Breadth-First

- Algorithm outline:
  - Always select from the OPEN the node with the **smallest depth** for expansion, and put all newly generated nodes into OPEN
  - OPEN is organized as **FIFO** (first-in, first-out) list, i.e., a **queue**.
  - Terminate if a node selected for expansion is a goal
- **Properties**
  - **Complete**
  - **Optimal** (i.e., admissible) if all operators have the same cost. Otherwise, not optimal but finds solution with **shortest path length** (shallowest solution).
  - **Exponential time and space complexity**,

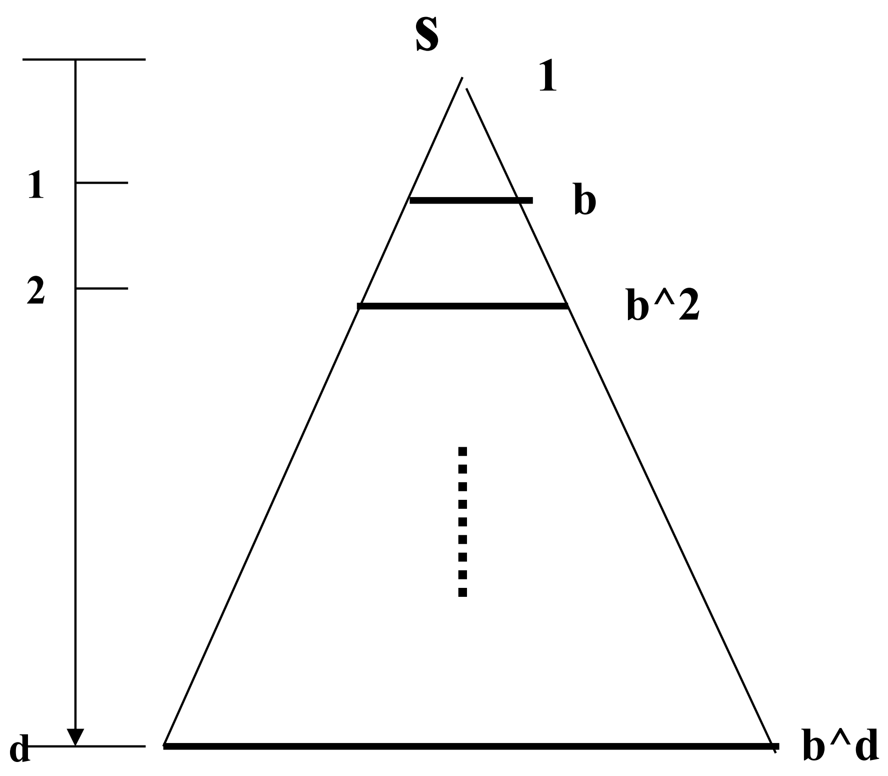
    O(b^d) nodes will be generated, where

    d is the depth of the solution and

    b is the branching factor (i.e., number of children) at each node

Breadth-First

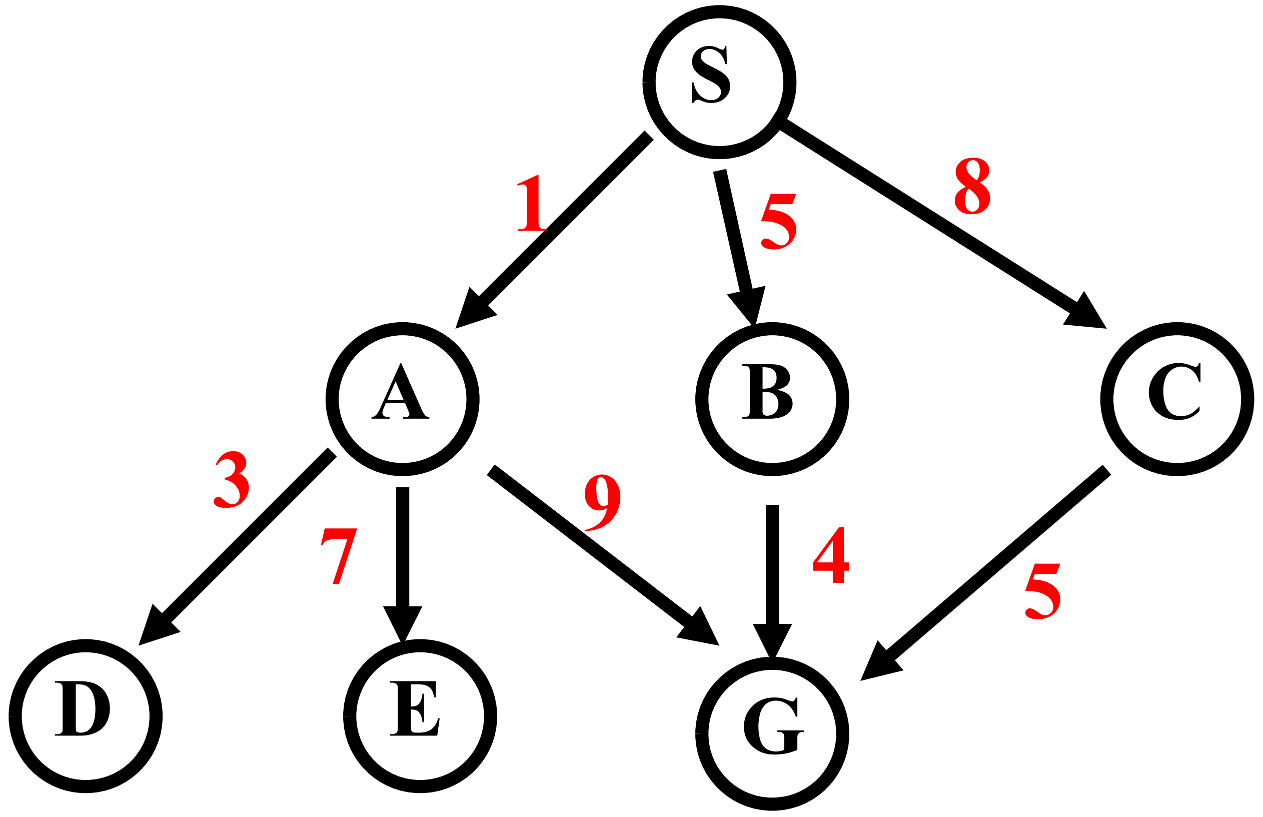
- A complete search tree of depth d where each non-leaf node has b children, has a total of 1 + b + b^2 + ... + b^d = (b^(d+1) - 1)/(b-1) nodes
- Time complexity (# of nodes generated): O(b^d)
- Space complexity (maximum length of OPEN): O(b^d)



- For a complete search tree of depth 12, where every node at depths 0, ..., 11 has 10 children and every node at depth 12 has 0 children, there are 1 + 10 + 100 + 1000 + ... + 10^12 = (10^13 - 1)/9 = O(10^12) nodes in the complete search tree.

• BFS is suitable for problems with shallow solutions

Example Illustrating Uninformed Search Strategies



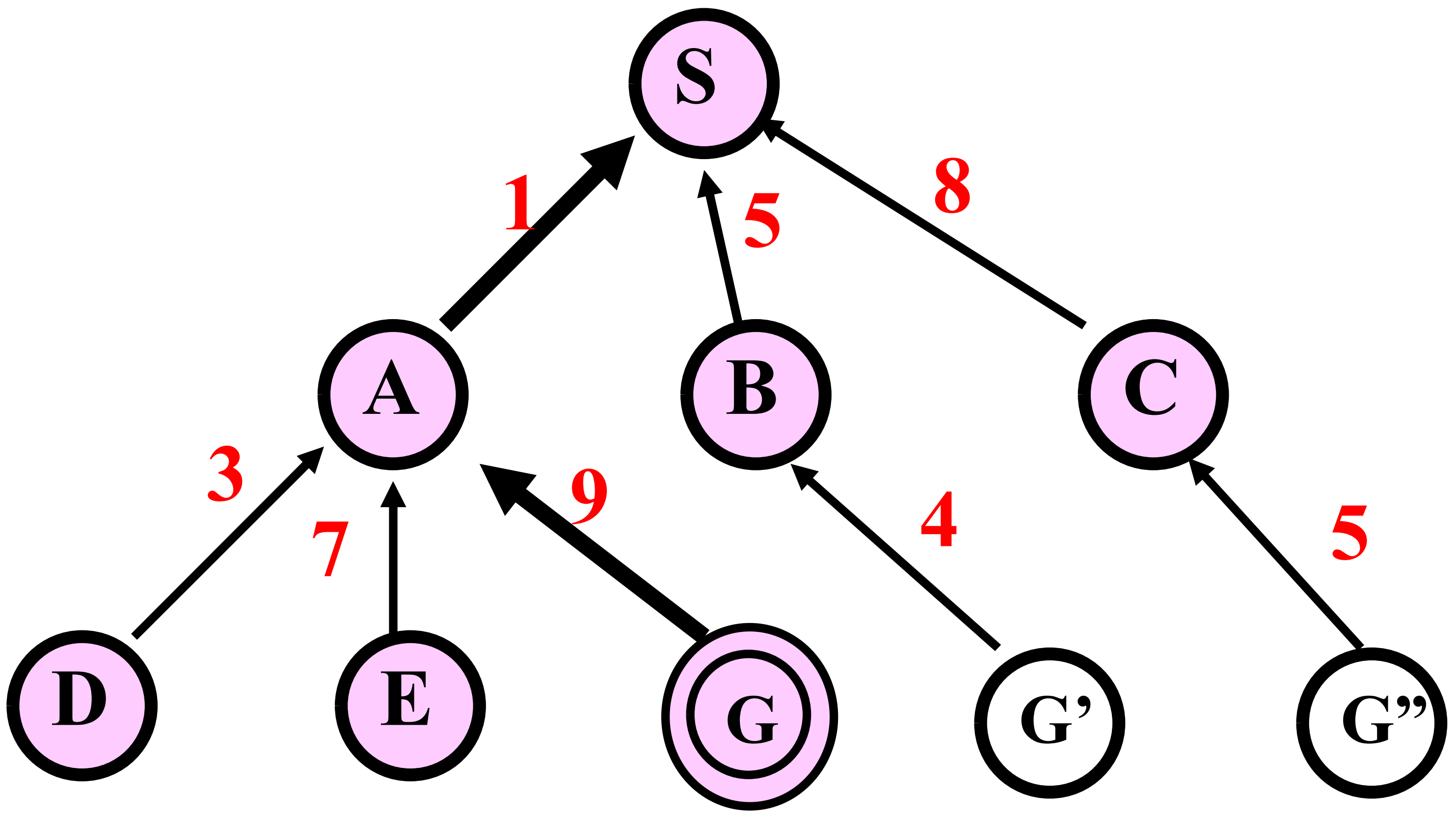
Breadth-First Search

exp. node	OPEN list	CLOSED list
	{ S }	{}
S	{ A B C }	{S}
A	{ B C D E G }	{S A}
B	{ C D E G G' }	{S A B}
C	{ D E G G' G" }	{S A B C}
D	{ E G G' G" }	{S A B C D}
E	{ G G' G" }	{S A B C D E}
G	{ G' G" }	{S A B C D E}

Solution path found is S A G <-- this G also has cost 10

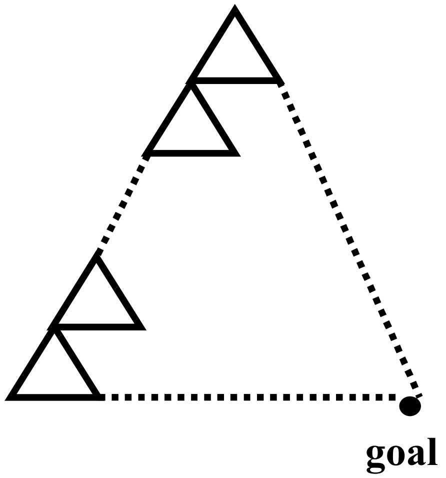
Number of nodes expanded (including goal node) = 7

CLOSED List: the search tree connected by backpointers



Depth-First (DFS)

- Algorithm outline:
  - Always select from the OPEN the node with the **greatest depth** for expansion, and put all newly generated nodes into OPEN
  - OPEN is organized as **LIFO** (last-in, first-out) list.
  - Terminate if a node selected for expansion is a goal



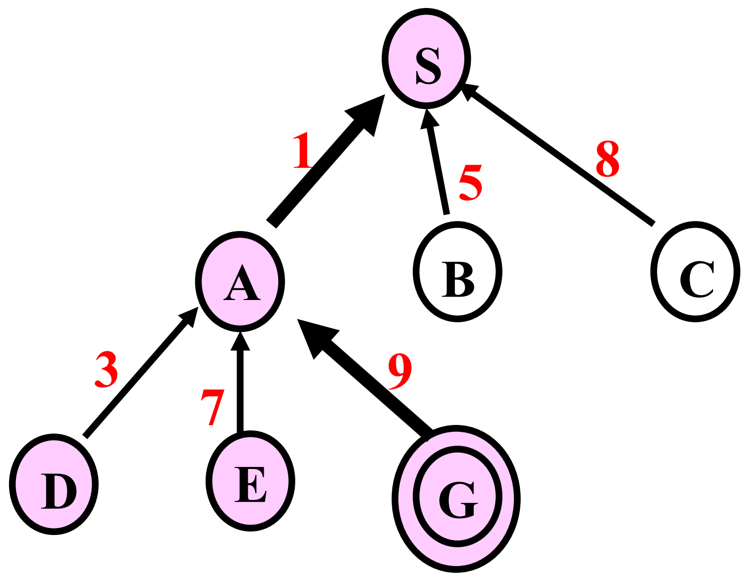
- **May not terminate** without a "depth bound," i.e., cutting off search below a fixed depth D (How to determine the depth bound?)
- **Not complete** (with or without cycle detection, and with or without a cutoff depth)
- **Exponential time**, O(b^d), but only **linear space**, O(bd), required
- Can find deep solutions quickly if lucky
- When search hits a deadend, can only back up one level at a time even if the "problem" occurs because of a bad operator choice near the top of the tree. Hence, only does "chronological backtracking"

Depth-First Search

return GENERAL-SEARCH(problem, ENQUEUE-AT-FRONT)

exp. node	**OPEN list**
	{ S }
S	{ A B C }
A	{ D E G B C}
D	{ E G B C }
E	{ G B C }
G	{ B C }

CLOSED list



Solution path found is S A G <-- this G has cost 10

Number of nodes expanded (including goal node) = 5

Uniform-Cost (UCS)

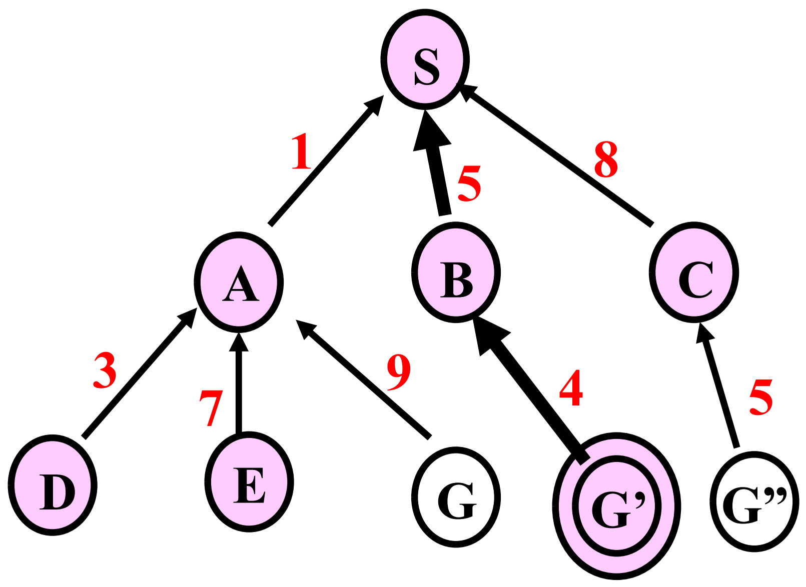
- Let g(n) = cost of the path from the start node to an open node n
- Algorithm outline:
  - Always select from the OPEN the node with the **least g(.) value** for expansion, and put all newly generated nodes into OPEN
  - Nodes in OPEN are sorted by their g(.) values (in ascending order)
  - Terminate if a node selected for expansion is a goal
- Called *“Dijkstra's Algorithm”* in the algorithms literature and similar to *“Branch and Bound Algorithm”* in operations research literature

Uniform-Cost Search

GENERAL-SEARCH(problem, ENQUEUE-BY-PATH-COST)

exp. node	**nodes list**
	{S(0)}
S	{A(1) B(5) C(8)}
A	{D(4) B(5) C(8) E(8) G(10)}
D	{B(5) C(8) E(8) G(10)}
B	{C(8) E(8) G'(9) G(10)}
C	{E(8) G'(9) G(10) G''(13)}
E	{G'(9) G(10) G''(13) }
G'	{G(10) G''(13) }

CLOSED list



Solution path found is S B G <-- this G has cost 9, not 10

Number of nodes expanded (including goal node) = 7

Uniform-Cost (UCS)

- **Complete** (if cost of each action is not infinitesimal)
  - The total # of nodes n with g(n) <= g(goal) in the state space is finite
  - If n’ is a child of n, then g(n’) = g(n) + c(n, n’) > g(n)
  - Goal node will eventually be generated (put in OPEN) and selected for expansion (and passes the goal test)
- **Optimal/Admissible**
  - Admissibility depends on the goal test being applied when a node is removed from the OPEN list, not when it's parent node is expanded and the node is first generated (delayed goal testing)
  - Multiple solution paths (following different backpointers)
  - Each solution path that can be generated from an open node n will have its path cost >= g(n)
  - When the first goal node is selected for expansion (and passes the goal test), its path cost is less than or equal to g(n) of every OPEN node n (and solutions entailed by n)
- **Exponential time and space complexity**,
  - worst case: becomes BFS when all arcs cost the same

Depth-First Iterative Deepening (DFID)

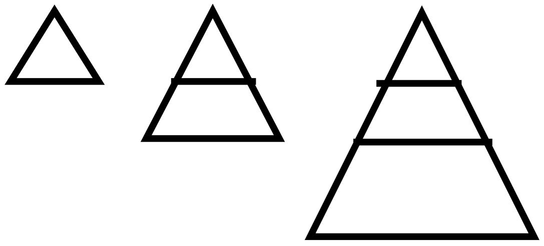
- BF and DF both have exponential time complexity O(b^d)

  BF is **complete** but has exponential space complexity (**conservative**)

  DF has **linear space complexity** but is incomplete (**radical**)
- Space is often a **harder** resource constraint than time
- Can we have an algorithm that
  - Is complete
  - Has linear space complexity, and
  - Has time complexity of O(b^d)
- DFID by Korf in 1985 (17 years after A*)

  First do DFS to depth 0 (i.e., treat start node as having no successors), then, if no solution found, do DFS to depth 1, etc.

```
until solution found do
    DFS with depth bound c
    c = c+1
```



Depth-First Iterative Deepening (DFID)

- **Complete** (iteratively generate all nodes up to depth d)
- **Optimal/Admissible** if all operators have the same cost. Otherwise, not optimal but does guarantee finding solution of shortest length (like BF).
- **Linear space complexity:** O(bd), (like DF)
- **Time complexity** is a little worse than BFS or DFS because nodes near the top of the search tree are generated multiple times, but because almost all of the nodes are near the bottom of a tree, the worst case time complexity is still exponential, O(b^d)

Depth-First Iterative Deepening

- If branching factor is b and solution is at depth d, then nodes at depth d are generated once, nodes at depth d-1 are generated twice, etc., and node at depth 1 is generated d times.

  Hence

  total(d) = b^d + 2b^(d-1) + ... + db

  <= b^d / (1 - 1/b)^2 = O(b^d).

  – If b=4, then worst case is 1.78 * 4^d, i.e., 78% more nodes searched than exist at depth d (in the worst case).

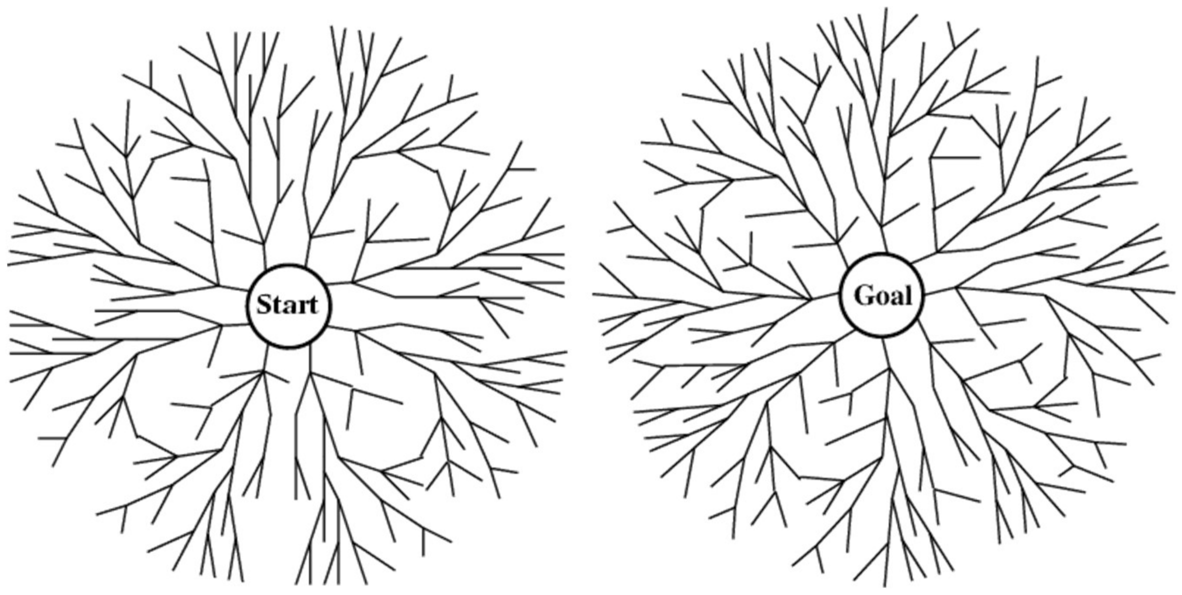
$$
\begin{aligned}
tota(d) &= 1\cdot b^{d} + 2\cdot b^{d-1} + \cdots + (d-1)\cdot b^{2} + d\cdot b \\
&= b^{d}(1 + 2\cdot b^{-1} + \cdots + (d-1)\cdot b^{2-d} + d\cdot b^{1-d})
\end{aligned}
$$

Let $x = b^{-1}$, then

$$
\begin{aligned}
tota(d) &= b^{d}(1 + 2\cdot x^{1} + \cdots + (d-1)\cdot x^{d-2} + d\cdot x^{d-1}) \\
&= b^{d}\frac{d}{dx}(x + x^{2} + \cdots + x^{d-1} + x^{d}) \\
&= b^{d}\frac{d}{dx}\frac{(x - x^{d+1})}{1-x} \\
&\le b^{d}\frac{d}{dx}\frac{x}{1-x} \qquad /{*}x^{d+1} << 1 \text{ when } d \text{ is large since } 1/b < 1{*}/ \\
&= b^{d}\frac{1\cdot(1-x) - x\cdot(-1)}{(1-x)^{2}}. \text{ Therefore}
\end{aligned}
$$

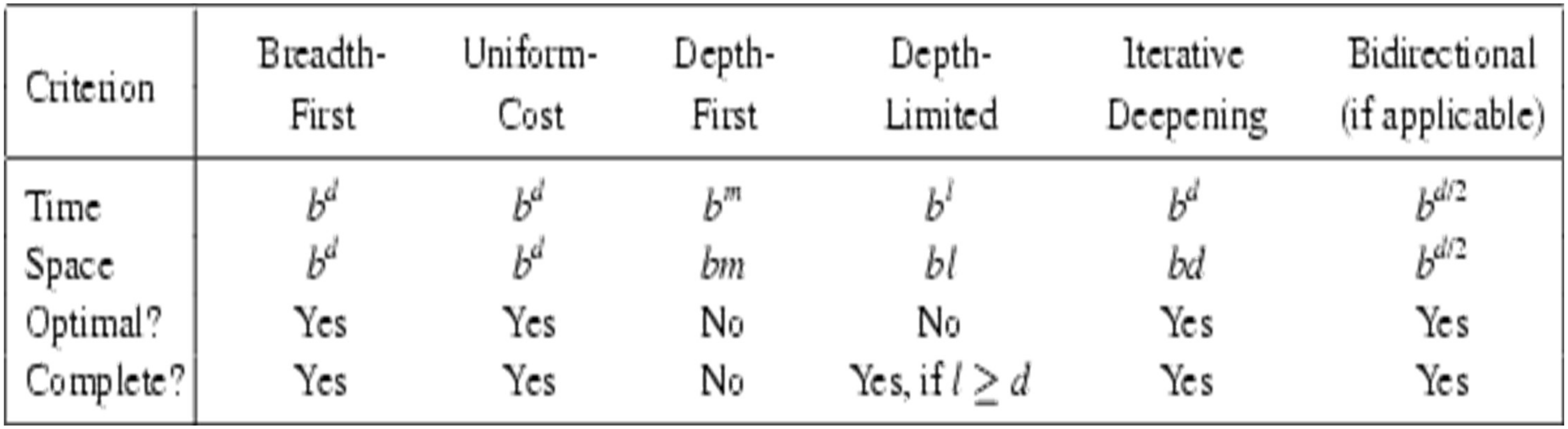
$$
tota(d) \le b^{d}/(1-x)^{2} = b^{d}/(1-b^{-1})^{2}
$$

Bi-directional search



- Alternate searching from the start state toward the goal and from the goal state toward the start.
- Stop when the frontiers intersect.
- Works well only when there are unique start and goal states and when actions are reversible
- Can lead to finding a solution more quickly (but watch out for pathological situations).

Comparing Search Strategies



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

When to use what

- **Depth-First Search:**
  - Many solutions exist
  - Know (or have a good estimate of) the depth of solution
- **Breadth-First Search**:
  - Some solutions are known to be shallow
- **Uniform-Cost Search**:
  - Actions have varying costs
  - Least cost solution is required

  This is the only uninformed search that worries about costs.
- **Iterative-Deepening Search**:
  - Space is limited and the shortest solution path is required