CPSC 322: Introduction to Artificial Intelligence

Uninformed Search

Textbook reference: [3.5]

Instructor: Varada Kolhatkar
University of British Columbia

Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Announcements

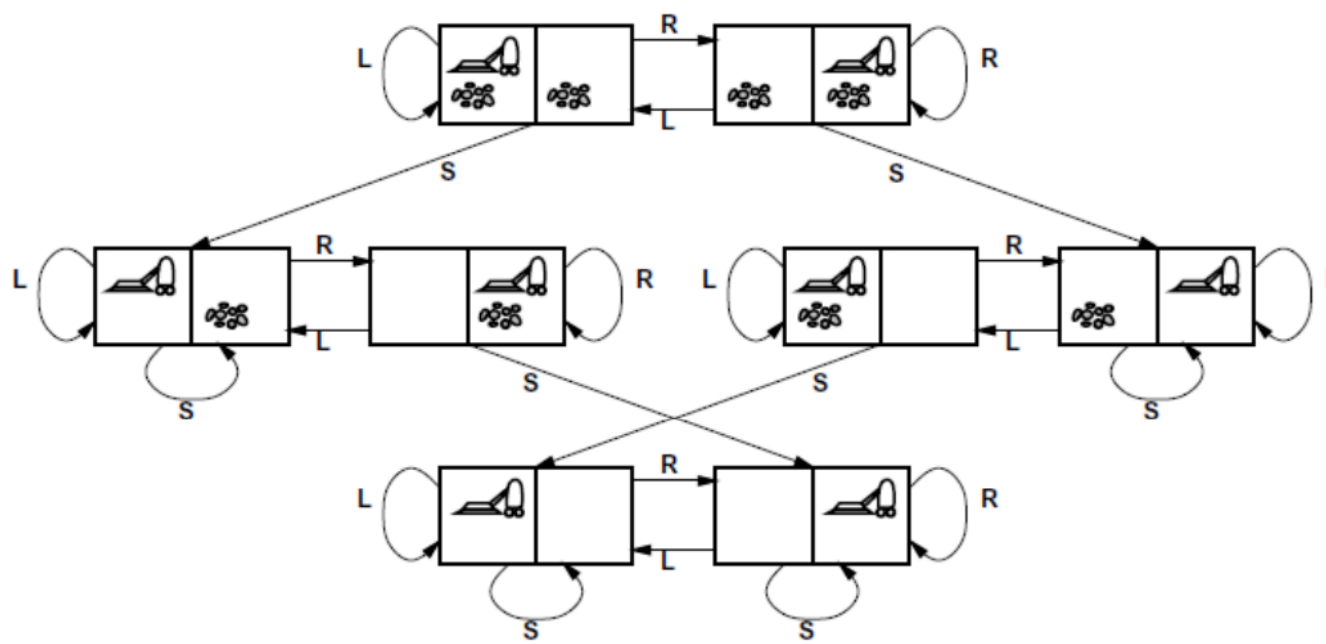
- Please take the midterm timing survey on Canvas.
- Assignment 1 has been released.
    - We have also released the .docx version so that it's easier for editing.
- We expect to return assignment0 grades by the end of this week.

Lecture outline

- **Recap from last lecture** (~10 mins)
- Criteria to compare search strategies (~10 mins)
- Depth-first search (~15 mins)
- Breadth-first search (~15 mins)
- Break (~5 mins)
- Class activity (~15 mins)
- Summary and wrap-up (~5 mins)

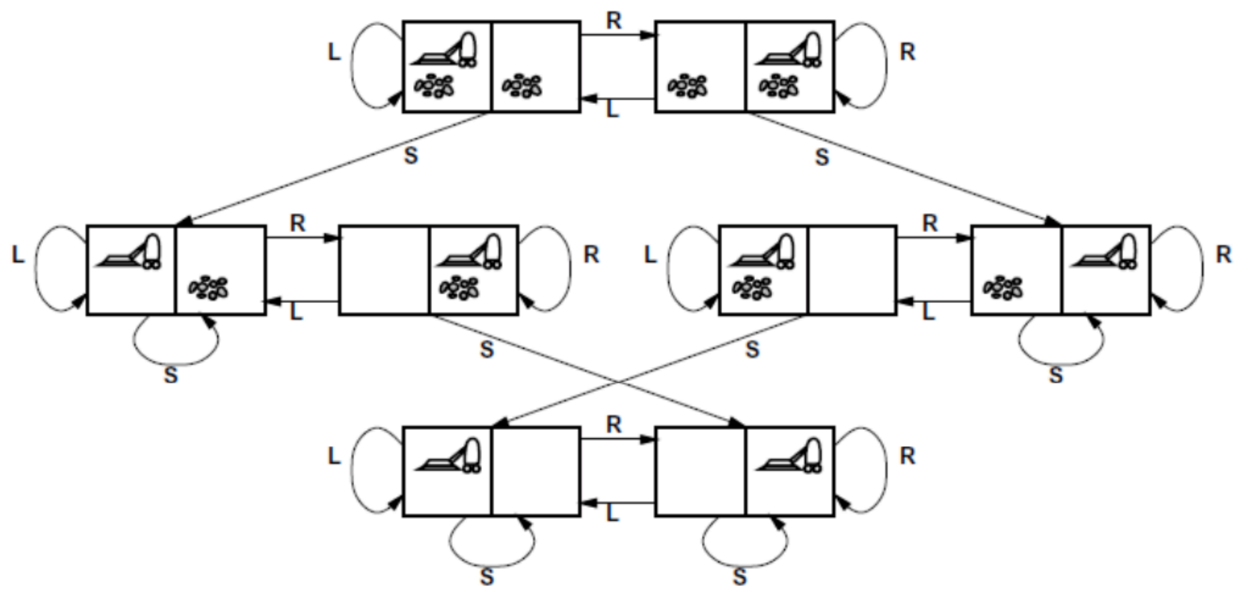
State space graphs

- We formulated problems using
  - States
  - Initial state
  - Actions (operators)
  - Transition model
  - Goal state



State space graphs

- State space graph is a mathematical representation of a search problem
  - **Nodes** are abstracted world configurations
  - **Arcs** represent action results
  - A **goal** is a set of goal nodes.
- In a search graph, each state occurs only once!



We can rarely build this graph in memory because its too big. But it's a useful idea.

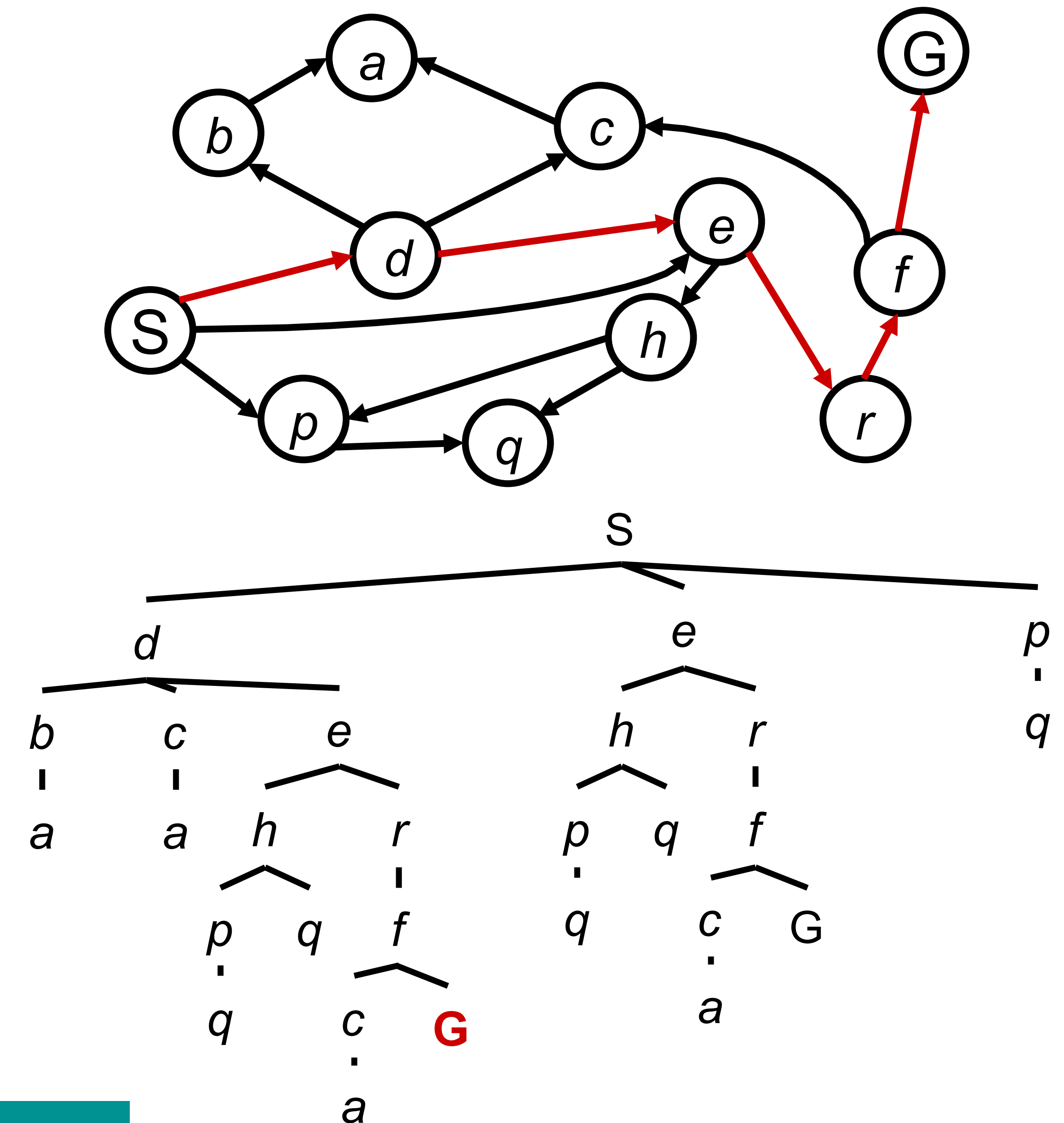
Credit: Berkeley AI course material

Search trees

- Once the problem is formulated, we need to solve it.
- A **solution** is an action sequence.
- We find solutions using search algorithms, which consider various possible action sequences with the help of **search trees**.
- The possible sequences starting at the initial state form a search tree with the **initial state at the root**; the **branches are actions**, and the **nodes correspond to states in the state space** of the problem.

Search trees

- Search tree is a "what if" tree of plans and their outcomes.
- Nodes show states, but correspond to **plans** that achieve those states; each node in the search tree is an entire path in the state space graph.

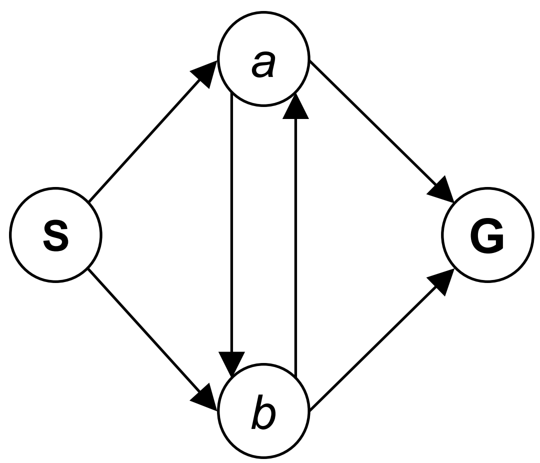


For most problems, we can never actually build the whole tree.

Credit: Berkeley AI course material

State space graphs vs. search trees

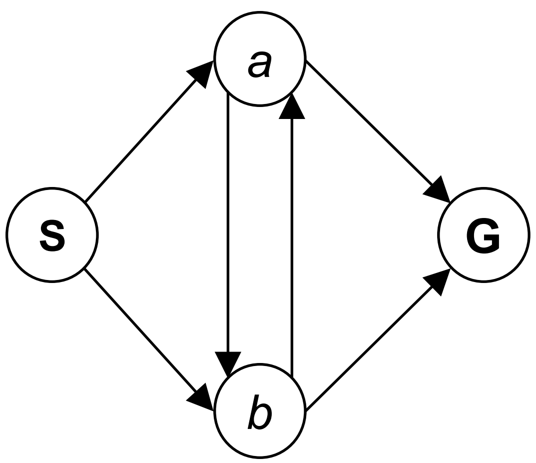
Consider this state graph



How big is the search tree?

State space graphs vs. search trees

Consider this state graph



How big is the search tree?

$$\infty$$

Lots of repeated structure in the search tree.

Activity: Find three bugs in the generic search algorithm below

Inputs: a graph,

a start node n_o

a boolean procedure *goal(n)* that tests if n is a goal node

frontier:= [$< g >$: g is a start node];

While frontier is not empty:

select and **remove** path $< n_0, n_1, \ldots, n_k >$ from frontier;

If *goal*(n_k)

return $< n_0, n_1, \ldots, n_k >$;

Find a neighbour n of n_k

add $< n >$ to frontier;

return NULL

Activity: Find three bugs in the generic search algorithm below

Inputs: a graph,

a start node n_o

a boolean procedure *goal(n)* that tests if n is a goal node

frontier:= [$<g>$: g is a start node];

While frontier is not empty:

select and **remove** path $< n_0, n_1, \ldots, n_k >$ from frontier;

If *goal*(n_k)

return $< n_0, n_1, \ldots, n_k >$;

Find a neighbour n of n_k

add $< n >$ to frontier;

return NULL

Generic search algorithm

Inputs: a graph,

a start node n_o

a boolean procedure *goal(n)* that tests if n is a goal node

frontier:= [$< n_0 >$: n_0 is a start node];

While frontier is not empty:

select and **remove** path $< n_0, n_1, \ldots, n_k >$ from frontier;

If *goal(n_k)*

return $< n_0, n_1, \ldots, n_k >$;

For every neighbour n of n_k

add $< n_0, n_1, \ldots, n_k, n >$ to frontier;

return NULL

Learning outcomes

- Define basic properties of search algorithms:
  - Completeness, optimality, time and space complexity
- Select the most appropriate search algorithms for specific problems.
- BFS vs. DFS vs. IDS
- LCFS vs. BestFS
- A* vs. B&B vs. IDA* vs. MBA*

Lecture outline

- Recap from last lecture (~10 mins)
- **Criteria to compare search strategies** (~10 mins) 👈
- Depth-first search (~15 mins)
- Breadth-first search (~15 mins)
- Break (~5 mins)
- Class activity (~15 mins)
- Summary and wrap-up (~5 mins)

Measuring problem-solving performance

We can evaluate search algorithm's performance in four ways:

- Completeness
- Optimality
- Time complexity
- Space complexity

Completeness and optimality

A search algorithm is **complete** if, whenever at least one solution exists, the algorithm is guaranteed to find a solution within a finite amount of time.

A search algorithm is **optimal** if, when it returns a solution, it is the best solution (i.e., there is no better solution).

- Optimality can be in terms of path costs, for example.

Time and space complexity

The **time complexity** of a search algorithm is an expression for the worst-case amount of time it will take to run expressed in terms of the **maximum path length** m and the **maximum branching factor** b.

The **space complexity** of a search algorithm is an expression for the worst-case amount of memory that the algorithm will use (number of paths). Also expressed in terms of the **maximum path length** m and the **maximum branching factor** b.

Why not in terms of $|V|$ and $|E|$?

Lecture outline

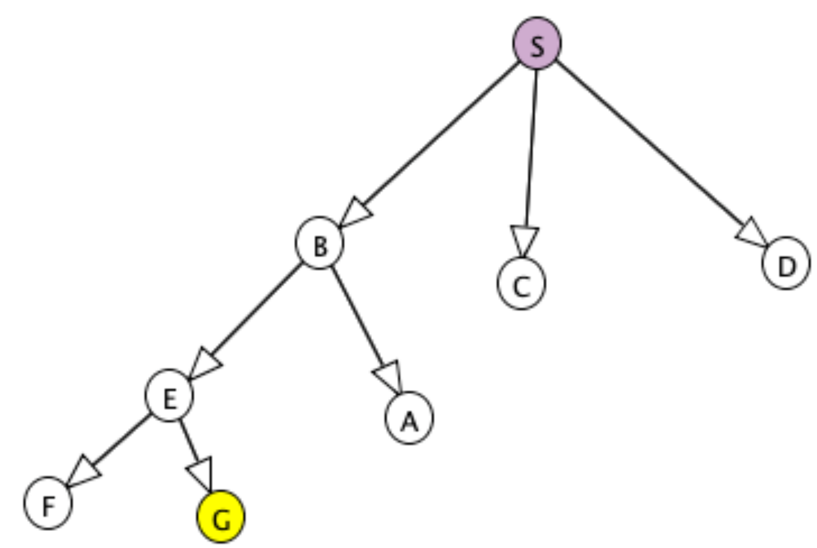
- Recap from last lecture (~10 mins)
- Criteria to compare search strategies (~10 mins)
- **Depth-first search** (~15 mins) 👈
- Breadth-first search (~15 mins)
- Break (~5 mins)
- Class activity (~15 mins)
- Summary and wrap-up (~5 mins)

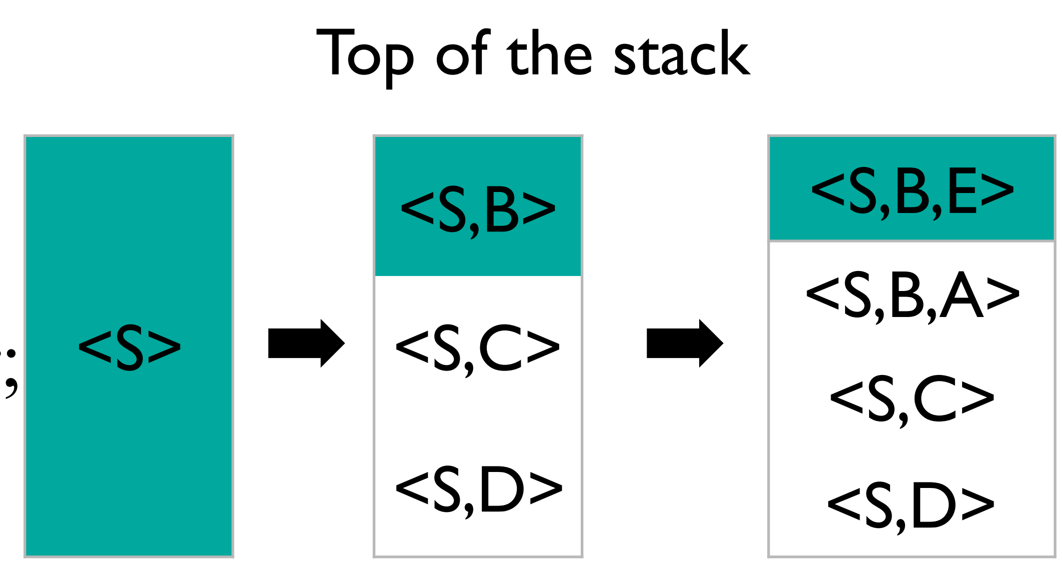
Depth-first search (DFS)

In DFS the **frontier** is a **last-in-first-out stack**.

Inputs: a graph,
a start node n_o
a boolean procedure *goal(n)* that tests if n is a goal node

frontier:= [$< n_0 >$: n_0 is a start node];
While frontier is not empty:
select and **remove** path $< n_0, n_1, \ldots, n_k >$ from frontier;
If $goal(n_k)$
return $< n_0, n_1, \ldots, n_k >$;
For every neighbour n of n_k
add $< n_0, n_1, \ldots, n_k, n >$ to frontier;
return NULL

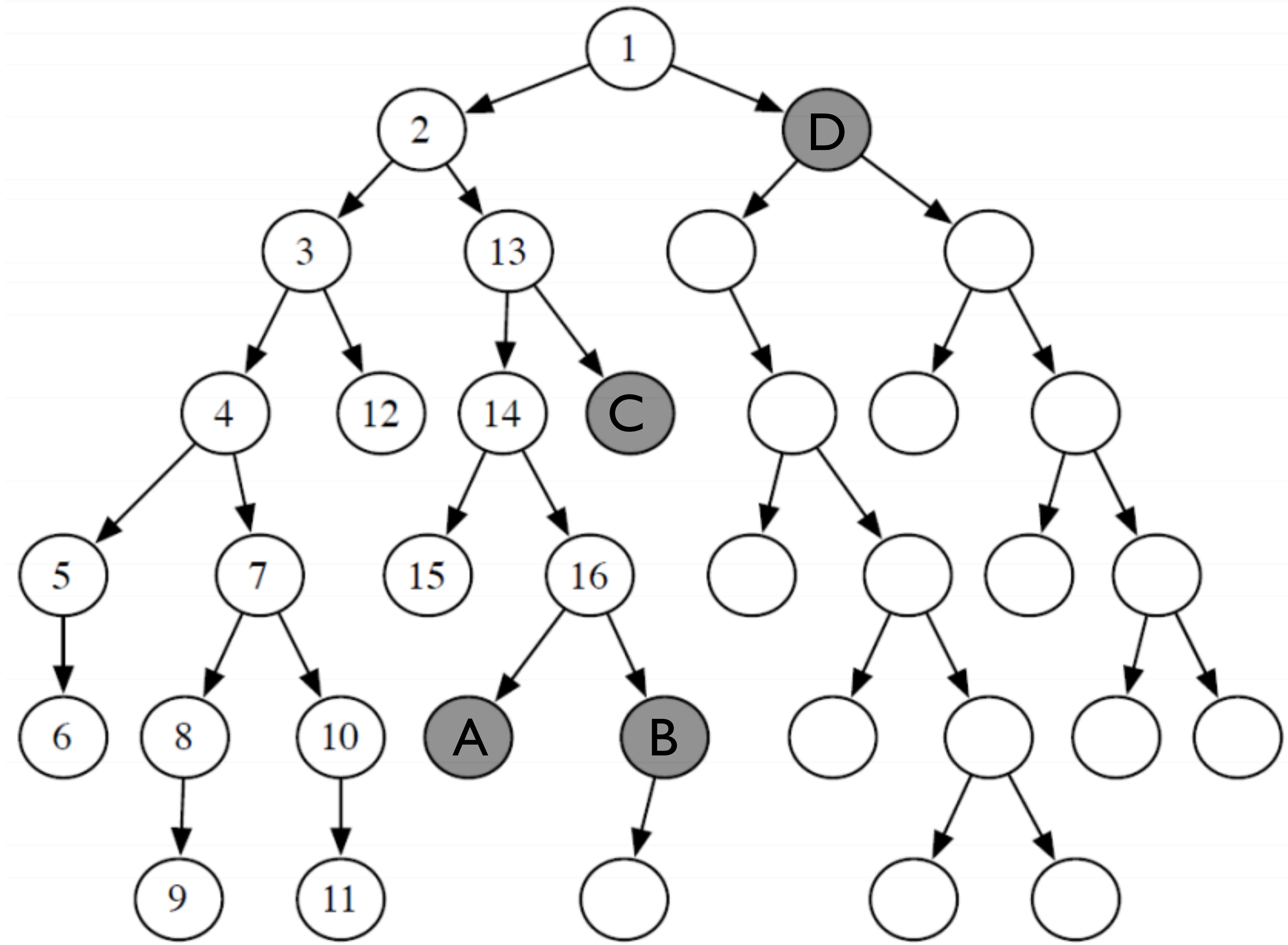




Depth-first search (DFS)

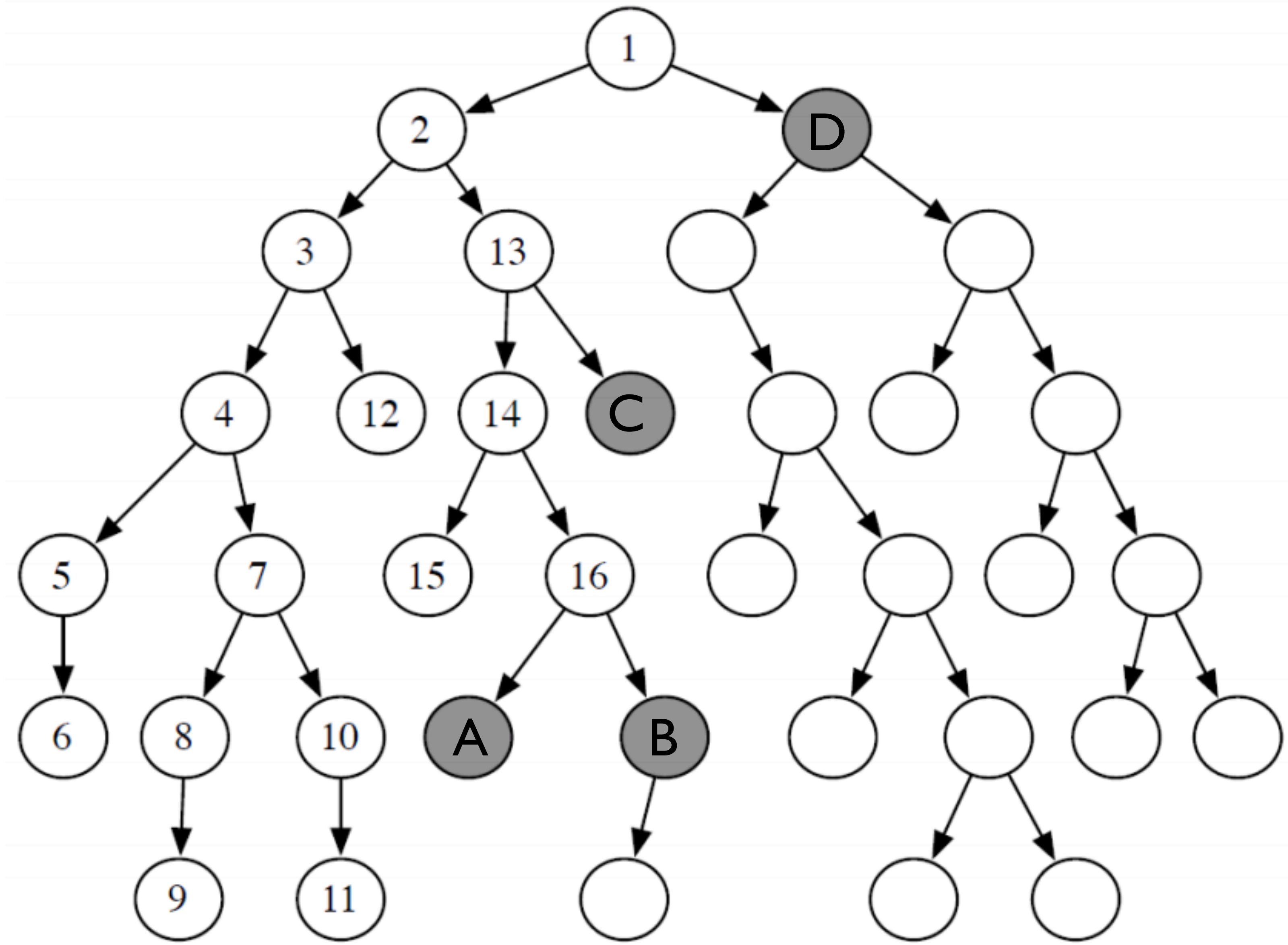
- Expands the deepest node in the current frontier of the search tree

- What's frontier here?



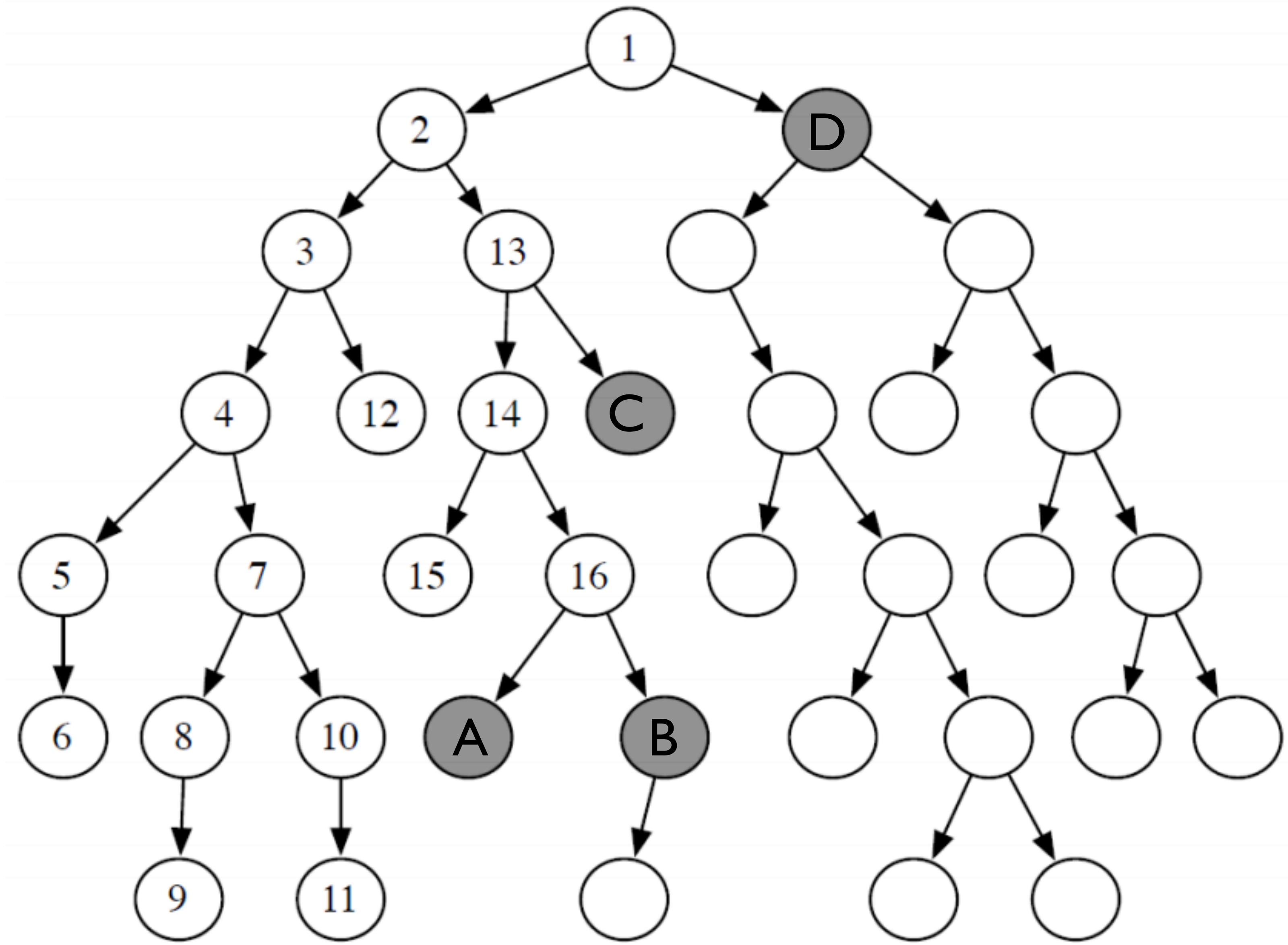
Depth-first search (DFS)

- Frontier?
  - Paths representing the shaded nodes



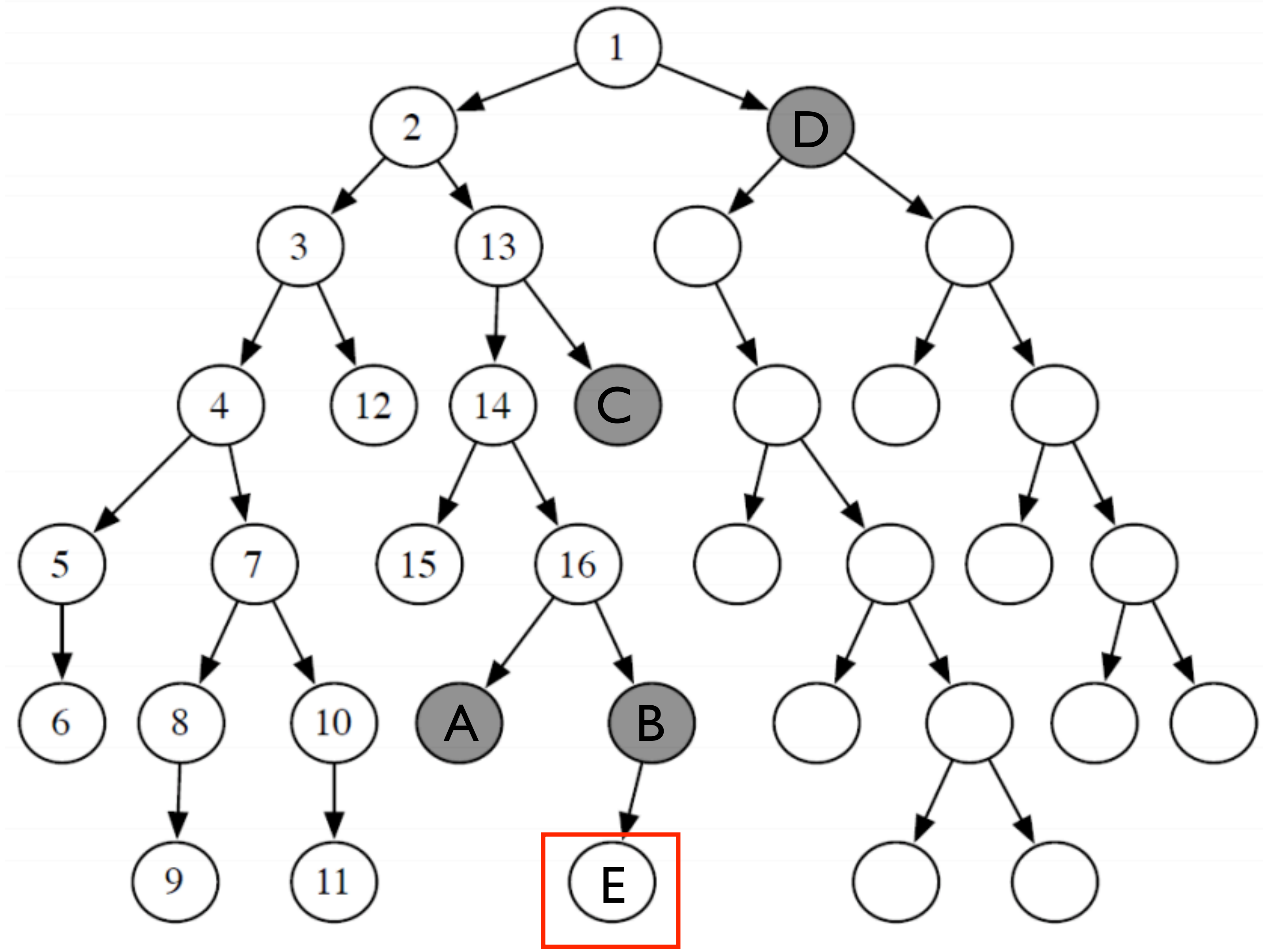
Depth-first search (DFS)

- Frontier?
  - Shaded nodes
- Which node will be expanded next?
- expand = “remove path ending at node from frontier & put its successors on”



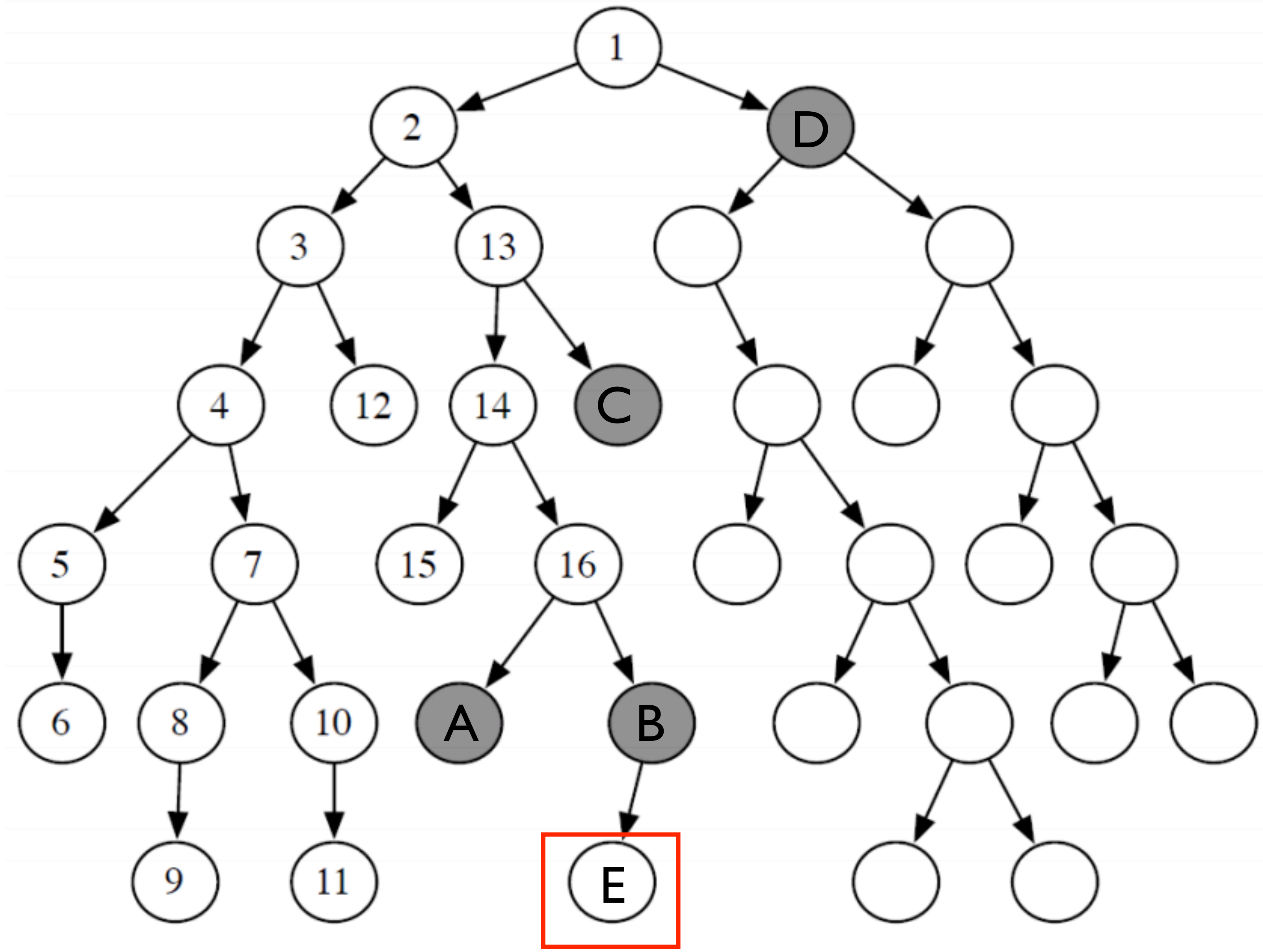
Depth-first search (DFS)

- Say, node in red box is a goal
- How many more nodes will be expanded?



Depth-first search (DFS)

- Say, node in red box is a goal
- How many more nodes will be expanded?



3 because you only return once the goal is being expanded and not when a goal is put onto the frontier

Analysis of DFS

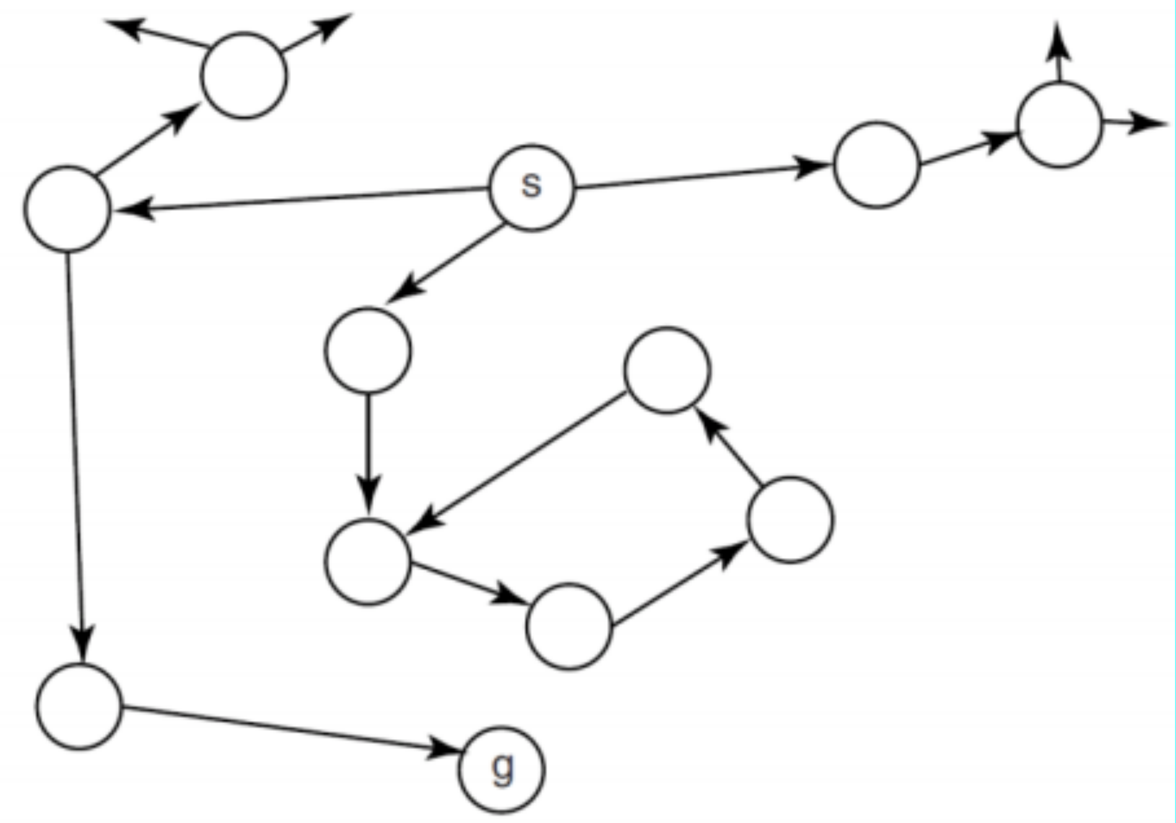
Completeness of DFS

A search algorithm is **complete** if, whenever at least one solution exists, the algorithm is guaranteed to find a solution within a finite amount of time.

Is DFS complete?

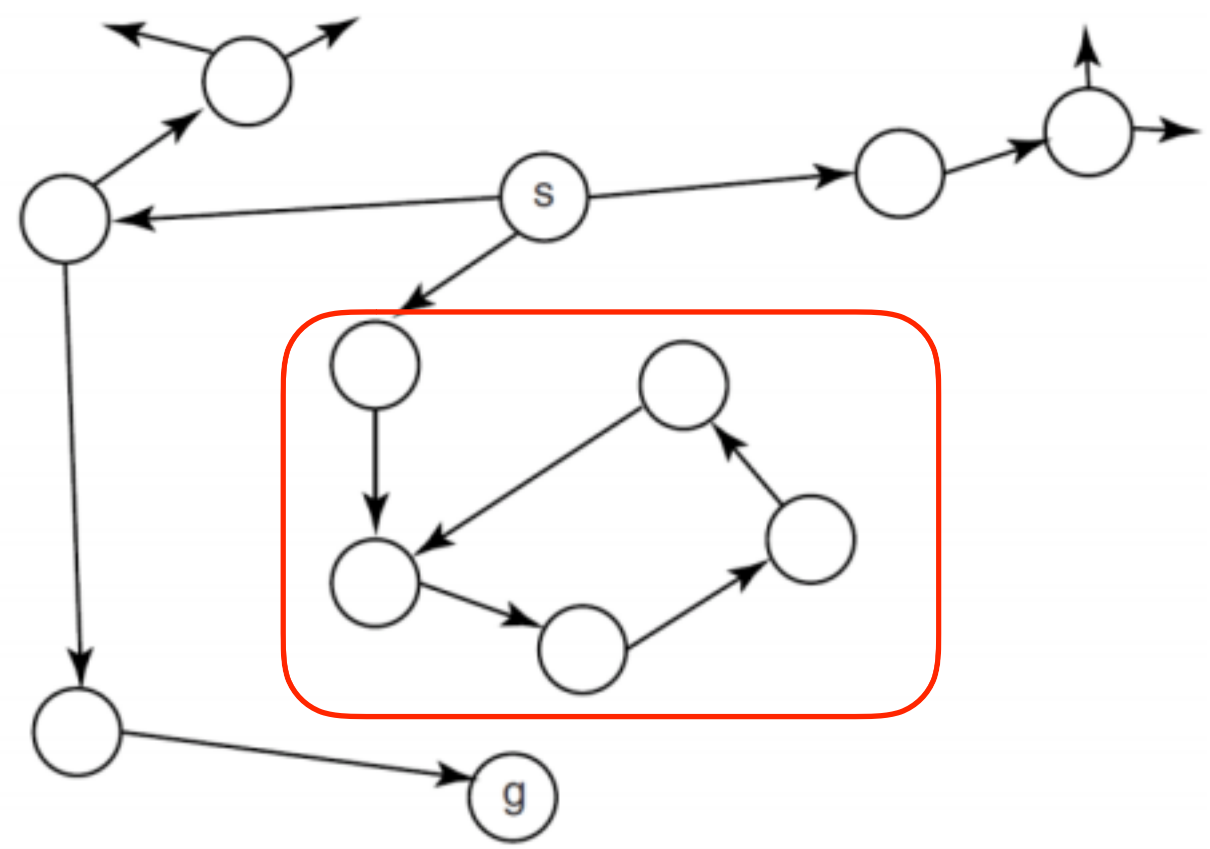
A. Yes

B. No ✅



Completeness of DFS

If there are cycles in the graph, DFS might get "stuck" in one of them.



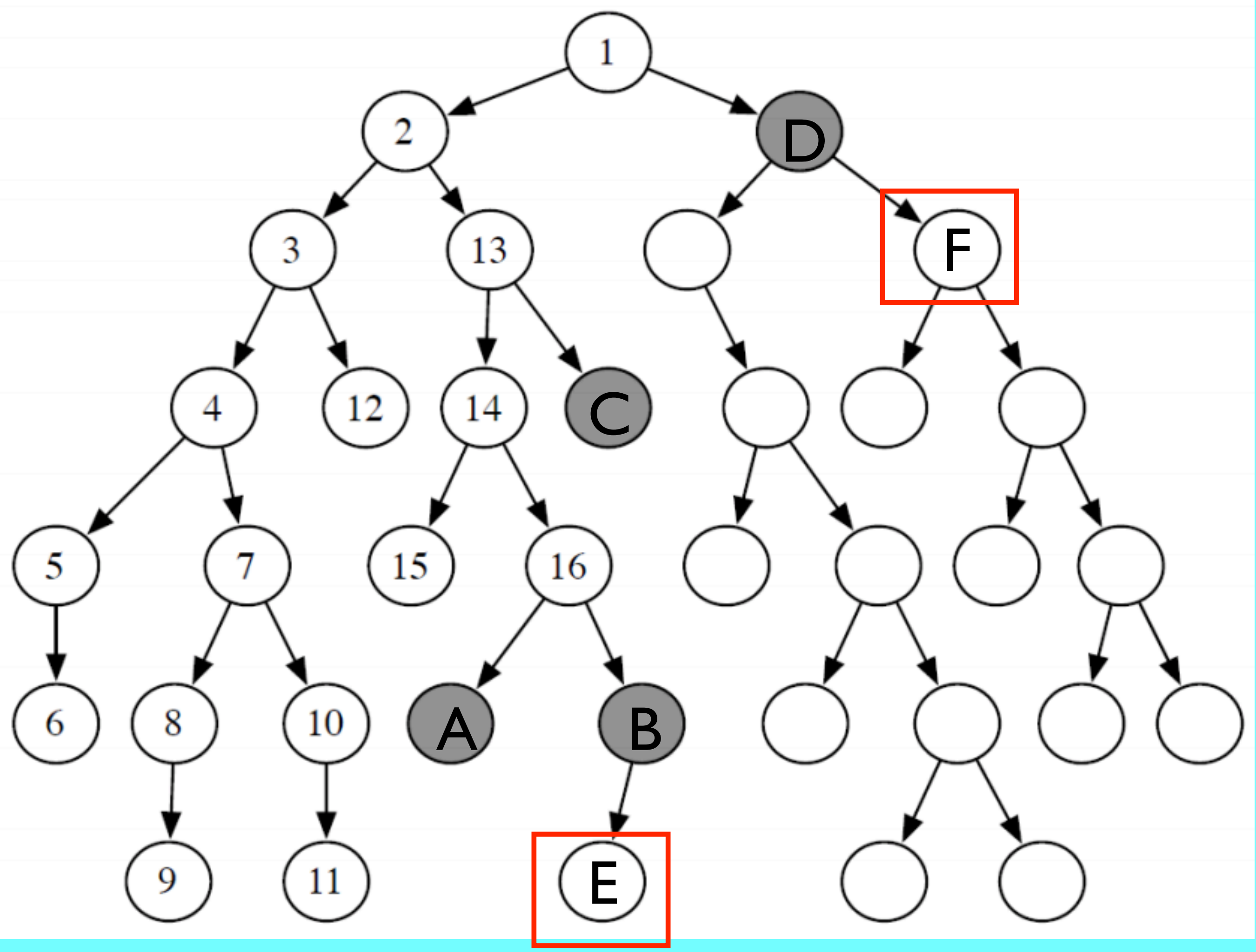
Optimality of DFS

A search algorithm is **optimal** if, when it returns a solution, it is the best solution (i.e. there is no better solution).

Is DFS optimal?

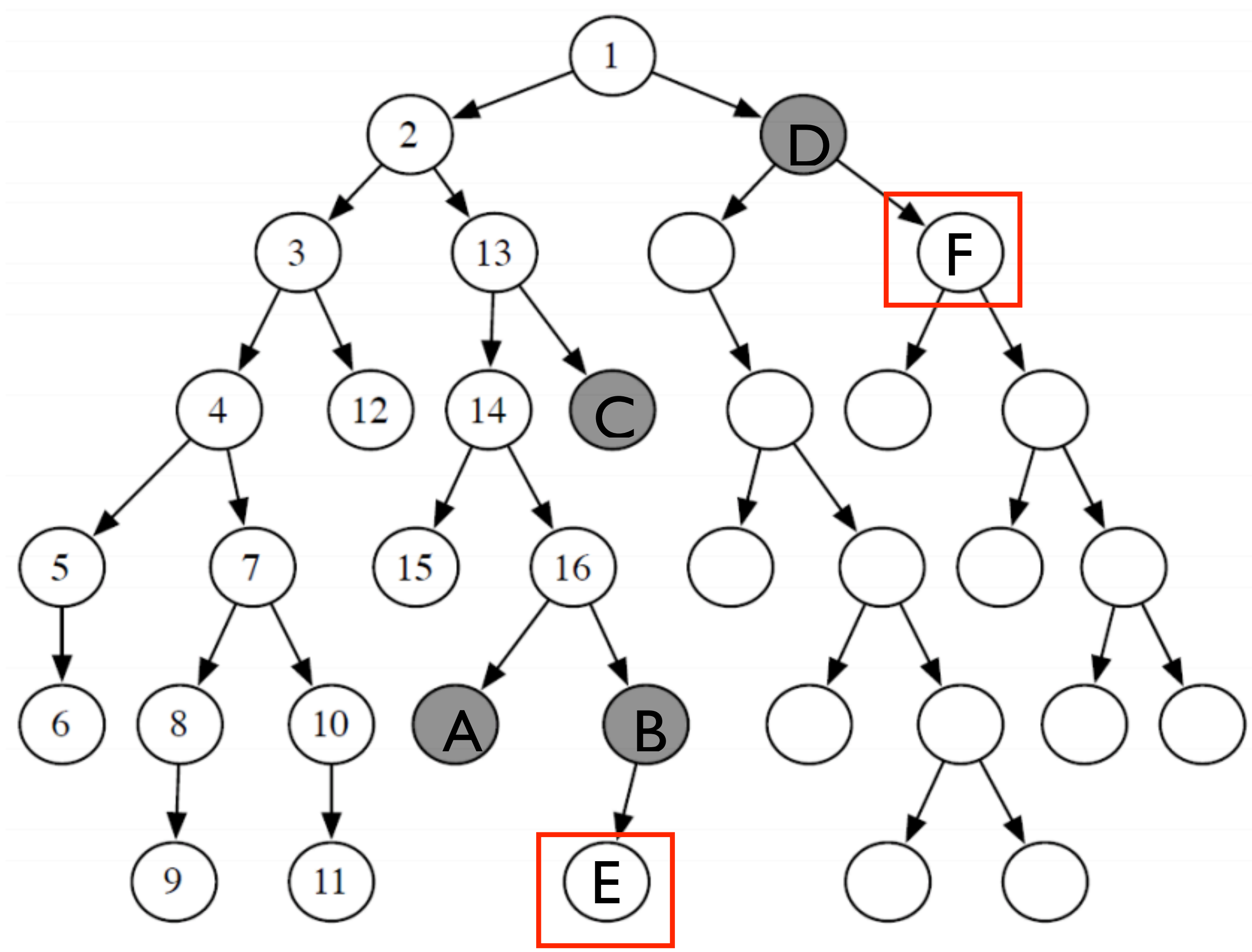
A. Yes

B. No ✅



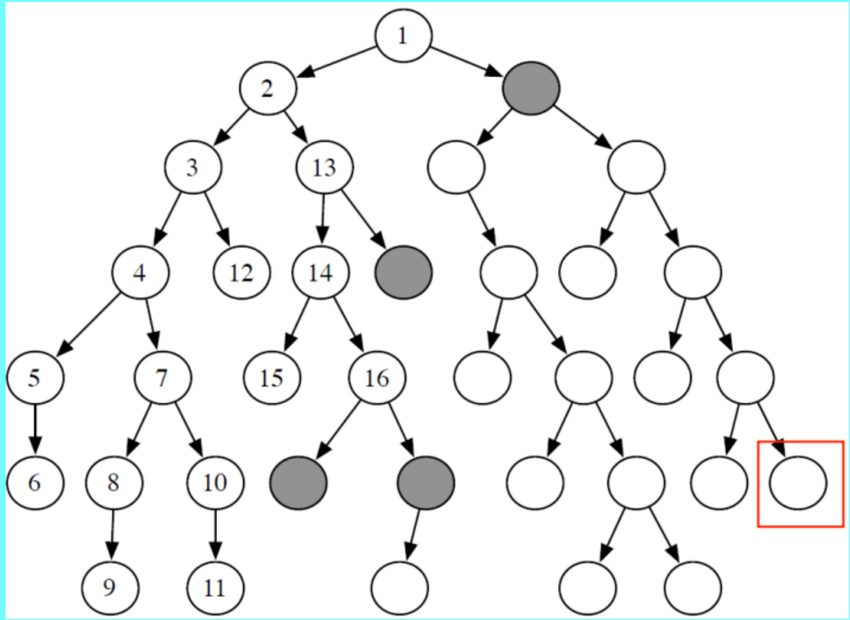
Optimality of DFS

It can "stumble" onto longer solution paths before it gets to shorter one.



Time complexity of DFS

The **time complexity** of a search algorithm is an expression for the worst-case amount of time it will take to run expressed in terms of the maximum path length m and the maximum branching factor b



Single goal node

What's the time complexity of DFS in terms of m and b?

A. $O(b^m)$

B. $O(m^b)$

C. $O(mb)$

D. $O(b+m)$

Time complexity of DFS

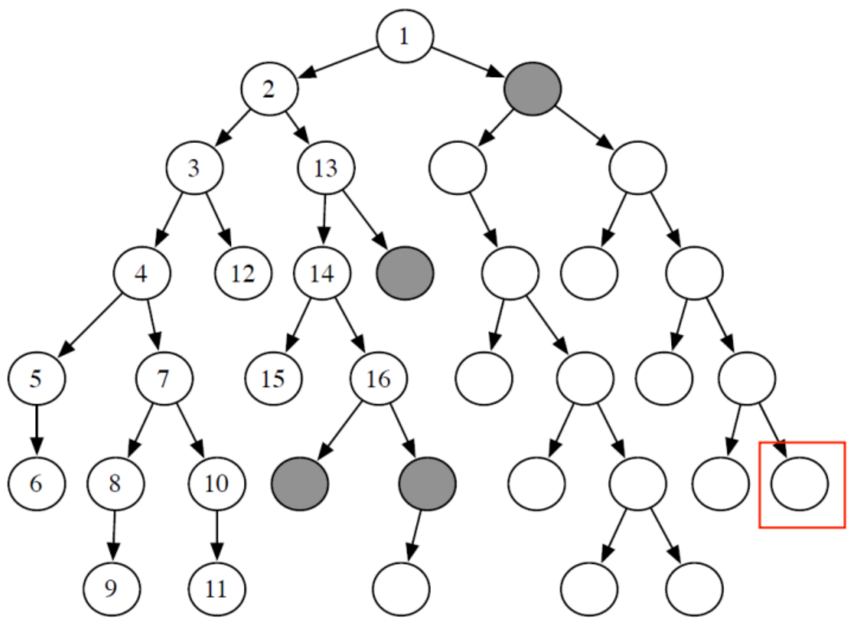
What's the time complexity of DFS in terms of m and b?

A. $O(b^m)$ ✅

B. $O(m^b)$

C. $O(mb)$

D. $O(b + m)$



Single goal node

In the worst case, it must examine every node in the search tree.

Space complexity of DFS

The **space complexity** of a search algorithm is an expression for the worst-case amount of memory that the algorithm will use (number of paths), expressed in terms of the maximum path length m and the maximum branching factor b.

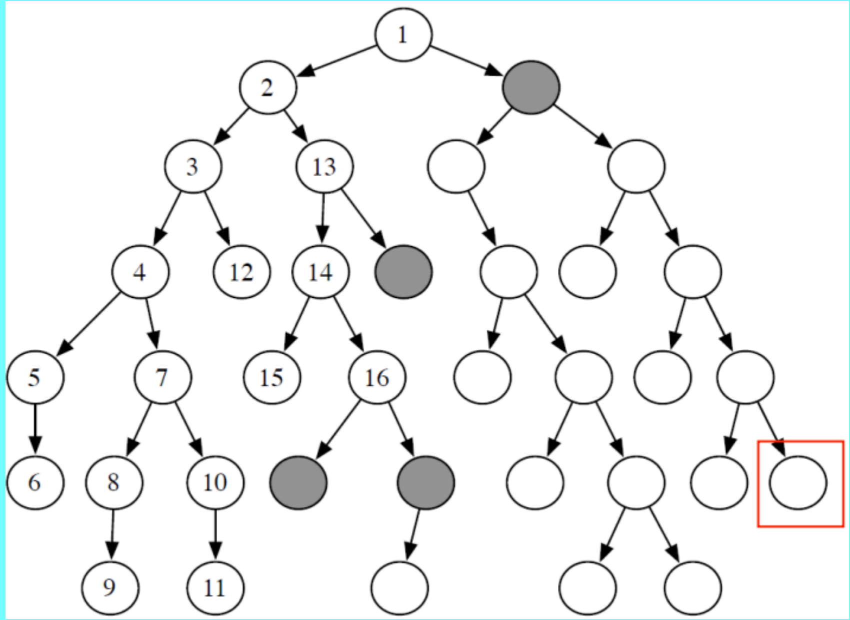
What's the space complexity of DFS in terms of m and b?

A. $O(b^m)$

B. $O(m^b)$

C. $O(mb)$

D. $O(b+m)$



Single goal node

Space complexity of DFS

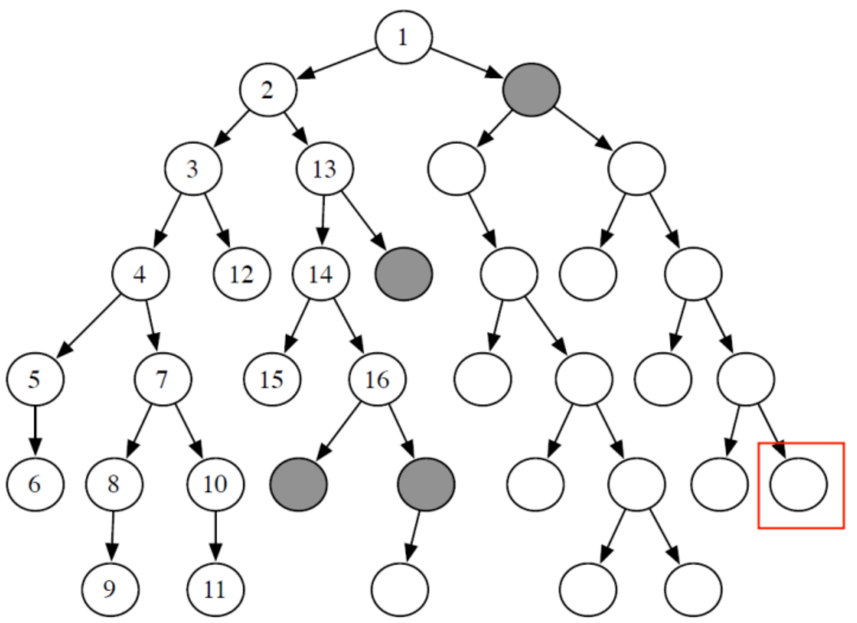
What's the space complexity of DFS in terms of m and b?

A. $O(b^m)$

B. $O(m^b)$

C. bm ✅

D. $O(b+m)$



Single goal node

The longest possible path is m, and for every node in that path must maintain a fringe of size b.

Summary of DFS analysis

- Is DFS complete? No
  - May not halt on graphs with cycles. However, DFS is complete for finite acyclic graphs.
- Is DFS optimal? No
  - It may stumble on a suboptimal solution first
- What is the time and space complexity, if the maximum path length is m and the maximum branching factor is b?
  - Time complexity is b^m: may need to examine every node in the tree.
  - Space complexity is bm: the longest possible path is m, and for every node in that path we must maintain a "fringe" of size b.

Why bother understanding DFS?

- It is simple enough to allow you to learn the basic aspects of searching (when compared with breadth-first search)
- It is the basis for a number of more sophisticated and/or useful search algorithms

DFS

DFS is appropriate when

A. There are cycles

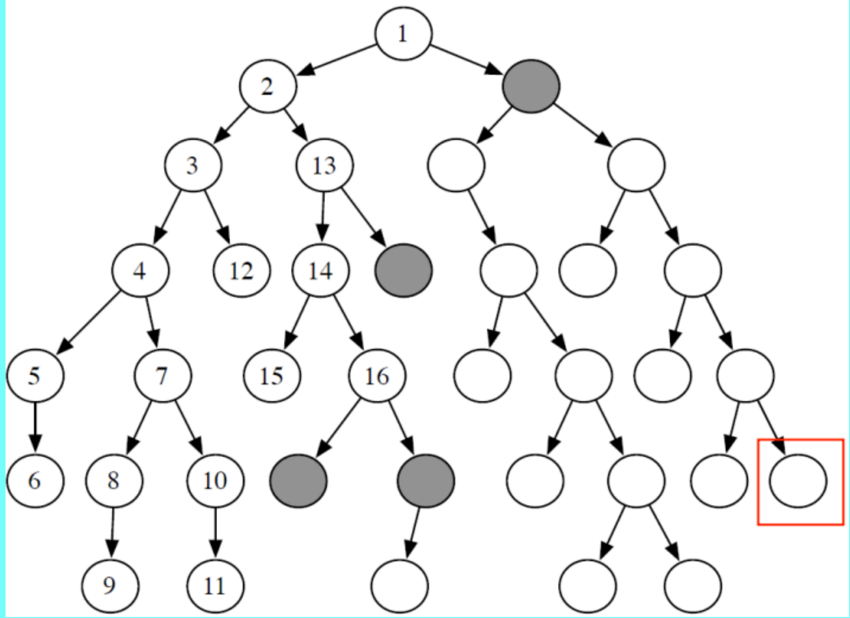
B. There are shallow solutions

C. You care about optimality

D. Space is restricted (complex space representation, e.g., in robotics) ✅

E. You are studying for CPSC 322

Single goal node



DFS: When is it appropriate

Appropriate

- Space is restricted (complex state representation e.g., robotics assembly)
- There are many solutions, perhaps with long path lengths, particularly for the case in which all paths lead to a solution

Inappropriate

- When there are cycles
- When there are shallow solutions
- If you care about optimality

Lecture outline

- Recap from last lecture (~10 mins)
- Criteria to compare search strategies (~10 mins)
- Depth-first search (~15 mins)
- **Breadth-first search (~15 mins)** 👈
- Break (~5 mins)
- Class activity (~15 mins)
- Summary and wrap-up (~5 mins)

BFS as an instantiation of generic search algorithm

In BFS the **frontier** is a **first-in-first-out queue**.

Inputs: a graph,

a start node n_0

a boolean procedure *goal(n)* that tests if n is a goal node

frontier:= [$< n_0 >$: n_0 is a start node];

While frontier is not empty:

select and **remove** path $< n_0, n_1, \ldots, n_k >$ from frontier;

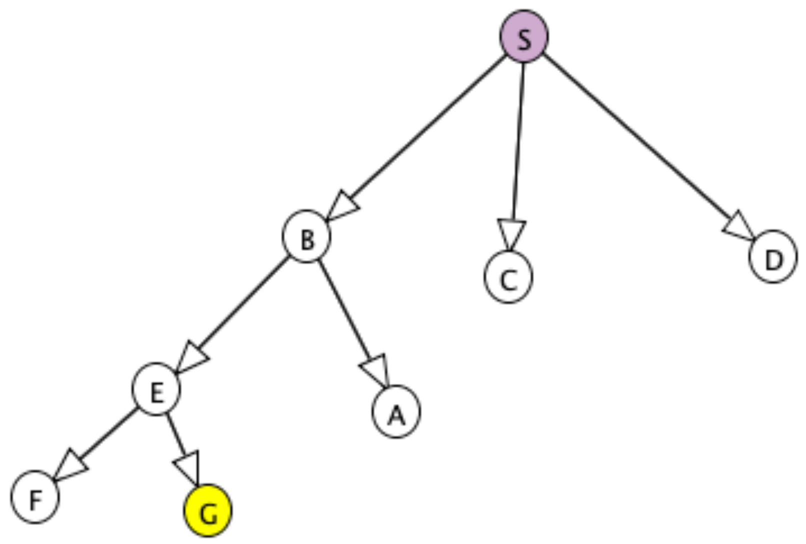
If *goal(n_k)*

return $< n_0, n_1, \ldots, n_k >$;

For every neighbour n of n_k

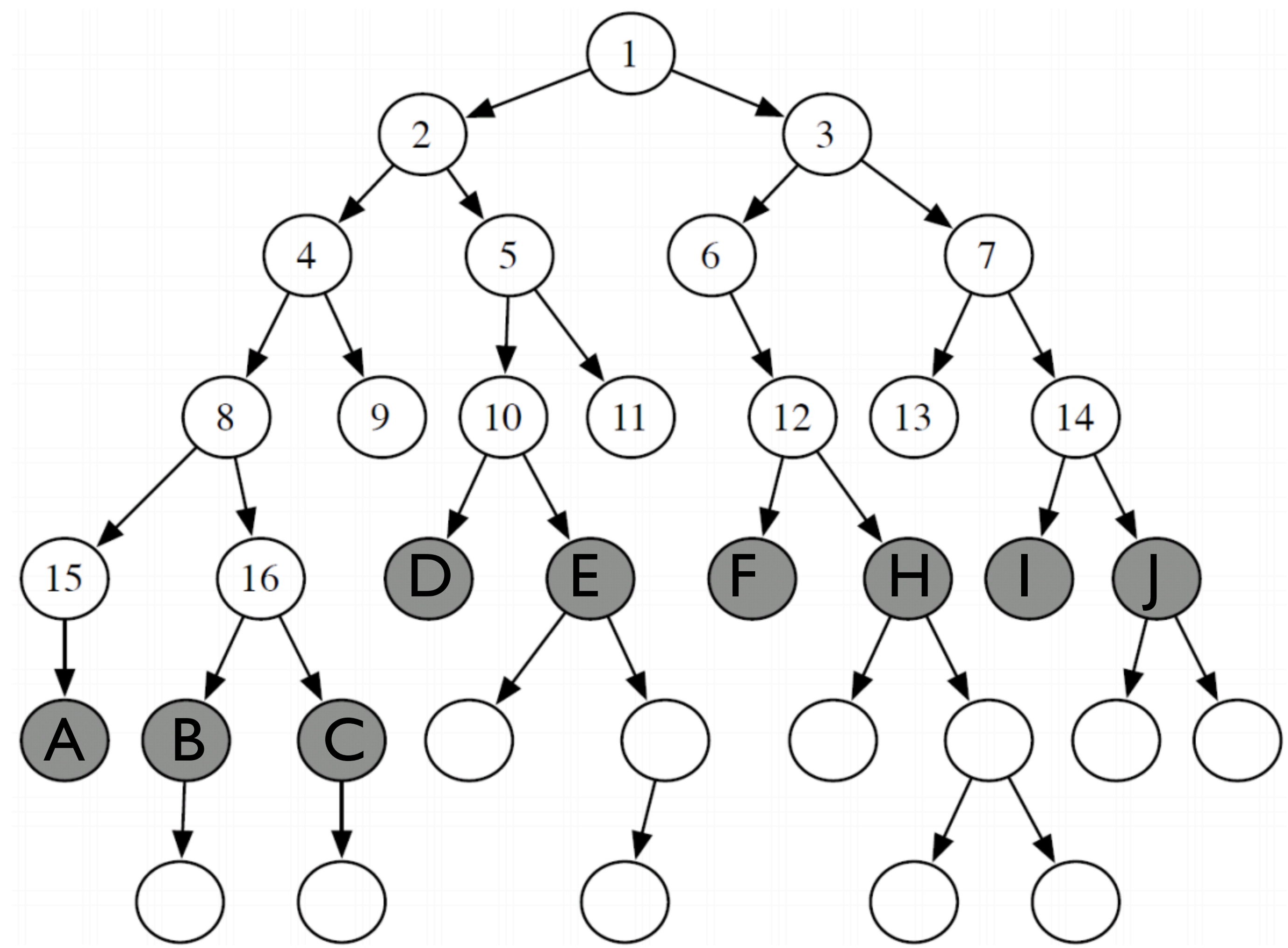
add $< n_0, n_1, \ldots, n_k, n >$ to frontier;

return NULL



Breadth-first search (BFS)

- The shallowest unexpanded node is chosen for expansion.
- All nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.



Analysis of BFS

Completeness of BFS

iclicker.

A search algorithm is **complete** if, whenever at least one solution exists, the algorithm is guaranteed to find a solution within a finite amount of time.

Is BFS complete?

A. Yes

B. No

Optimality of BFS

A search algorithm is **optimal** if, when it returns a solution, it is the best solution (i.e. there is no better solution)

Is BFS optimal?

A. Yes

B. No

Time complexity of BFS

iclicker

The **time complexity** of a search algorithm is an expression for the worst-case amount of time it will take to run expressed in terms of the maximum path length m and the maximum branching factor b.

What's the time complexity of BFS in terms of m and b?

A. $O(b^m)$

B. $O(m^b)$

C. $O(mb)$

D. $O(b + m)$

Space complexity of BFS

The **space complexity** of a search algorithm is an expression for the worst-case amount of memory that the algorithm will use (number of paths), expressed in terms of the maximum path length m and the maximum branching factor b.

What's the space complexity of BFS in terms of m and b?

A. $O(b^m)$

B. $O(m^b)$

C. $O(mb)$

D. $O(b + m)$

Summary of BFS analysis

- Is BFS complete? Yes
  - Does not get stuck in cycles
- Is BFS optimal? Yes
  - Guaranteed to find the path that involves fewest arcs
- What is the time and space complexity, if the maximum path length is m and the maximum branching factor is b?
  - Time complexity is b^m: may need to examine every node in the tree.
  - Space complexity is b^m: frontier contains all paths of the relevant length (which is <= the shortest path length to a goal node)

Time and memory requirements of BFS

The numbers in the table assume branching factor $b = 10$; a million nodes/second and 1000 byte/node

Depth	Nodes	Time	Memory
2	10^2	.11 ms	107 KB
4	10^4	11 ms	10.6 MB
6	10^6	1.1 sec	1 GB
8	10^8	2 mins	103 GB
10	10^{10}	3 hours	10 TB
12	10^{12}	13 days	1 Petabyte
14	10^{14}	3.5 years	99 petabyte
16	10^{16}	350 years	10 exabyte

Source: Russell and Norvig Figure 3.13

BFS: When is it appropriate?

Appropriate

- Space is not a problem
- It's necessary to find the solution with the fewest arcs
- Although all solutions may not be shallow, at least some are

Inappropriate

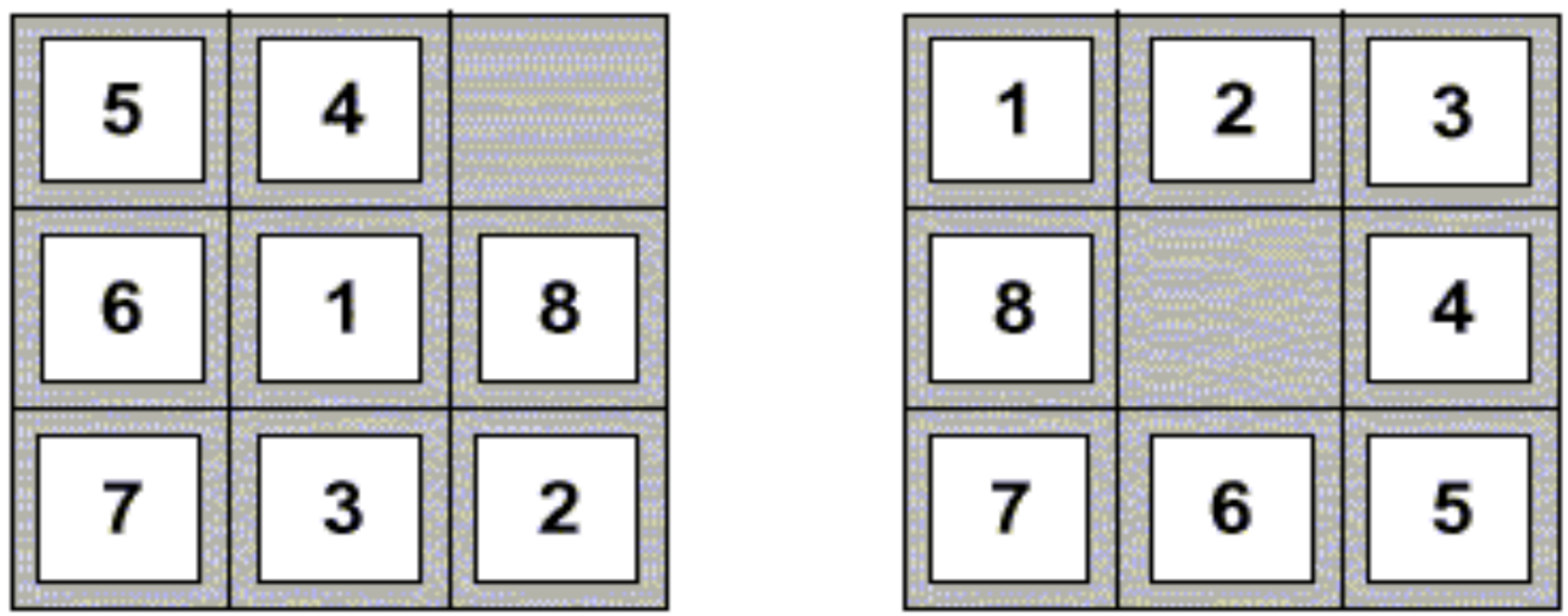
- Space is limited
- All solutions tend to be located deep in the tree
- The branching factor is very large

When to use BFS vs. DFS

A. BFS **B. DFS**

1. The search graph has cycles or is infinite.
2. We need a shortest path to a solution.
3. There are only solutions at great depth.
4. There are some solutions at shallow depth.
5. Memory is limited.

Real example: Solving Eight Puzzle



Which search method would you use if you want to find the shortest solution?

A. BFS

B. DFS

What have we done so far?

- Learned how to formulate a problem as a search problem with an initial state, a goal state, a set of actions, and a transition model.
- Studied the generic search algorithm. Studied two instantiations of the generic search algorithm, BFS and DFS, which can find solutions (plans) given a search problem.
- Compared the algorithms using four different measures: completeness, optimality, time and space complexity.

Summary table

	complete?	optimal?	time O( )	space O( )
DFS	FALSE	FALSE	b^m	bm
BFS	TRUE	True*	b^m	b^m

*Assuming arcs all have the same cost. (We'll get to this later.)

Class activity (~15 mins)

Homework

- To test your understanding of today's class
  - Work on Practice Exercise 3.B on http://www.aispace.org/exercises.shtml

Coming up

- Read the following before the next class.
  - For the next Iterative deepening [3.5.3]
  - Search with costs [3.5.4]
  - Heuristic Search [3.6]