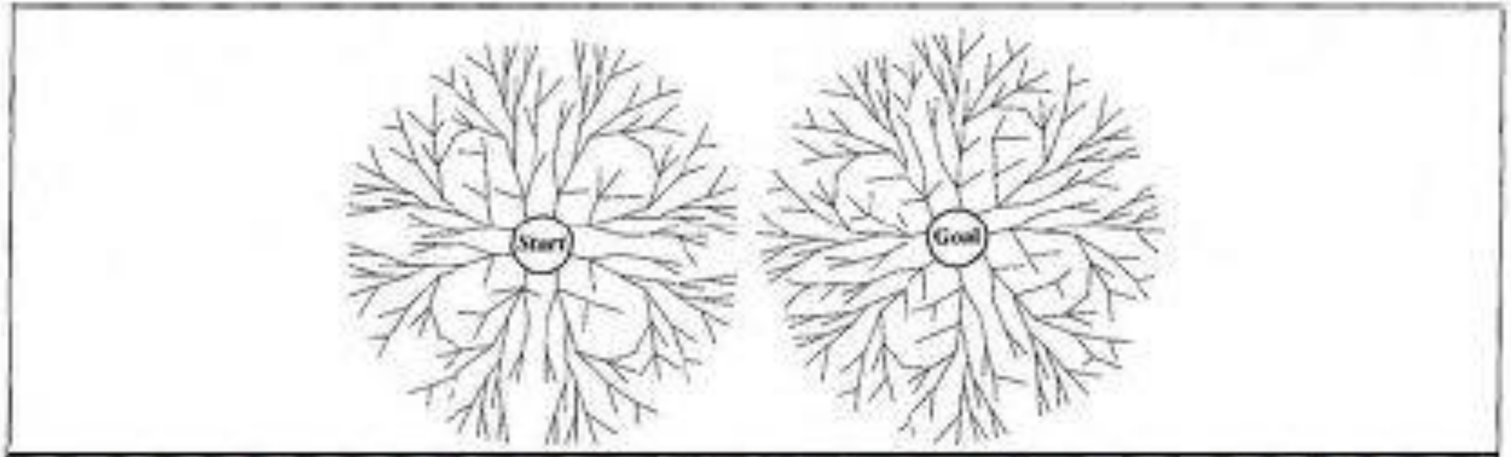
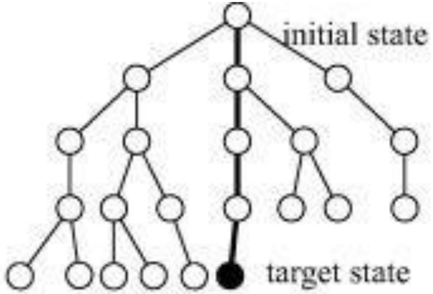
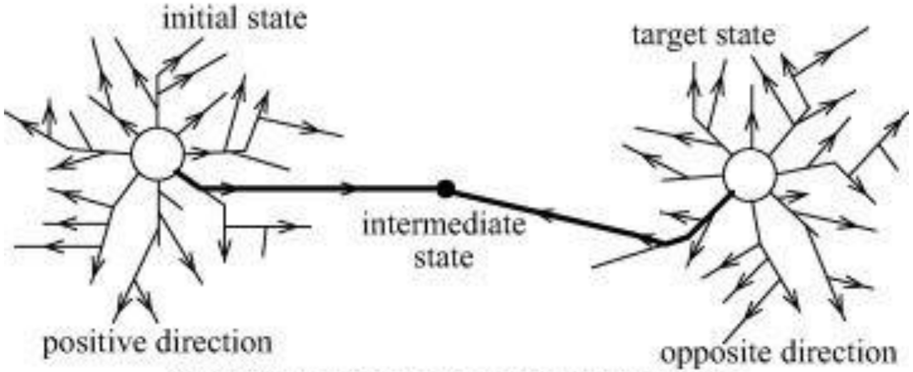


Figure 3.16 A schematic view of a bidirectional search that is about to succeed, when a branch from the start node meets a branch from the goal node.

Bidirectional Search

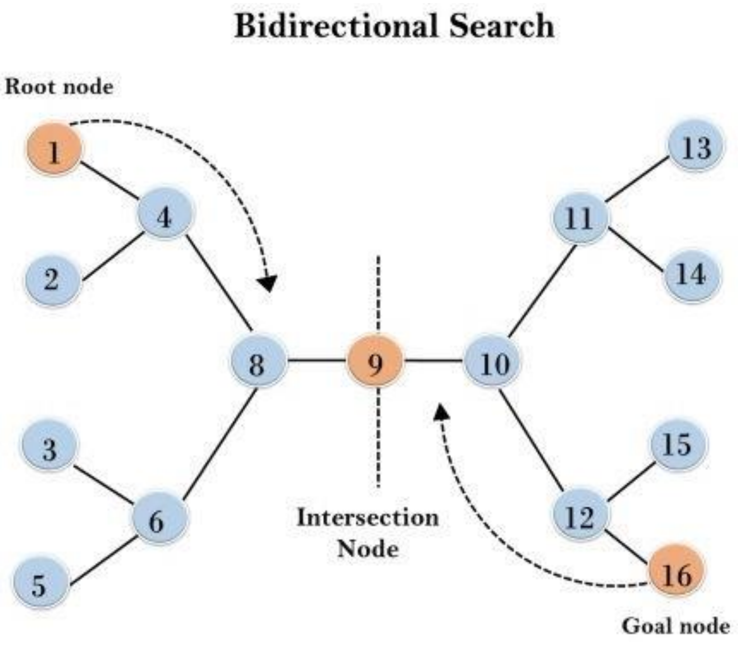


(a) Traditional breadth-first search

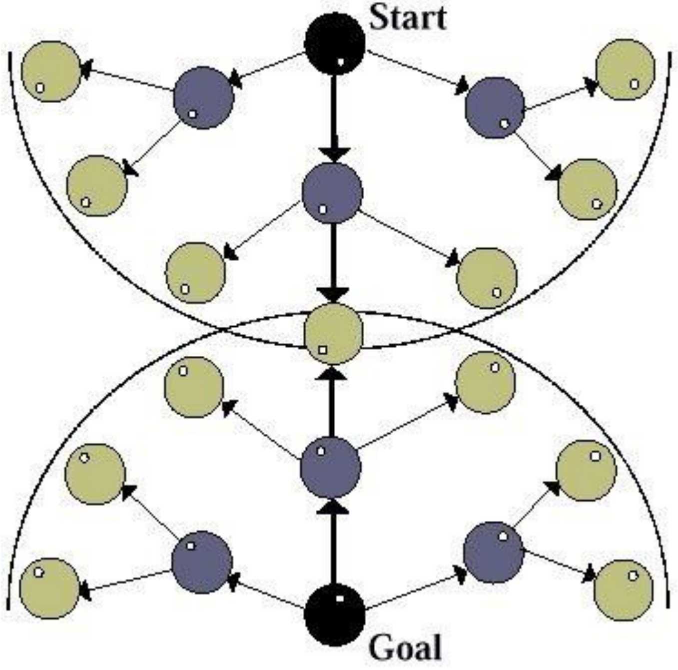


(b) bidirectional breadth-first search

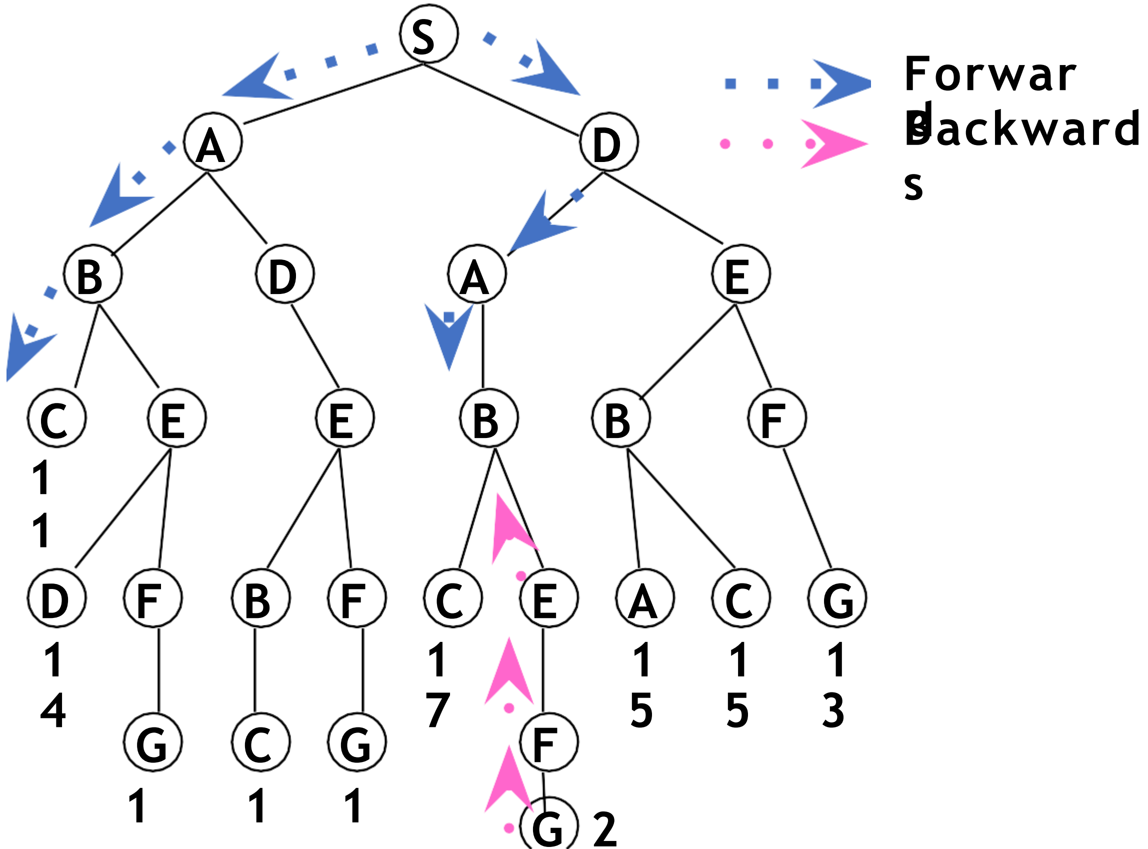
Bidirectional Search



Bidirectional Search



Bidirectional search



Bidirectional Search

- **Complete : Yes**
  - Bidirectional search is complete if BFS is used in both searches.
- **Optimality : Yes**
  - It is optimal if BFS is used for search and paths have uniform cost.
- **Time and Space Complexity :**
  - O($b^{d/2}$)

Comparing uninformed search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Reading Reference

The content of this lecture comprise of material from Chapter 3 of the course book.