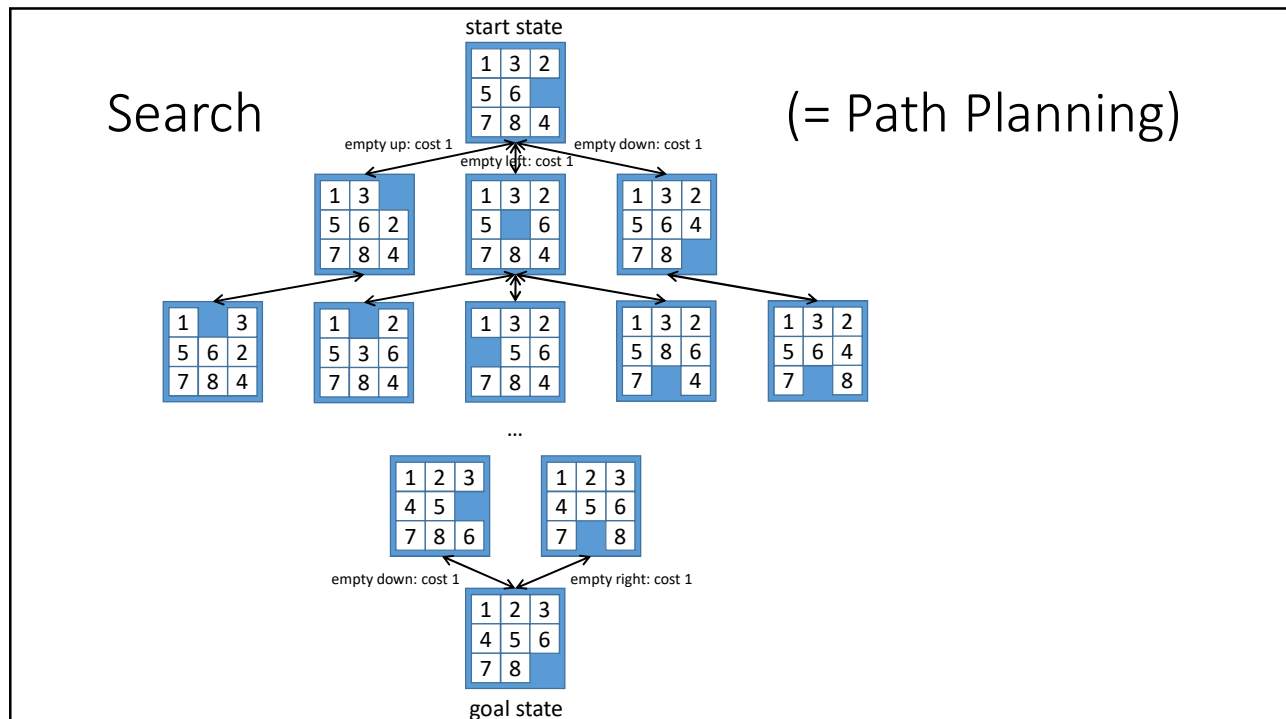


# Uninformed Search

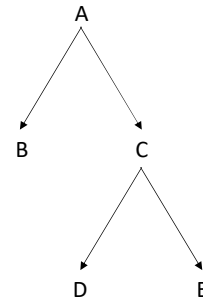
Sven Koenig, USC

Russell and Norvig, 3<sup>rd</sup> Edition, Sections 10.2.1 and 3.3-3.4

These slides are new and can contain mistakes and typos.  
Please report them to Sven (skenig@usc.edu).



## Search



- Terminology for trees

- C is the parent node of D.
- D and E are the child nodes of C.
- A is the root node since it has no parent node.
- B, D and E are the leaf nodes since they have no child nodes.
- All leaf nodes form the fringe (or frontier).
- The depth of node C is one since it is one edge traversal away from the root node.
- The depth of the tree is two since the largest depth of any of its nodes is two.

## Skeleton of Search Algorithms

1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. **Pick an unexpanded fringe node  $n$ .** Let  $s(n)$  be the state it is labeled with.
4. If  $s(n)$  is a goal state, stop successfully and return the path from the root node to  $n$  in the tree (i.e. return the sequence of states that label the nodes on the path).
5. Expand  $n$ , that is, create a child node of  $n$  for each of the successor states of  $s(n)$ , labeled with that successor state.
6. Go to 2.

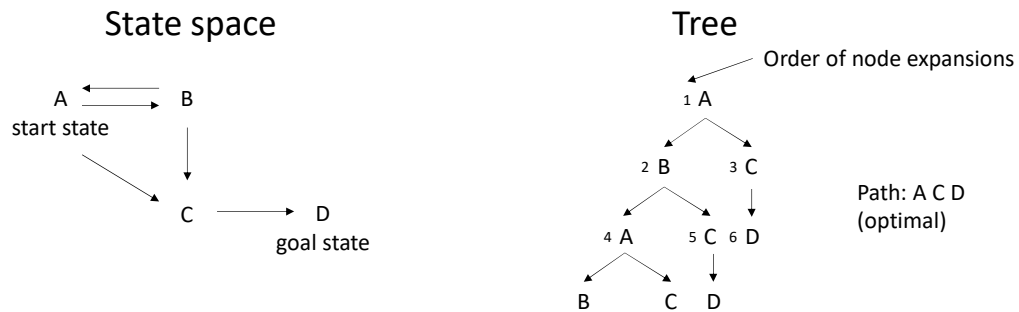
## Skeleton of Search Algorithms

- The search algorithms differ only in how they select the unexpanded fringe node.
  - If no knowledge other than the current tree is available to guide the decision, then a search algorithm is called **uninformed** (or blind).
  - Otherwise, a search algorithm is called **informed**. If the knowledge consists of approximations of the goal distances of the states, the informed search algorithm is called heuristic.

## Breadth-First Search BFS (operator cost = one)

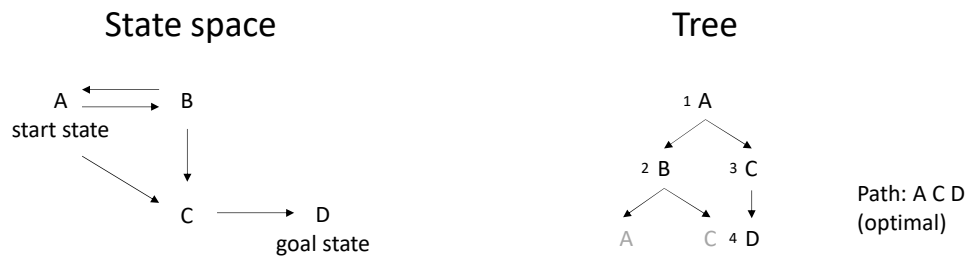
1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. **Pick an unexpanded fringe node  $n$  with the smallest depth.**  
Let  $s(n)$  be the state it is labeled with.
4. If  $s(n)$  is a goal state, stop successfully and return the path from the root node to  $n$  in the tree.
5. Expand  $n$ , that is, generate (= create) a child node of  $n$  for each of the successor states of  $s(n)$ , labeled with that successor state.
6. Go to 2.

## Breadth-First Search BFS (operator cost = one)



We always break ties alphabetically in the following.

## Breadth-First Search BFS (operator cost = one)

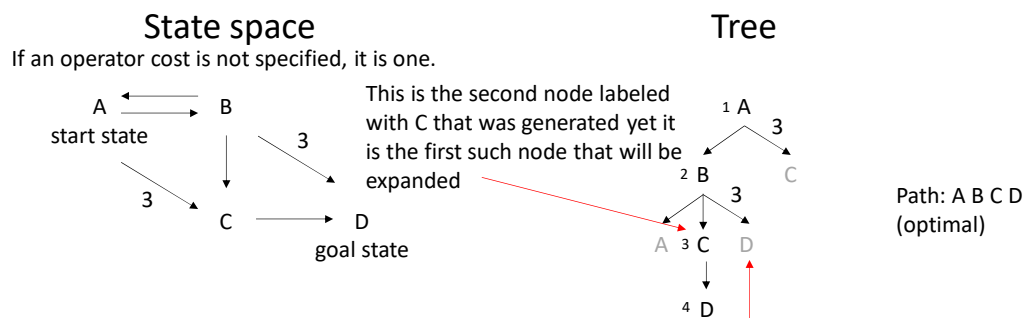


- Pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- Optional termination rule: terminate once a node labeled with a goal state has been generated.

## Uniform-Cost Search (operator cost = positive)

1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node  $n$  with the smallest cost  $g(n)$  from the root to  $n$ . Let  $s(n)$  be the state it is labeled with.
4. If  $s(n)$  is a goal state, stop successfully and return the path from the root node to  $n$  in the tree.
5. Expand  $n$ , that is, generate (= create) a child node of  $n$  for each of the successor states of  $s(n)$ , labeled with that successor state.
6. Go to 2.

## Uniform-Cost Search (operator cost = positive)



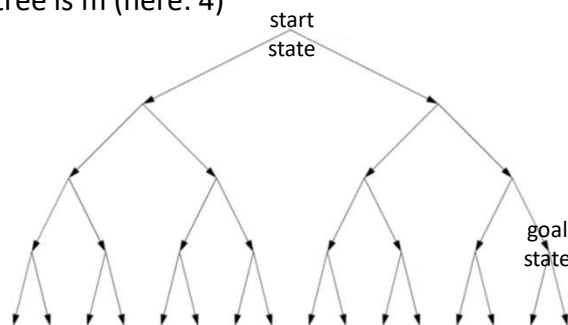
- Pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- Termination rule: terminate once a node labeled with a goal state has been generated

## Properties

- **Completeness**
  - If an operator sequence exists that transforms the start state to a goal state, will the search algorithm report one?
- **Time complexity**
  - How many nodes are expanded at most?
- **Space complexity**
  - How many nodes need to be in memory at most at any point in time?
- **Optimality**
  - If an operator sequence exists that transforms the start state to a goal state, will the search algorithm report a cost-minimal one?

## Properties

- We assume that the state space is a uniform tree when calculating the space and time complexity.
  - Branching factor is  $b$  (here: 2).
  - Depth of the goal state is  $d$  (here: 3).
  - Depth of the tree is  $m$  (here: 4)



## Properties

$$b^0 + b^1 + \dots + b^d = O(b^d)$$

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>				
Time	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$				
Space	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$				
Optimal?	Yes <sup>c</sup>	Yes				

**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: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete if step costs  $\geq \epsilon$  for positive  $\epsilon$ ; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.

## Breadth-First Search BFS (operator cost = one)

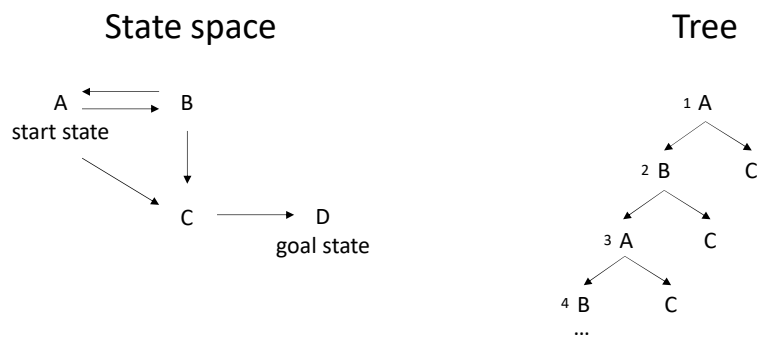
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	$10^6$	1.1 seconds	1 gigabyte
8	$10^8$	2 minutes	103 gigabytes
10	$10^{10}$	3 hours	10 terabytes
12	$10^{12}$	13 days	1 petabyte
14	$10^{14}$	3.5 years	99 petabytes
16	$10^{16}$	350 years	10 exabytes

**Figure 3.13** Time and memory requirements for breadth-first search. The numbers shown assume branching factor  $b = 10$ ; 1 million nodes/second; 1000 bytes/node.

## Depth-First Search DFS

1. Start with a tree that contains only one node, labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node  $n$  with the largest depth. Let  $s(n)$  be the state it is labeled with.
4. If  $s(n)$  is a goal state, stop successfully and return the path from the root node to  $n$  in the tree.
5. Expand  $n$ , that is, generate (= create) a child node of  $n$  for each of the successor states of  $s(n)$ , labeled with that successor state.
6. Go to 2.

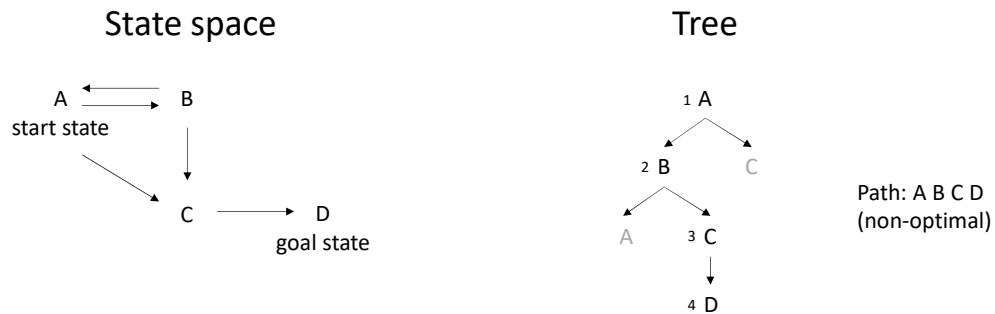
## Depth-First Search DFS (non-working)



Note: In infinite state spaces (= with infinitely many states) DFS can go down a branch of the tree without a cycle, e.g.  $X_1 - X_2 - X_3 - X_4 - X_5 - X_6 - X_7 - \dots$



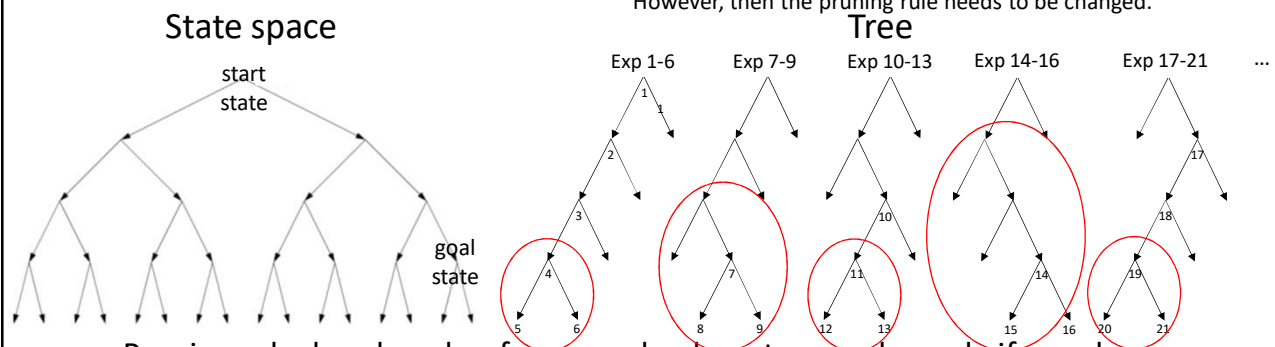
## Depth-First Search DFS (memory inefficient)



- Pruning rule: break cycles, for example, do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- Optional termination rule: terminate once a node labeled with a goal state has been generated.

## Depth-First Search DFS (memory efficient – use it)

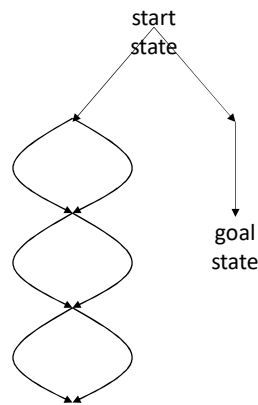
Delete **this** from memory since it is no longer needed.  
However, then the pruning rule needs to be changed.



- Pruning rule: break cycles, for example, do not expand a node if a node labeled with the same state exists from the root node to the node in question.
- Optional termination rule: terminate once a node labeled with a goal state has been generated.

## Depth-First Search DFS (memory efficient – use it)

- The new pruning rule is very weak.



## Properties

Yes, in **finite** state spaces.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup> <b>good</b>	Yes <sup>a,b</sup>	No <b>bad</b>			
Time	$O(b^d)$ <b>good</b>	$O(b^{l+\lceil C^*/\epsilon \rceil})$	$O(b^m)$ <b>bad</b>			
Space	$O(b^d)$ <b>bad</b>	$O(b^{l+\lceil C^*/\epsilon \rceil})$	$O(bm)$ <b>good</b>			
Optimal?	Yes <sup>c</sup> <b>good</b>	Yes	No <b>bad</b>			

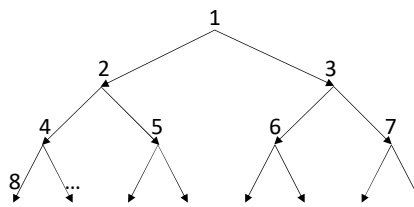
**Figure 3.21** Evaluation of tree-search strategies.  $b$  is the branching factor;  $d$  is the depth of the shallowest solution;  $m$  is the maximum depth of the search tree;  $l$  is the depth limit. Superscript caveats are as follows: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete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

## Iterative Deepening Search (operator cost = one)

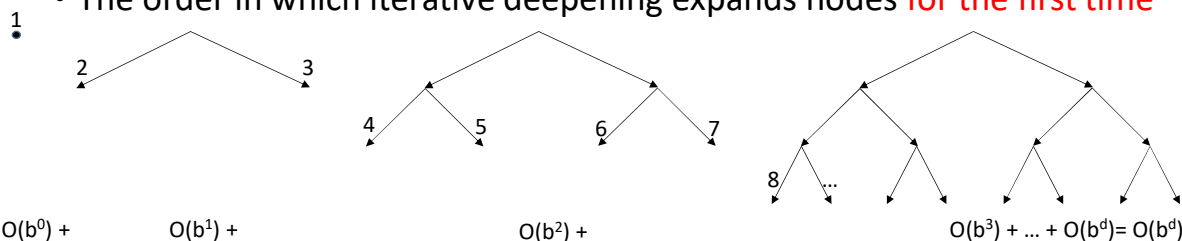
- Combine the best properties of breadth-first and depth-first searches
- Implement a breadth-first search with a series of depth-first searches with increasing depth limits (that is, depth-first searches that assume that nodes whose depths are **at least** the depth limit have no children).
  1.  $l := 0$ .
  2. Perform a depth-first search with depth limit  $l$ .
  3. If a node  $n$  labeled with a goal state was expanded, stop successfully and return the path from the root node to  $n$  in the tree.
  4. If no node at depth  $l$  was expanded, stop unsuccessfully.
  5.  $l := l+1$ .
  6. Go to 2.

## Iterative Deepening Search (operator cost = one)

- The order in which a breadth-first search expands nodes



- The order in which iterative deepening expands nodes **for the first time**



## Properties

If the branching factor is two, then about half of the expanded nodes are expanded for the first time during each depth-first search. This percentage is even larger for larger branching factors.

Yes, in **finite** state spaces.

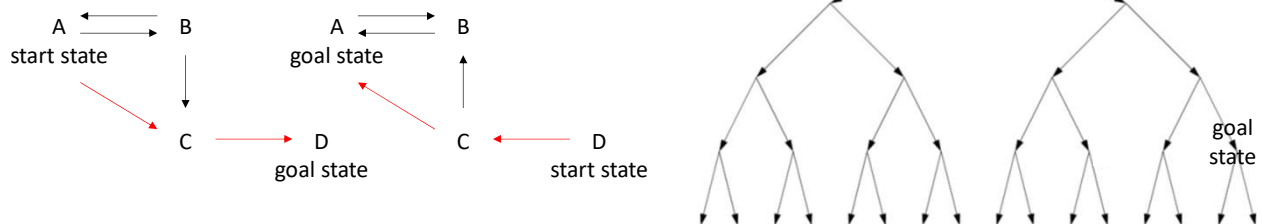
- Iterative deepening incurs a time overhead over breadth-first search since it expands nodes multiple times. But the overhead is small and thus a small price to pay for the small space complexity.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>	No	No	Yes <sup>a</sup>	good
Time	$O(b^d)$	$O(b^{l+⌈C^*/\epsilon⌉})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	good
Space	$O(b^d)$	$O(b^{l+⌈C^*/\epsilon⌉})$	$O(bm)$	$O(bl)$	$O(bd)$	good
Optimal?	Yes <sup>c</sup>	Yes	No	No	Yes <sup>c</sup>	good

**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: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete if step costs  $\geq \epsilon$  for positive  $\epsilon$ ; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.

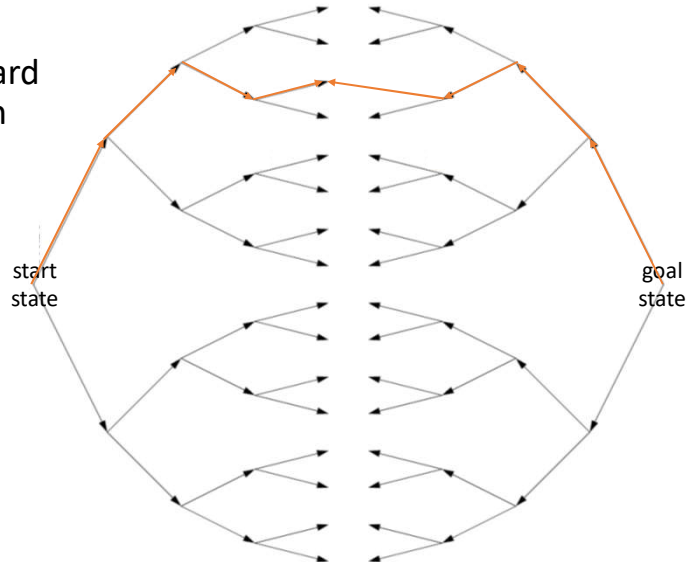
## Bidirectional Search (here: operator cost = one)

- Properties of backward search (from goal state to start state)
  - Can be done in principle by reversing all operators and exchanging the start and goal states, which finds a path in reverse for the original state space. However, this might be difficult to do in practice (e.g. for STRIPS planning).
  - Can be faster or slower than forward search, depending on the branching factors (which can be different)



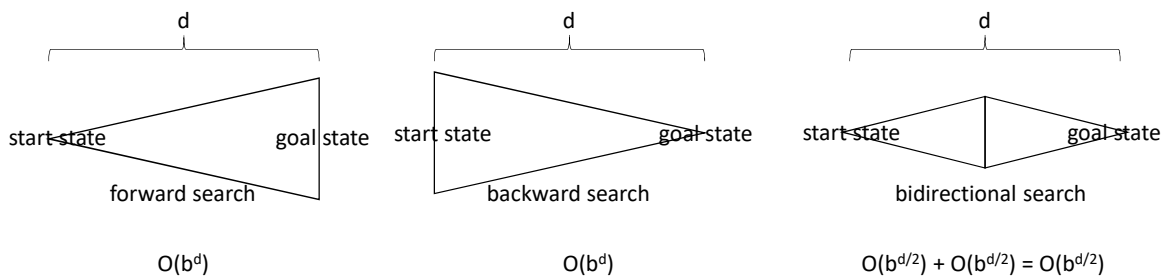
## Bidirectional Search (here: operator cost = one)

- It helps if the forward or backward search is a breadth-first search (since one needs to test for an intersection of the trees)



## Bidirectional Search (here: operator cost = one)

- We assume that the state space is a uniform tree when calculating the space and time complexity.
  - Forward and backward branching factor is  $b$
  - Depth of the goal state is  $d$



## Properties

Yes, in **finite** state spaces.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>	No	No	Yes <sup>a</sup>	Yes <sup>a,d</sup>
Time	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes <sup>c</sup>	Yes	No	No	Yes <sup>c</sup>	Yes <sup>c,d</sup>

**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: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete if step costs  $\geq \epsilon$  for positive  $\epsilon$ ; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.

## Uninformed Search

- Want to play around with uninformed search algorithms?
- Go here: <http://aispace.org/search/>