Artificial Intelligence

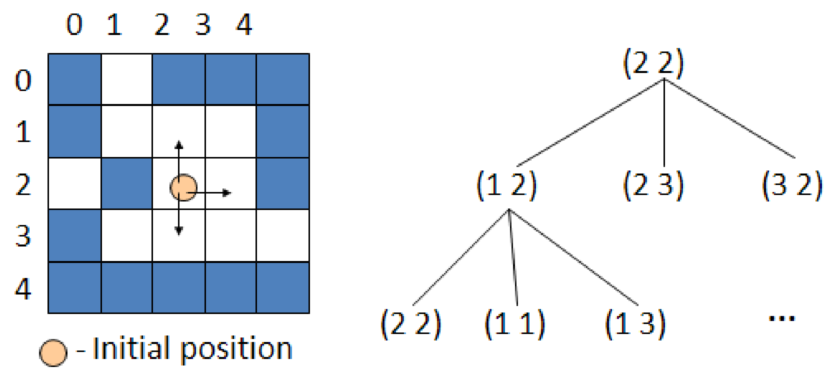
COURSE INSTRUCTOR: MUHAMMAD SAIF UL ISLAM

Outline

- Uninformed search
- Search strategies
- Problem formulation
- Uniform Search Strategies
  - Depth First Search
  - Breadth First Search
  - Uniform cost Search
  - Depth-Limit Search
  - Iterative Deepening Search
  - Bidirectional Search

Uninformed Search Strategies

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



Breadth-First Search (BFS)

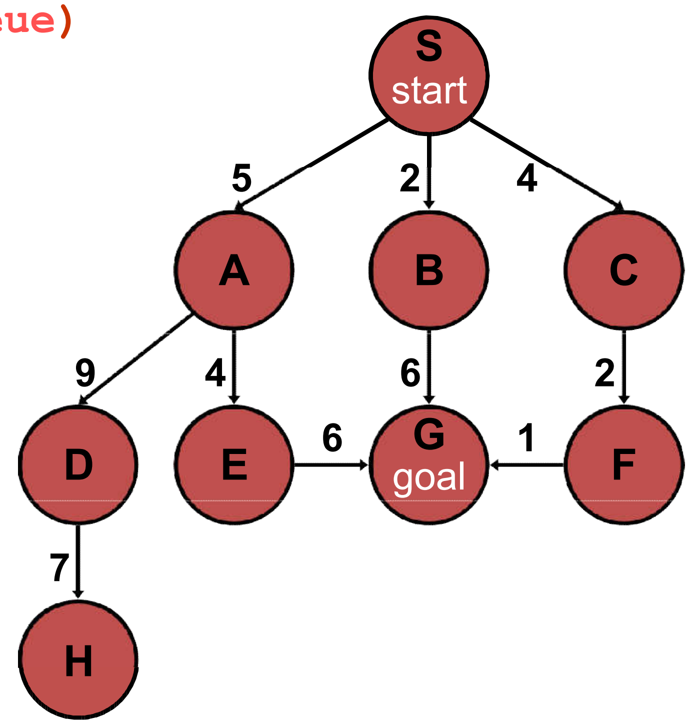
- Root node is expanded first -> successor of root node -> their successors
- All nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded
- Easy to implement by using FIFO queue

Breadth-First Search (BFS)

generalSearch(problem, queue)

# of nodes tested: 0, expanded: 0

expnd. node	Frontier list
	{S}

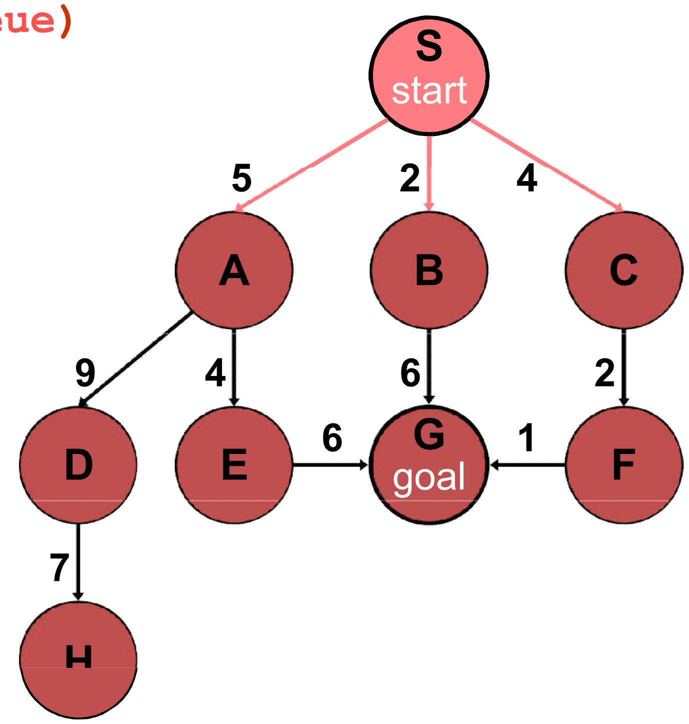


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 1, expanded: 1

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

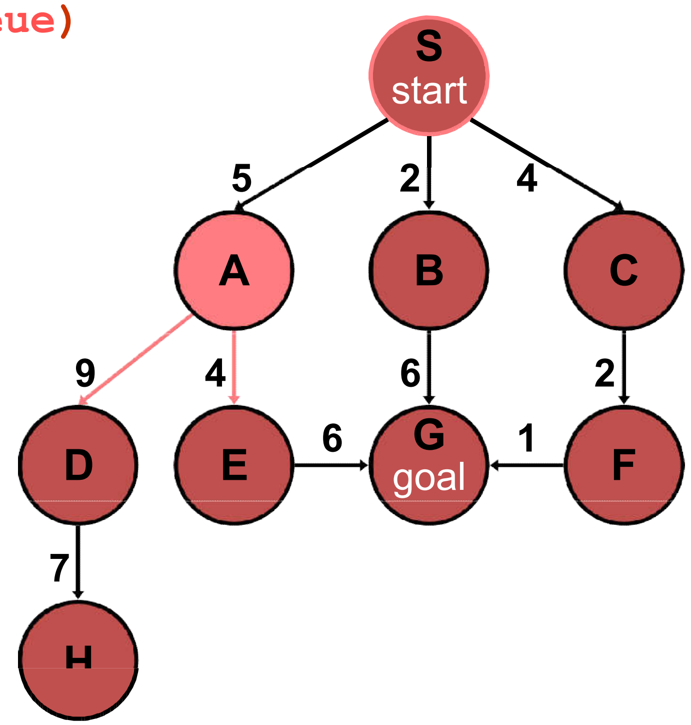


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 2, expanded: 2

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

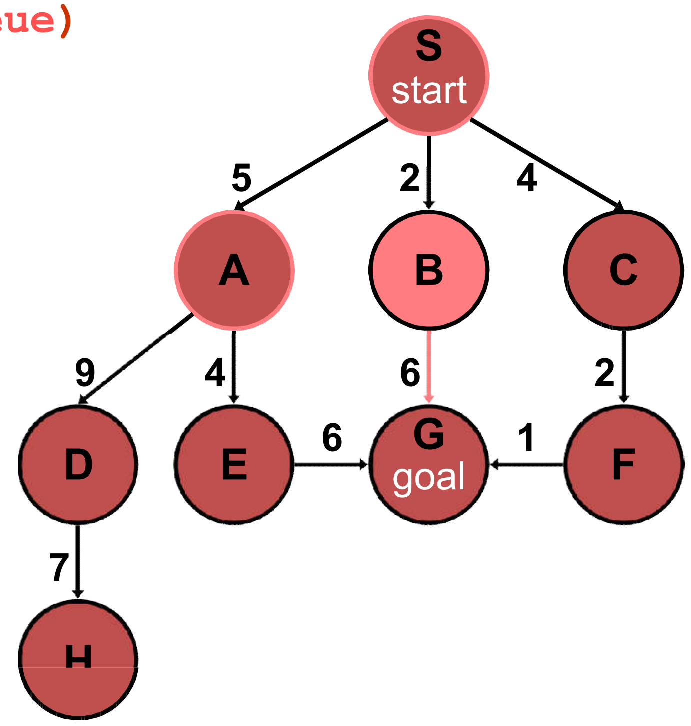


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 3, expanded: 3

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

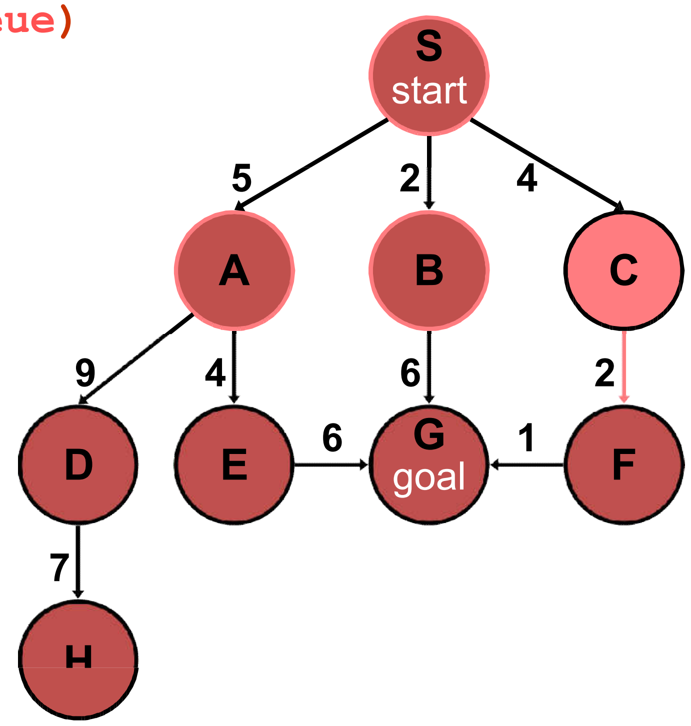


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 4, expanded: 4

expnd. 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}

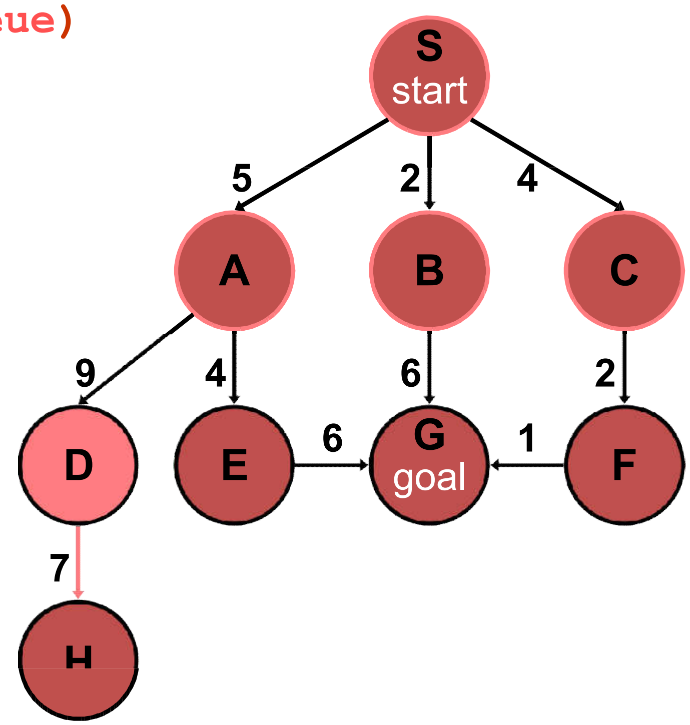


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 5, expanded: 5

expnd. 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}

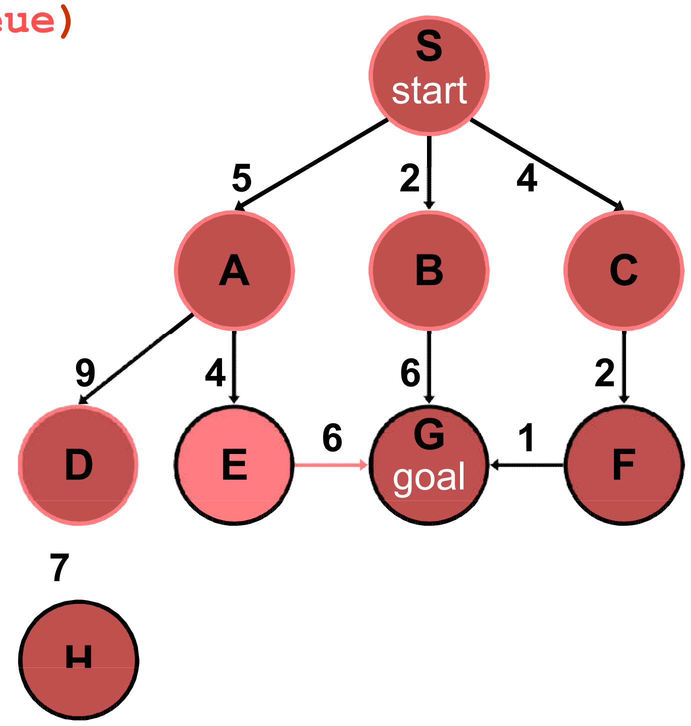


Breadth-First Search (BFS)

generalSearch(problem, queue)

# of nodes tested: 6, expanded: 6

expnd. 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}

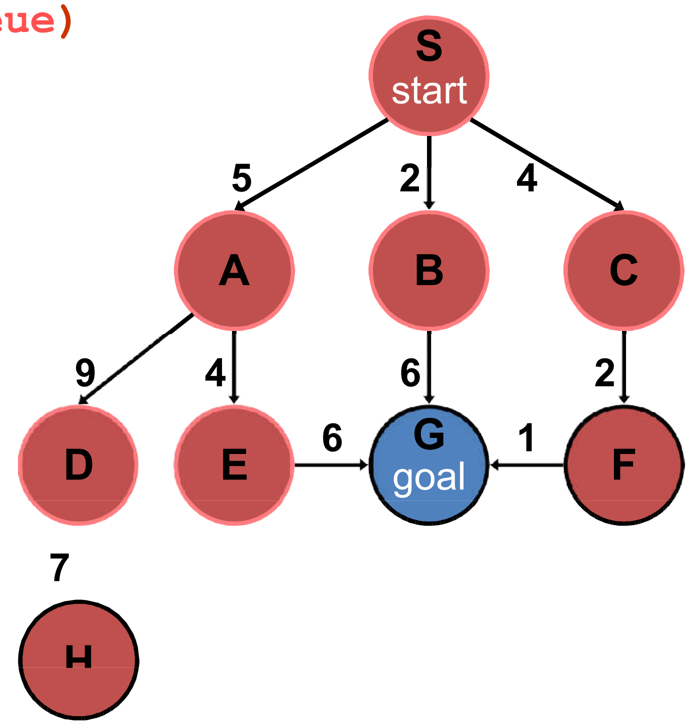


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

expnd. 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

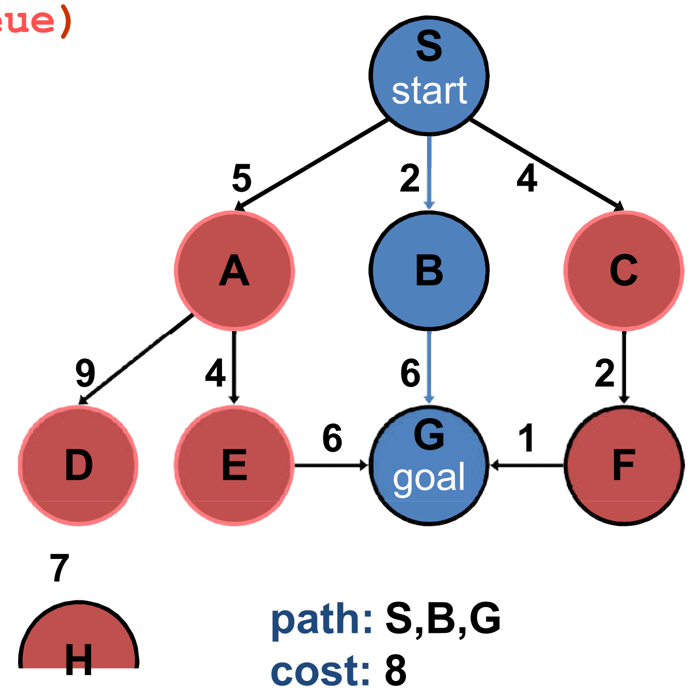


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

expnd. 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	{F,H,G}



path: S,B,G
cost: 8

Properties of Breadth-First Search

- Completeness: Yes, if *b* is finite
- Optimality: Yes (assuming cost = 1 per step)
- Time complexity: $1+b+b^2+\ldots+b^d = O(b^d)$, i.e., exponential in *d*
- Space complexity: $O(b^d)$

b -	maximum branching factor of the search tree
d -	depth of the least-cost solution
m -	maximum depth of the state space (may be infinite)

Exponential Growth

Depth	Nodes	Time		Memory	
2	110	0.11	millisecond	107	kbytes
4	11,110	11	millisecond	10.6	megabytes
6	10^6	1.1	seconds	1	gigabyte
8	10^8	2	minutes	103	gigabytes
10	10^{10}	3	hours	10	terabytes
12	10^{12}	13	days	1	petabyte
14	10^{14}	3.5	years	99	petabytes
16	10^{16}	350	years	10	exabytes

Time and memory requirements for breadth-first search, assuming a branching factor of 10, 100 bytes per node and searching 1000 nodes/second

Depth-First Search (DFS)

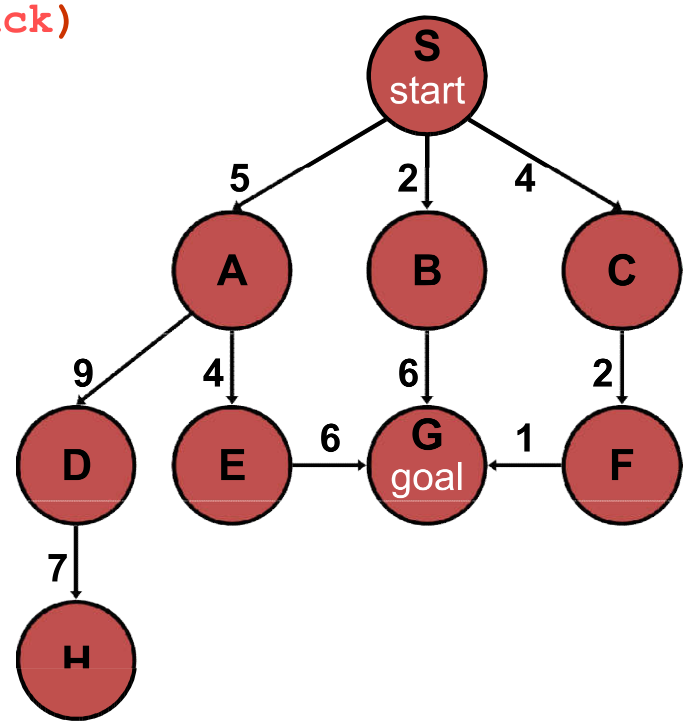
- Expands the deepest node in the current frontier of the search tree until the nodes have no successors, then backs up to the next deepest node that still has unexplored successors.
- Use LIFO queue
  - – The most recently generated node is chosen for expansion, the deepest unexpanded node
- Use recursive

Depth-First Search (DFS)

`generalSearch(problem, stack)`

# of nodes tested: 0, expanded: 0

expnd. node	Frontier
	{S}

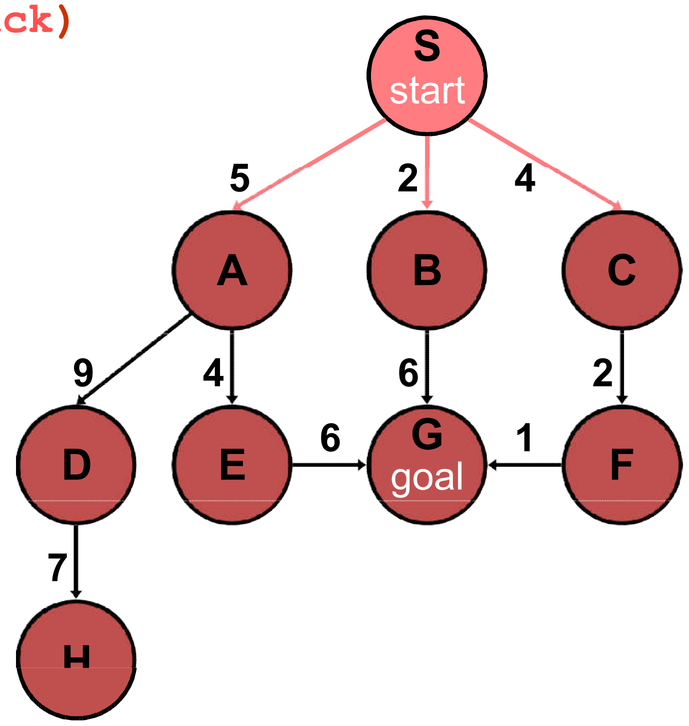


Depth-First Search (DFS)

generalSearch(problem, stack)

# of nodes tested: 1, expanded: 1

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

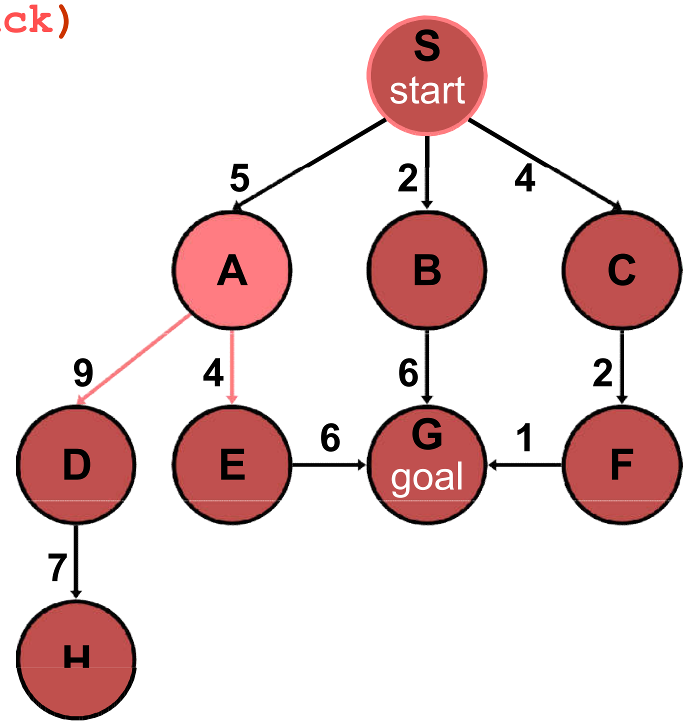


Depth-First Search (DFS)

`generalSearch(problem, stack)`

# of nodes tested: 2, expanded: 2

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

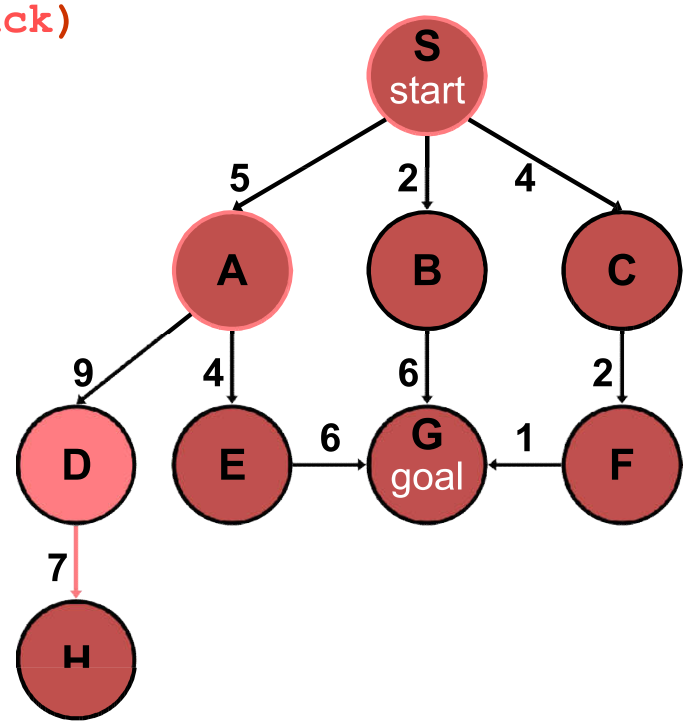


Depth-First Search (DFS)

`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}

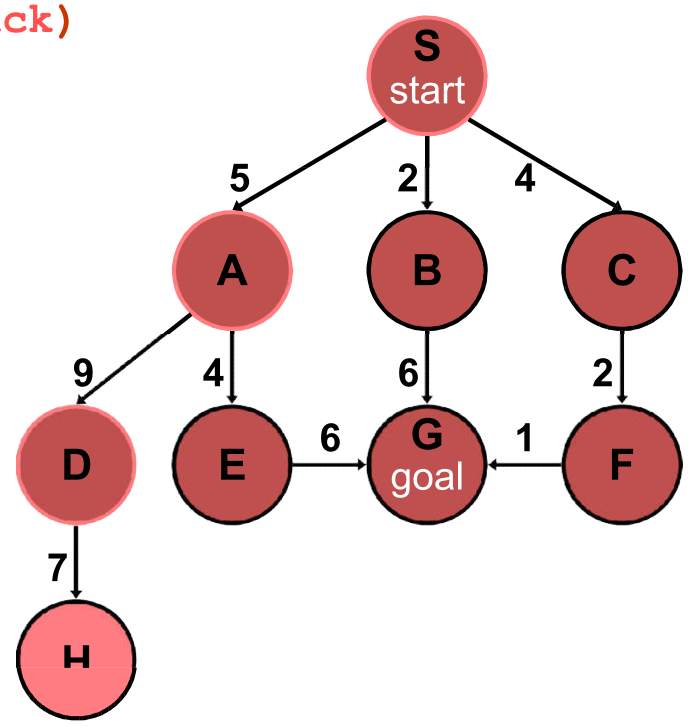


Depth-First Search (DFS)

`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}

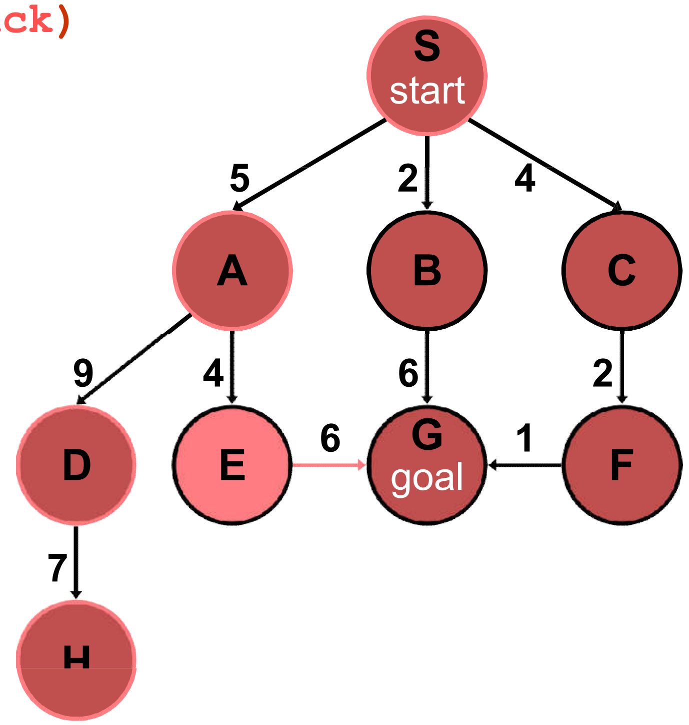


Depth-First Search (DFS)

`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}

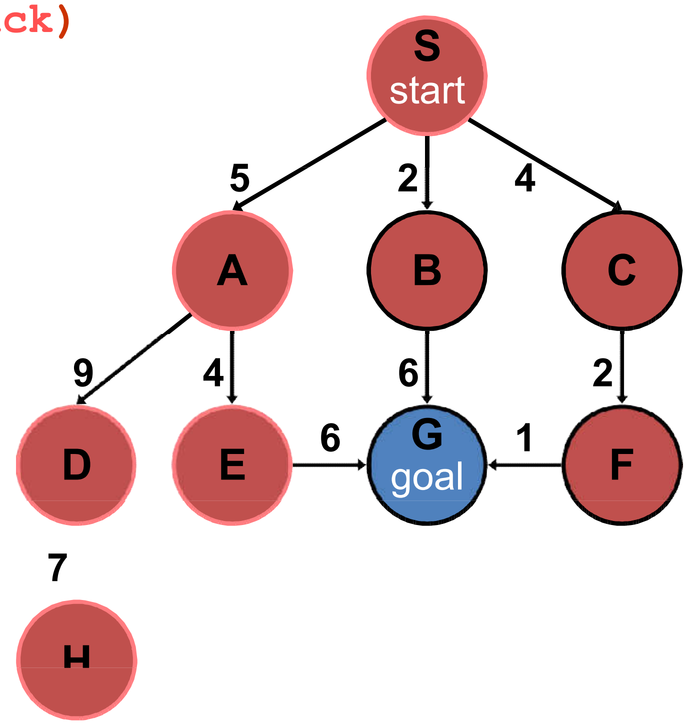


Depth-First Search (DFS)

`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

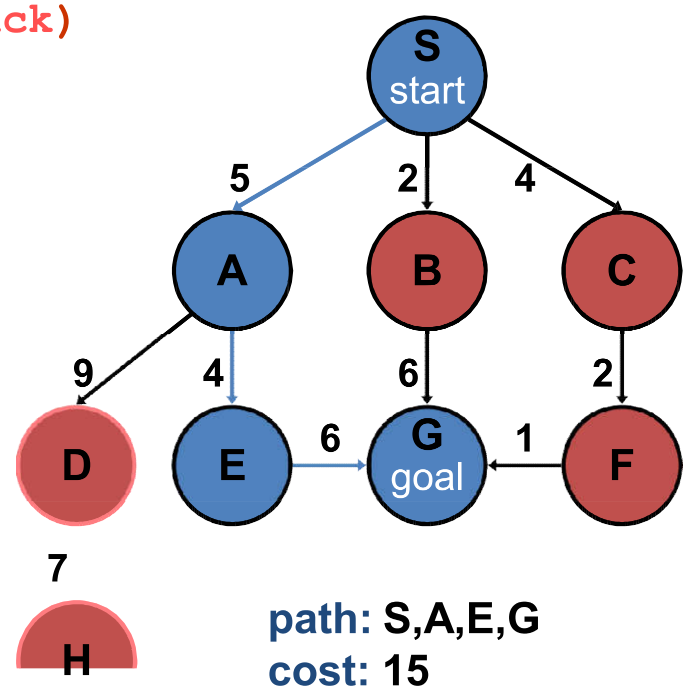


Depth-First Search (DFS)

`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

Properties of Depth-First Search

- Completeness: (yes, if finite state space)
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$
- Optimality: No

Remember:

b = branching factor

m = max depth of search tree

Uniform-Cost Search (UCS)

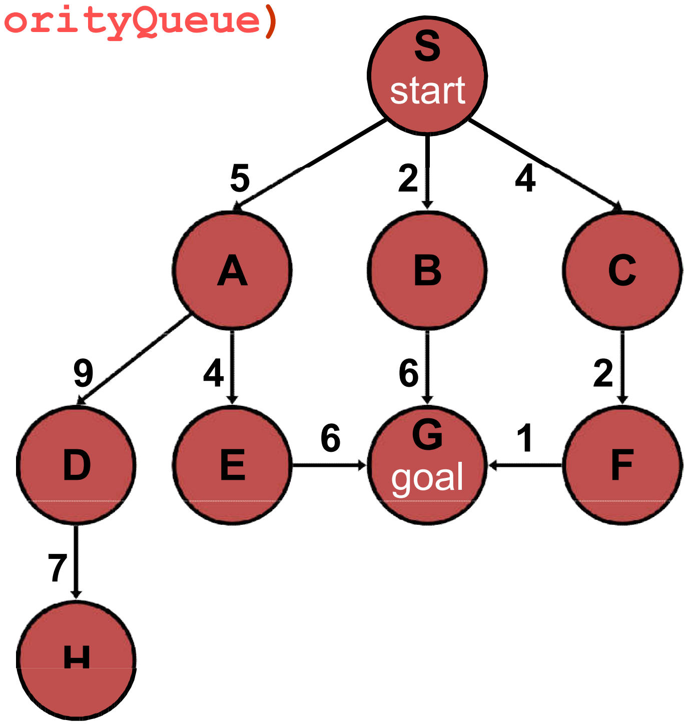
- Use a “Priority Queue” to order nodes on the *Frontier* list, sorted by path cost
- Let $g(n) =$ cost of the path from start node s to current node n
- Sort nodes by increasing the value of g

Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

# of nodes tested: 0, expanded: 0

expnd. node	Frontier list
	{S}

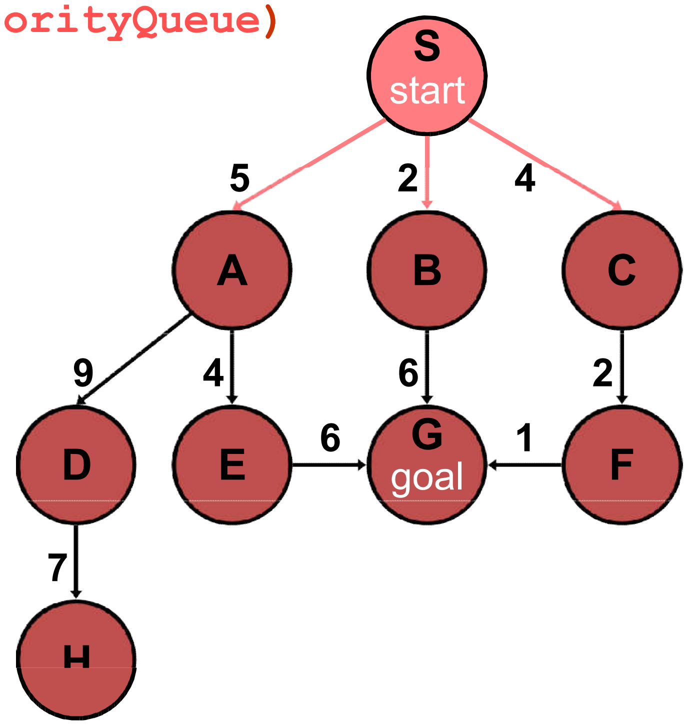


Uniform-Cost Search (UCS)

`generalSearch(problem, priorityQueue)`

# of nodes tested: 1, expanded: 1

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

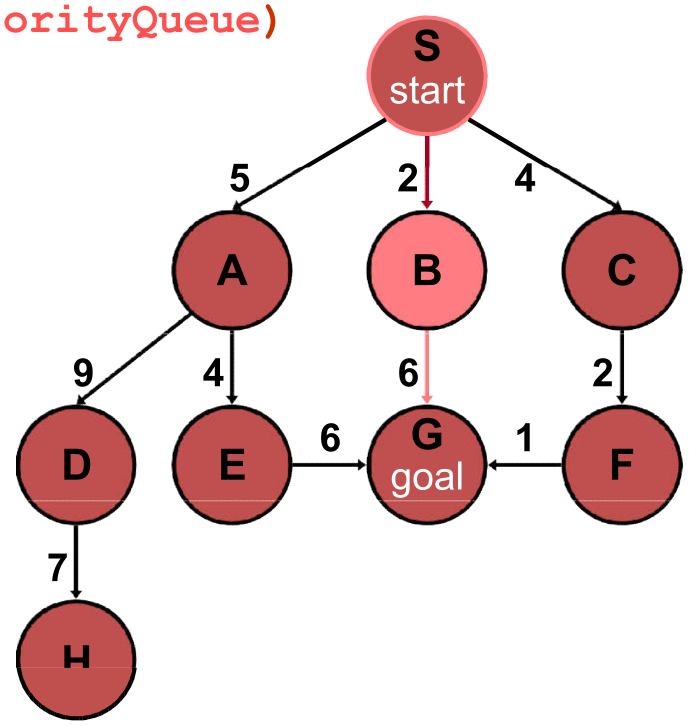


Uniform-Cost Search (UCS)

`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}

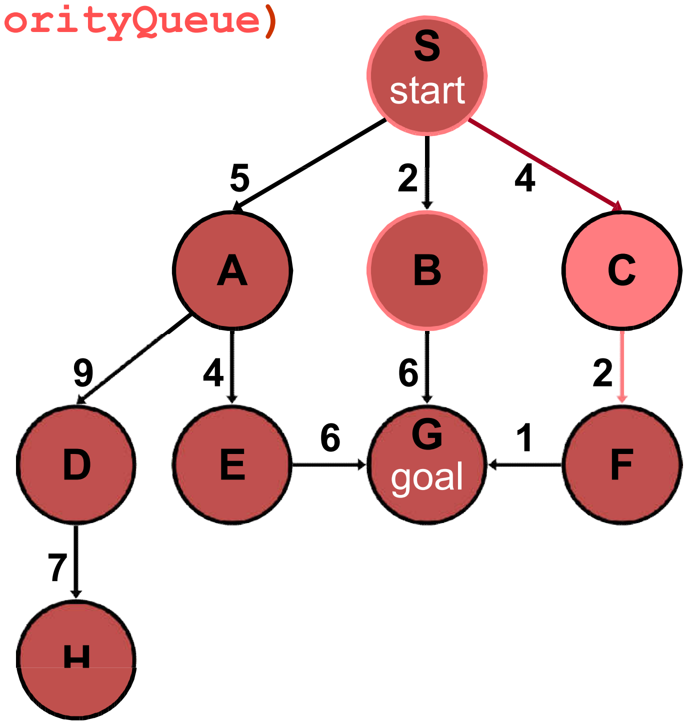


Uniform-Cost Search (UCS)

`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}

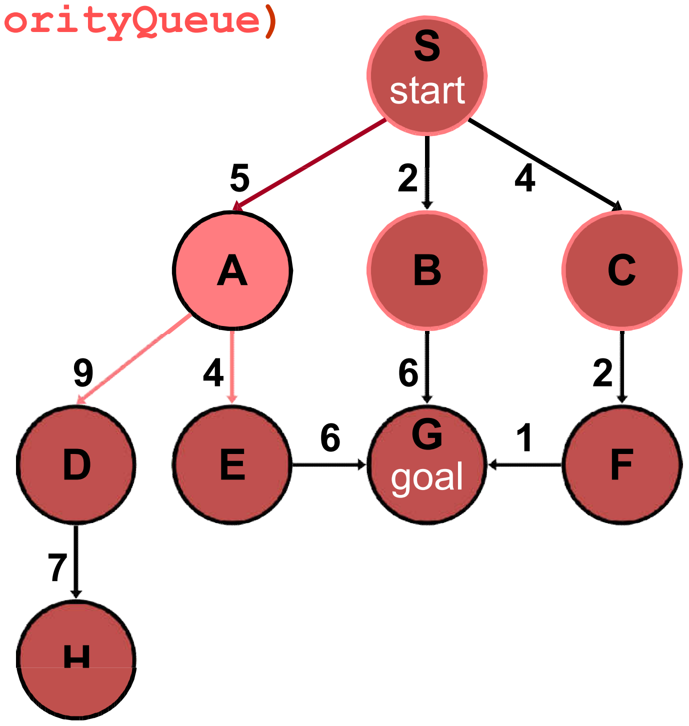


Uniform-Cost Search (UCS)

`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}

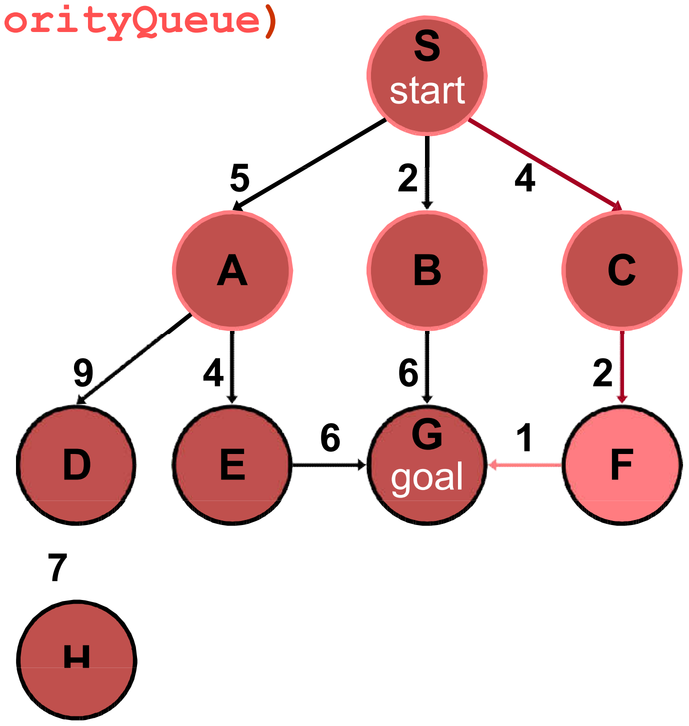


Uniform-Cost Search (UCS)

`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}

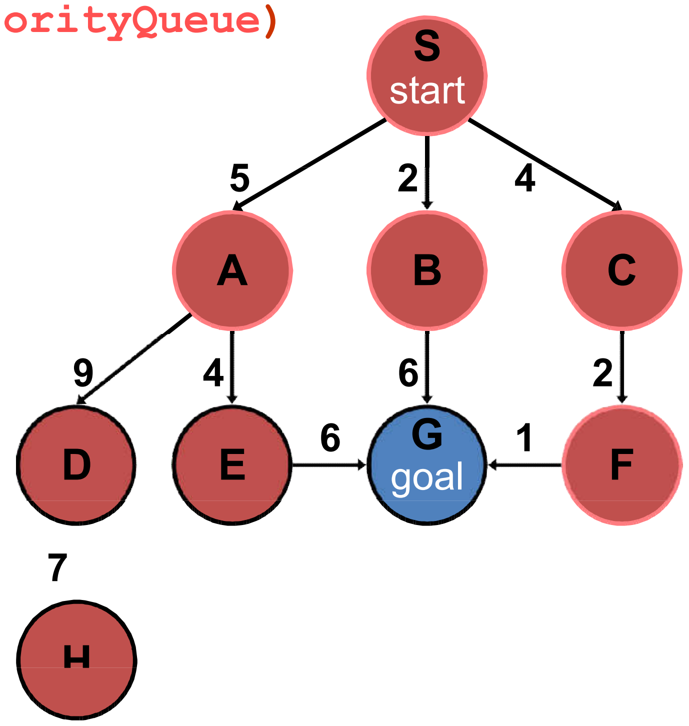


Uniform-Cost Search (UCS)

`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

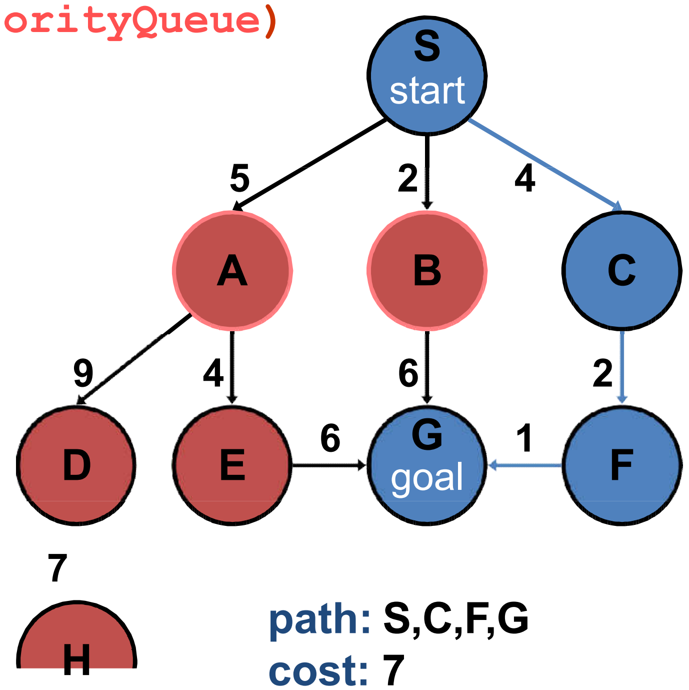


Uniform-Cost Search (UCS)

```
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

Properties of Uniform-Cost Search

- Complete: Yes if arc costs > 0.
- Optimal: Yes
- Time and space complexity: $O(b^d)$ (i.e., exponential)
  - d is the depth of the solution
  - is the branching factor at each non-leaf node

- More precisely, time and space complexity is $O(b^{C^*/\varepsilon})$ where all edge costs $\varepsilon \sum > 0$, and is the best goal path cost

Iterative-Deepening Search (IDS)

- requires modification to DFS search algorithm:
  - do DFS to depth 1
    and treat all children of the start node as leaves
  - if no solution is found, do DFS to depth 2
  - repeat by increasing “depth bound” until a solution found

- Start node is at depth 0

Iterative-Deepening Search (IDS)

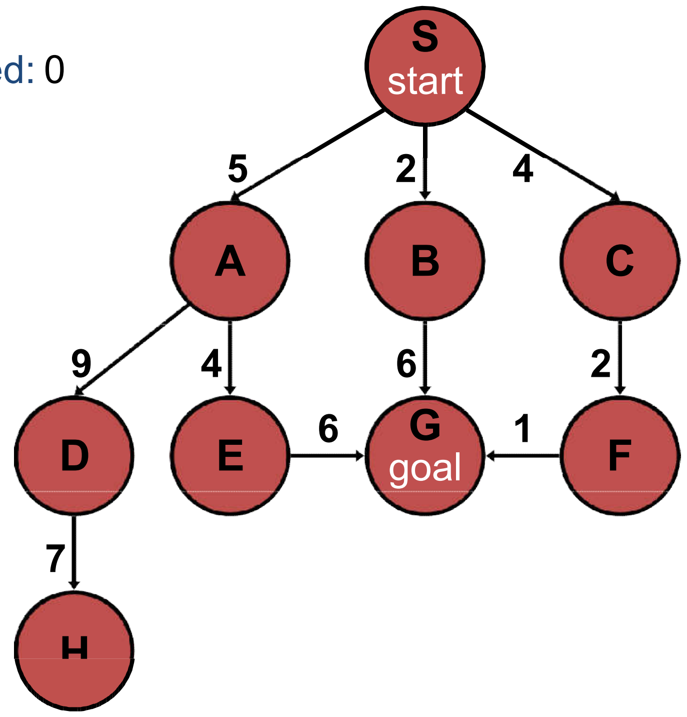
```
Node deepeningSearch (Problem problem) {
  int depth = 1;
  Stack DSnodes = new Stack();
  while (true) {  // while not solved
    Node node = DFS_depthBound(problem, DSnodes, depth);
    // DFS_depthBound limits DFS search to level <= depth
    if (node isn't "failure") return node; // solved
    depth++; // look deeper
  }
}
```

Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 1, # of nodes expanded: 0, tested: 0

expnd. node	Frontier
	{S}

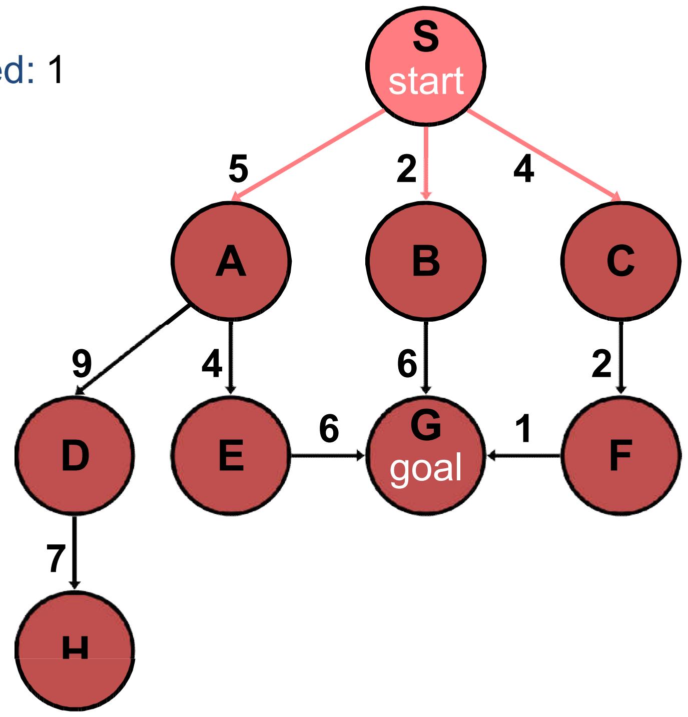


Iterative-Deepening Search (IDS)

`deepeningSearch(problem)`

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

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

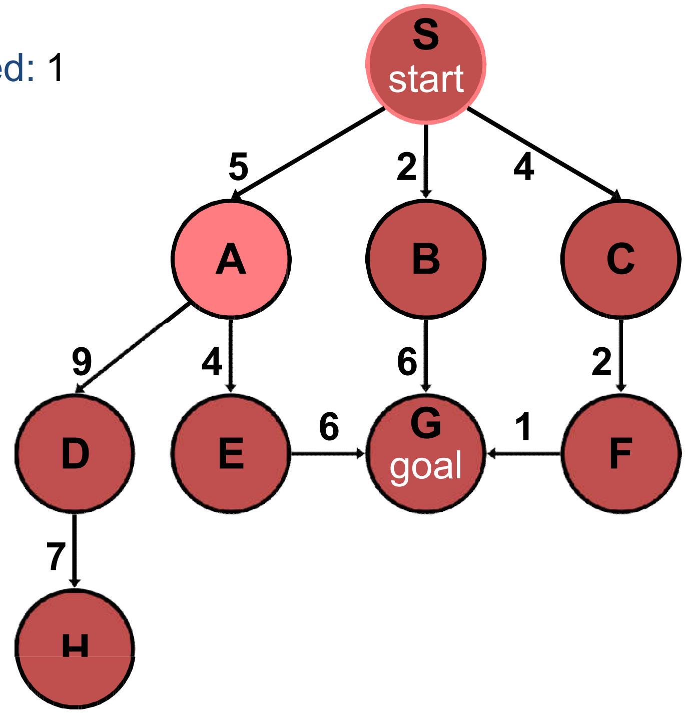


Iterative-Deepening Search (IDS)

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

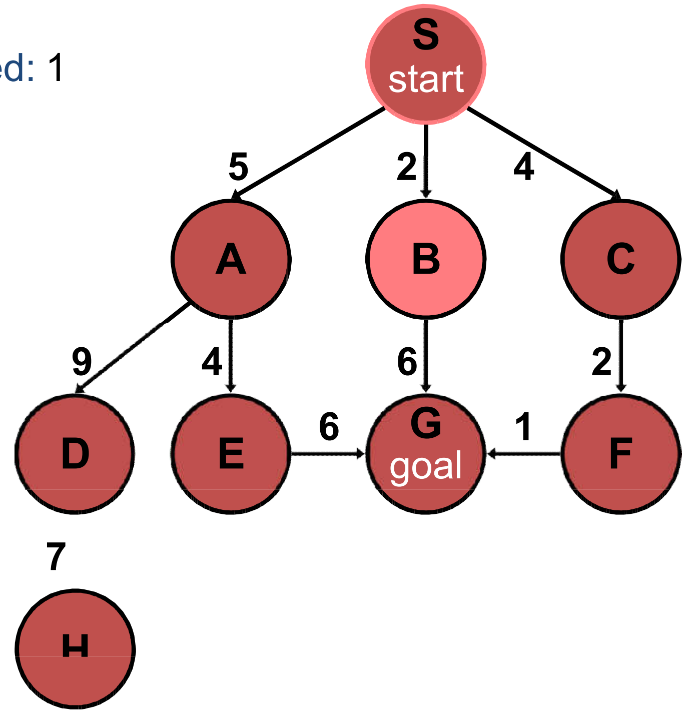


Iterative-Deepening Search (IDS)

```
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

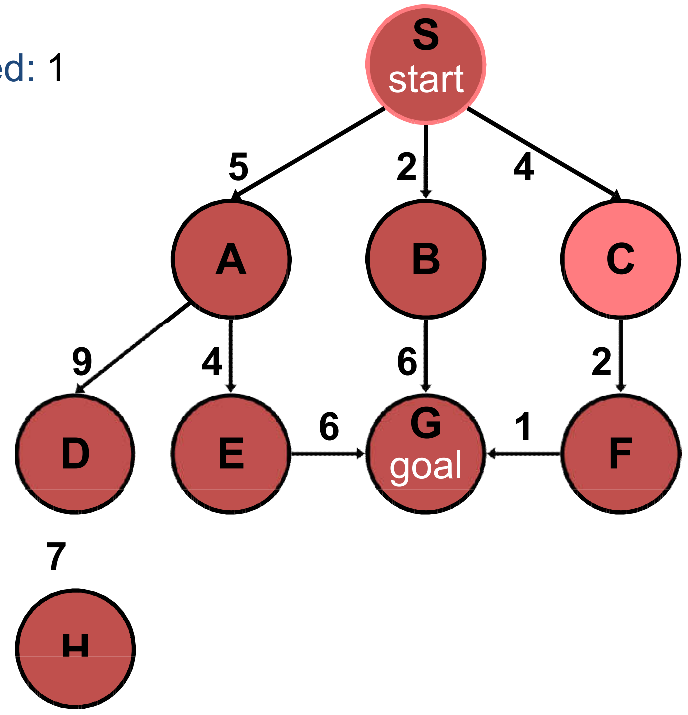


Iterative-Deepening Search (IDS)

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

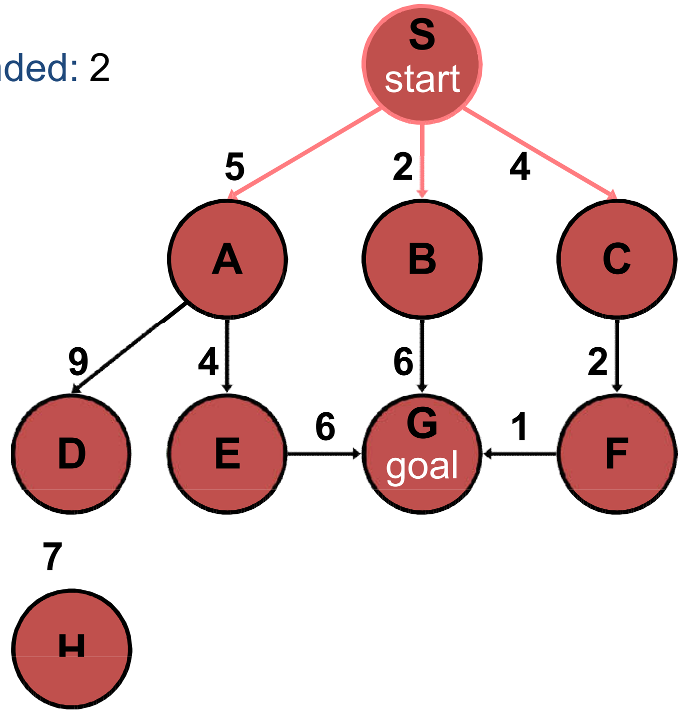


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 4(1), expanded: 2

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

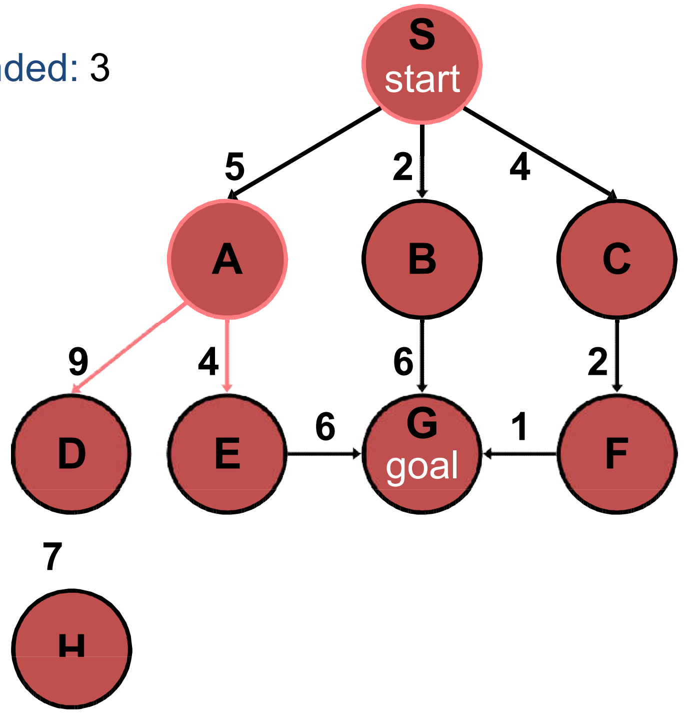


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 4(2), 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}

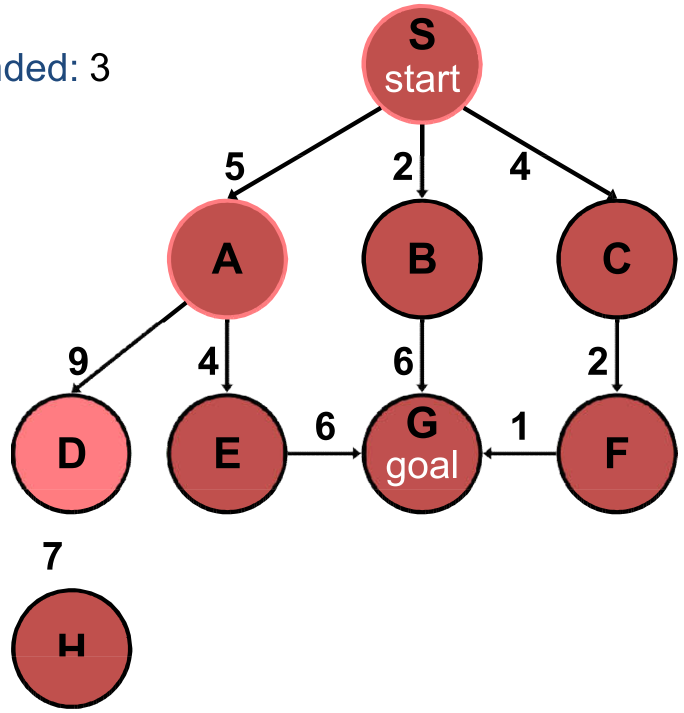


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 5(2), 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

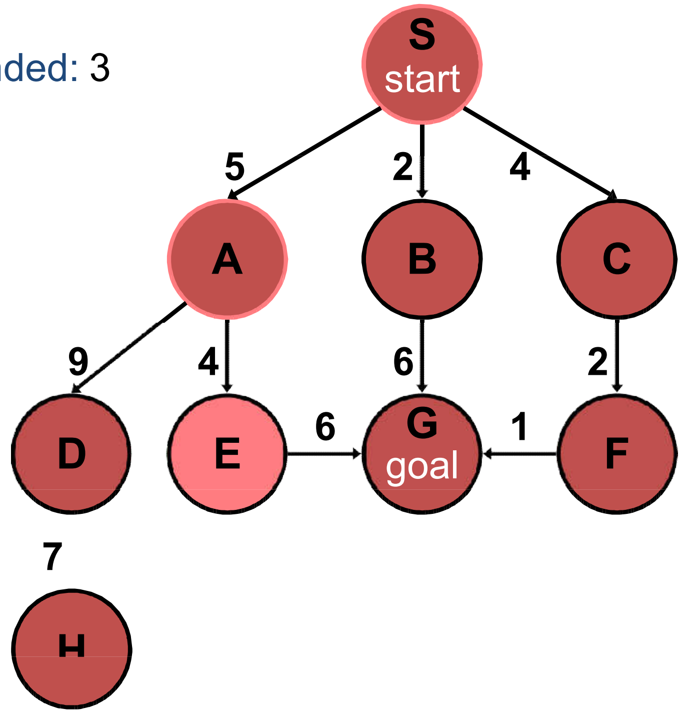


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 6(2), 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

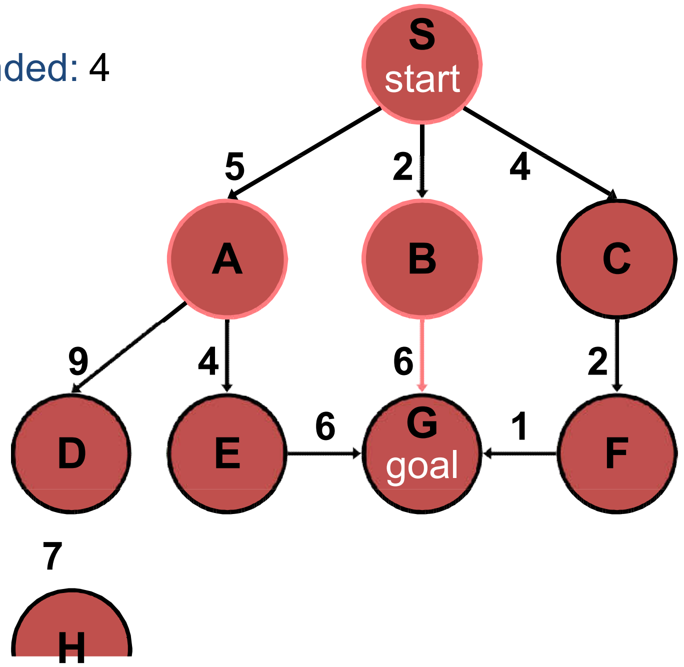


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 6(3), 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}

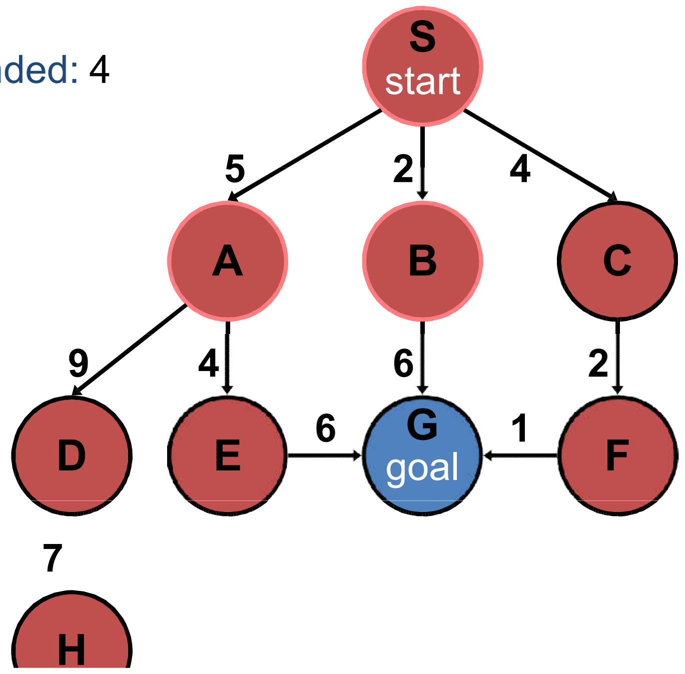


Iterative-Deepening Search (IDS)

`deepeningSearch(problem)`

depth: 2, # of nodes tested: 7(3), 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

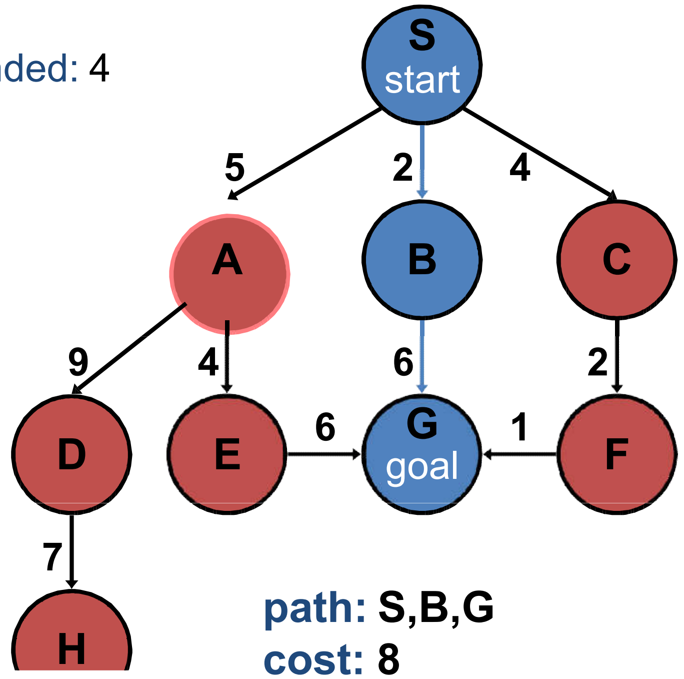


Iterative-Deepening Search (IDS)

deepeningSearch(problem)

depth: 2, # of nodes tested: 7(3), 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

Iterative-Deepening Search (IDS)

- Has advantages of BFS
  - completeness
  - optimality as stated for BFS

- Has advantages of DFS
  - limited space
  - in practice, even with the redundant effort it still finds longer paths more quickly than BFS

Iterative-Deepening Search (IDS)

- Space complexity: $O(bd)$ (i.e., linear like DFS)

- Time complexity is a little worse than BFS or DFS
  - because nodes near the top of the search tree are generated multiple times (redundant effort)

- Worst case time complexity: $O(b^d)$ exponential
  - because most nodes are near the bottom of tree

Iterative-Deepening Search (IDS)

Howmuch redundant effort is done?

- The number of times the nodes are generated:
  $1b^d + 2b^{(d-1)} + \ldots + db \leq b^d / (1-1/b)^2$
  $= O(b^d)$
  - d: the solution's depth
  - : the branching factor at each non-leaf node
- For example: $b = 4$
  $\mathbf{4^d / (1 - ¼)^2 = 4^d / (.75)^2 = 1.78 \times 4^d}$
  - in the worst case, 78% more nodes are searched (redundant effort) than exist at depth d
  - as increases, this % decreases

Iterative-Deepening Search

- Trades a little time for a huge reduction in space
  - lets you do breadth-first search with (more space efficient) depth-first search

applications, e.g.,games

Anytime algorithm: good for response-time critical

- An "anytime" algorithm is an algorithm that can return a valid solution to a problem even if it's interrupted at any time before it ends. The algorithm is expected to find better and better solutions the more time it keeps running.

Iterative Deepening Search

- It’s a Depth First Search, but it does it one level at a time, gradually increasing the limit, until a goal is found.
- Combine the benefits of depth-first and breadth-first search
- like DFS, modest memory requirements O(bd)
- like BFS, it is complete when branching factor is finite, and optimal when the path cost is a nondecreasing function of the dept of the node

Iterative Deepening Search

- May seem wasteful because states are generated multiple times
- but actually not very costly, because nodes at the bottom level are generated only once
- In practice, however, the overhead of these multiple expansions is small, because most of the nodes are towards leaves (bottom) of the search tree:

  thus, the nodes that are evaluated several times (towards top of tree) are in relatively small number.
- Iterative depending is the preferred uninformed search method when the search space is large and the depth of the solution is unknown

Properties of Iterative Deepening Search

Combines the best of breadth-first and depth-first search strategies.

• Completeness:	Yes,
• Time complexity:	$O(b^{d}$
• Space complexity:	$O(bd)$
• Optimality:	Yes, if step cost = 1

Depth-Limited Search (DLS)

Is a depth-first search with a predetermined depth limit l

Implementation:

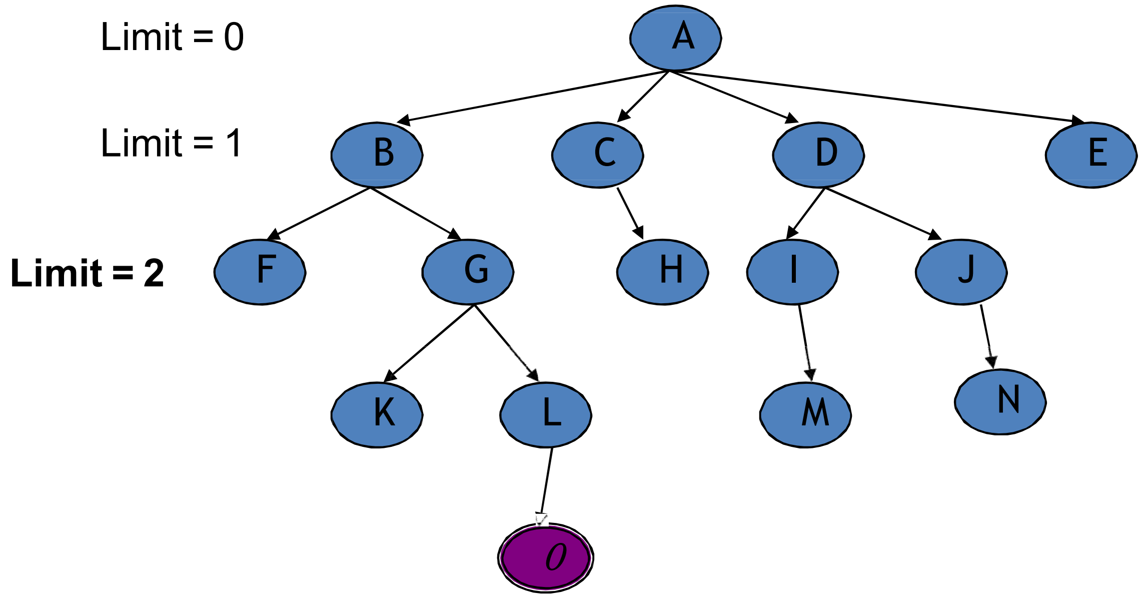
Nodes at depth l are treated as if they have no successors.

Complete: if cutoff chosen appropriately then it is guaranteed to find a solution.

Optimal: it does not guarantee to find the least-cost solution (If choosing $l > d$)

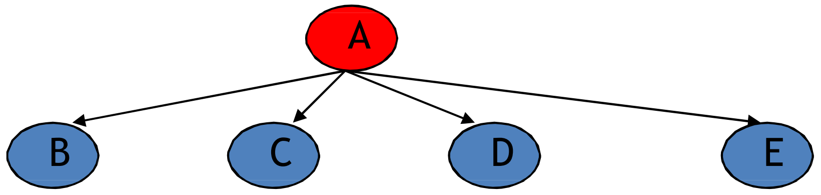
Depth-Limited Search (DLS)

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



Depth-Limited Search (DLS)

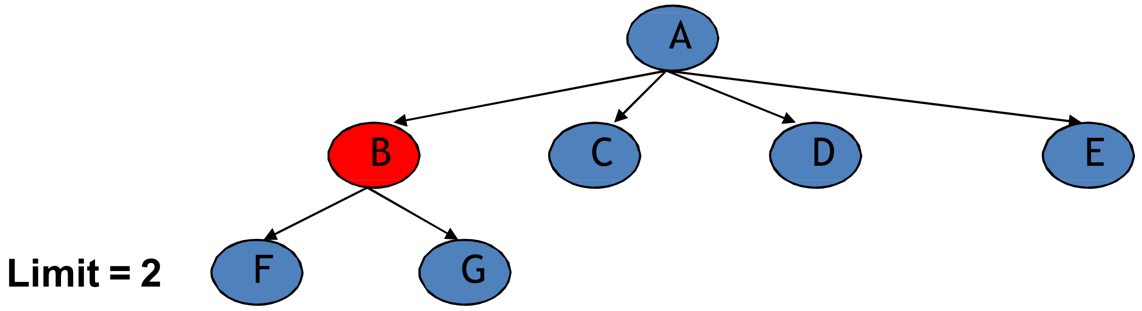
- A,



Limit = 2

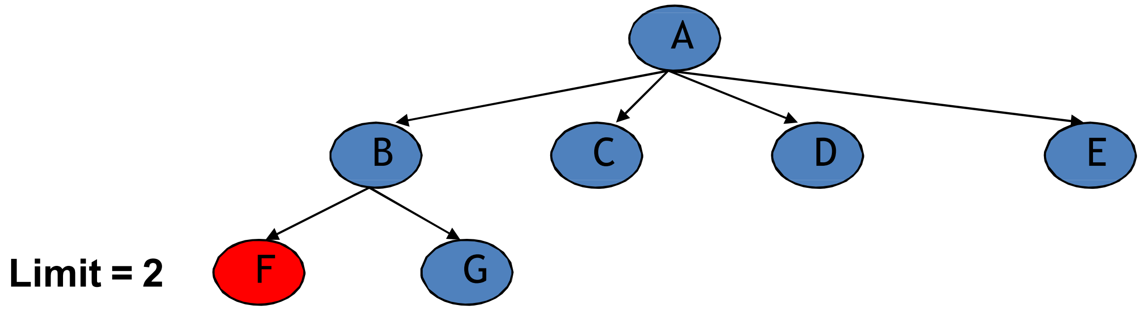
Depth-Limited Search (DLS)

- A,B,



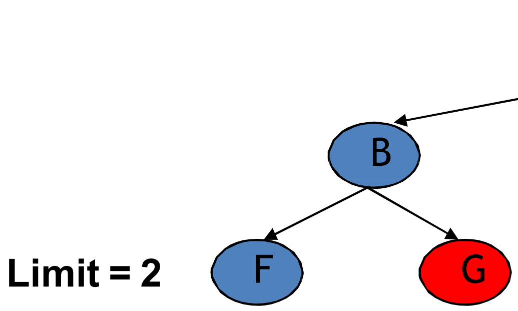
Depth-Limited Search (DLS)

- A,B,F,



Depth-Limited Search (DLS)

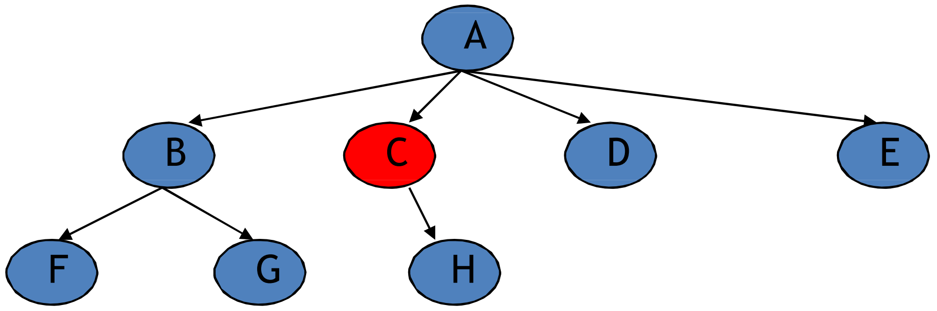
- A,B,F,
- G,



Depth-Limited Search (DLS)

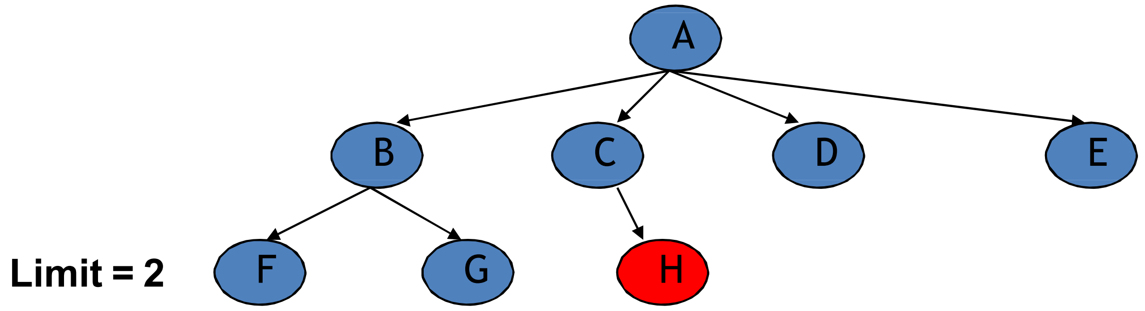
- A,B,F,
- G,
- C,

Limit = 2



Depth-Limited Search (DLS)

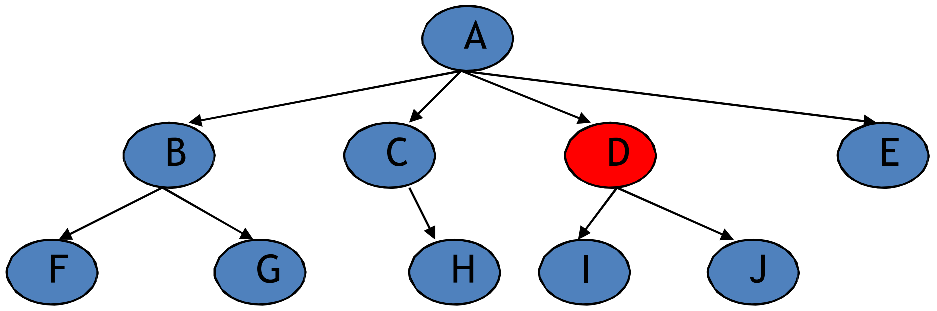
- A,B,F,
- G,
- C,H,



Depth-Limited Search (DLS)

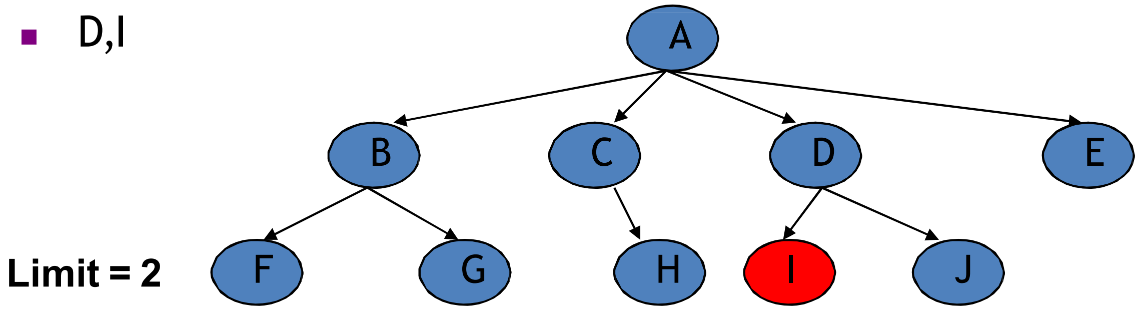
- A,B,F,
- G,
- C,H,
- D,

Limit = 2



Depth-Limited Search (DLS)

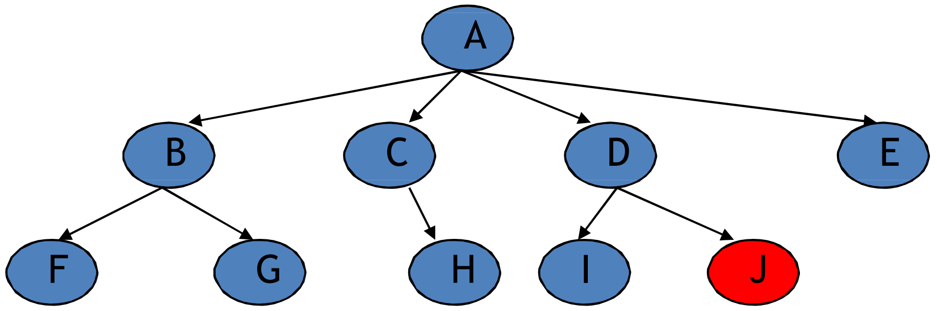
- A,B,F,
- G,
- C,H,
- D,I



Depth-Limited Search (DLS)

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

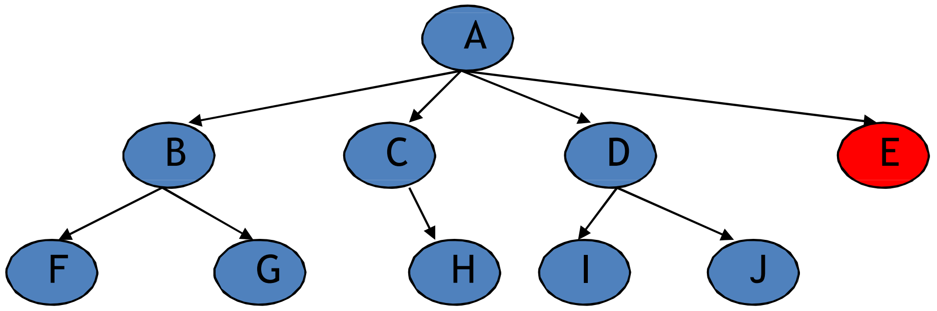
Limit = 2



Depth-Limited Search (DLS)

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

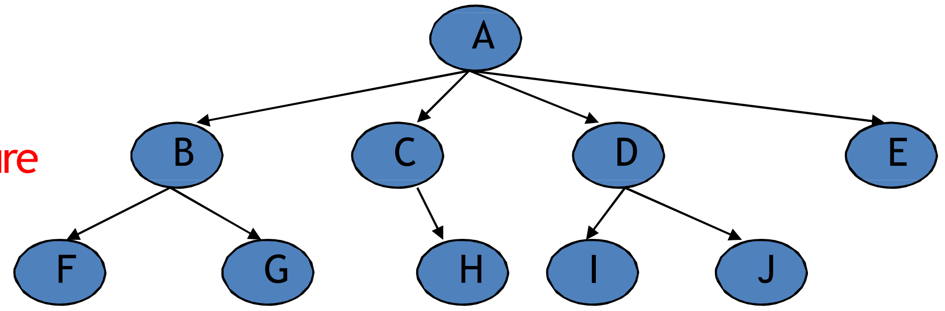
Limit = 2



Depth-Limited Search (DLS)

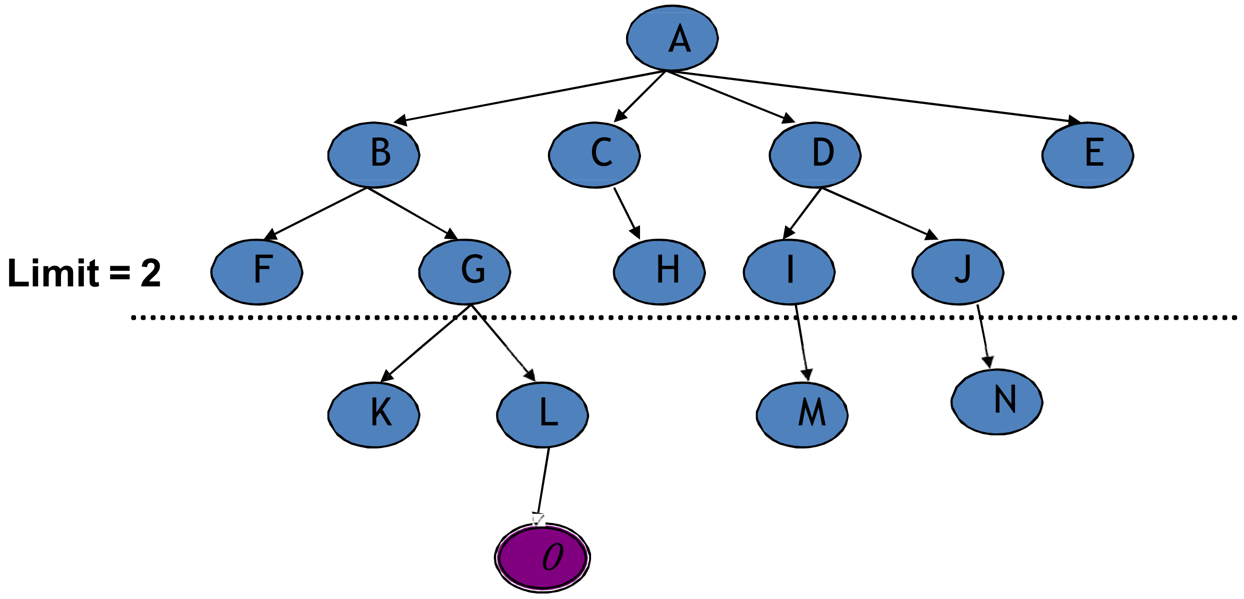
- A,B,F,
- G,
- C,H,
- D,I
- J,
- E, Failure

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)



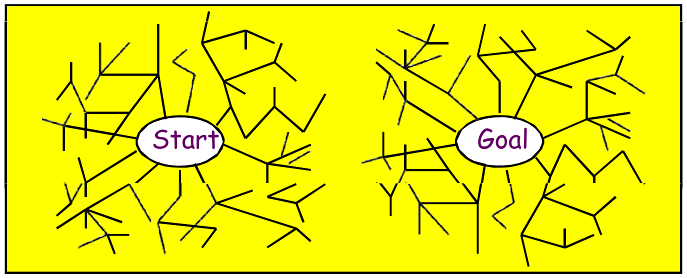
How to choose depth limit l

- E.g. Romania case, 20 cities
  - The longest is 19, so l can be 19
  - From the study, cities can be reached by other cities by 9 steps (diameter)
  - If the diameter is known, more efficient depth-limited search

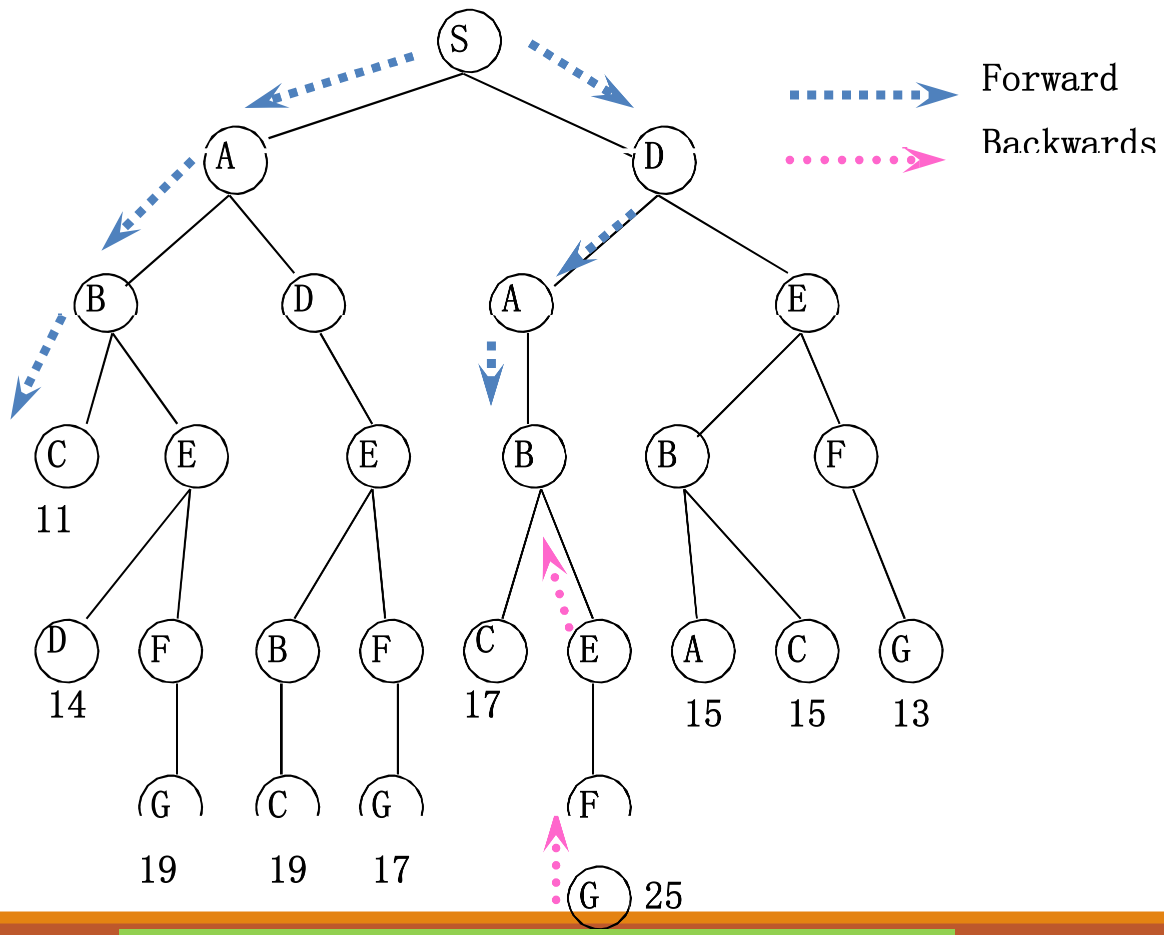
- Depth limit search can be terminated by
  - Failure ⓪ no solution
  - Cutoff ⓪ no solution within the depth limit

Bidirectional Search

- Both search forward from the initial state, and backward from the goal.
- Stop when the two searches meet in the middle.
- Motivation: $b^{d/2} + b^{d/2}$ is much less than b^d
- Implementation
  - Replace the goal test with a check to see whether the frontiers of the two searches intersect, if yes
    ⓪ solution is found



Bidirectional Search



Bidirectional Search

- Check when each node is expanded or selected for expansion
- Can be implemented using BFS or iterative deepening (but at least one frontier needs to be kept in memory)
- Significant weakness
  - Space requirement
- Time Complexity is good

Bidirectional Search

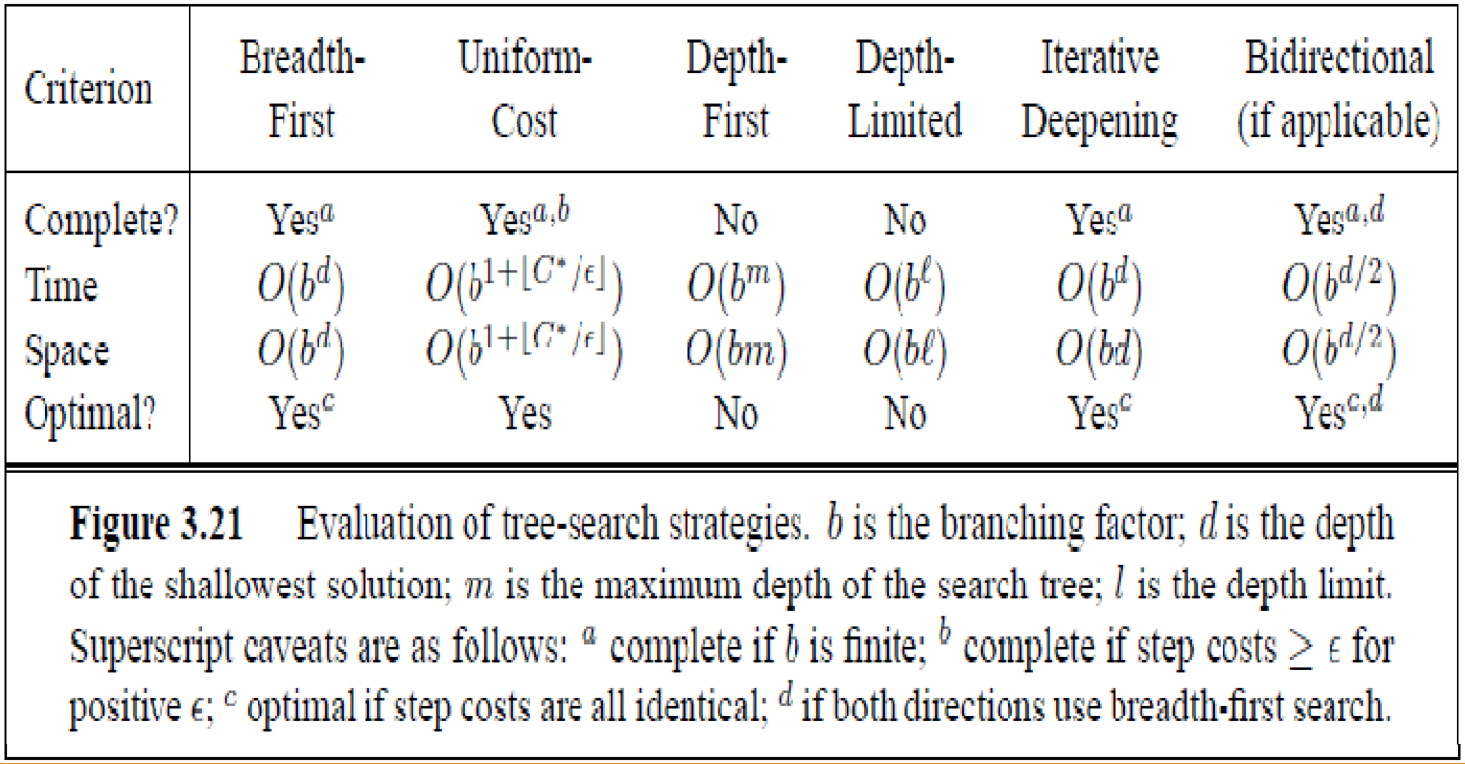
- Problem: how do we search backward from goal??
  - predecessor of node n = all nodes that have n as successor
  - this may not always be easy to compute!
  - if several goal states, apply predecessor function to them just as we applied successor (only works well if goals are explicitly known; may be difficult if goals only characterized implicitly).
  - for bidirectional search to work well, there must be an efficient way to check whether a given node belongs to the other search tree.
  - select a given search algorithm for each half.

Bidirectional Search

- Completeness: Yes,
- Time complexity: $2*O(b^{d/2}) = O(b^{d/2})$
- Space complexity: $O(b^{m/2})$
- Optimality: Yes

- To avoid one by one comparison, we need a hash table of size $O(b^{m/2})$
- *If hash table is used, the cost of comparison is O(1*

Comparison 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.

Conclusions

- Interesting problems can be cast as search problems, but you've got to set up state space
- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored