AI+X :
DEEP LEARNING

‹‹‹ 딥러닝을 위한 AI 기초 2 ›››

Start

AI+X : DEEP LEARNING

3. Search Procedure

Search: Abstract Definition

Q. How to search?

- Start at the start state
- Consider the effect of taking different actions starting from states that have been encountered in the search so far
- Stop when a goal state is encountered

"To make this more formal, we'll need review the **formal definition of a graph...**"

Search Graph(1/2)

graph: A graph consists of a set N of **nodes** and a set A of ordered pairs of nodes, called **arcs**.

Node n_2 is a ***neighbor*** of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in$ A.

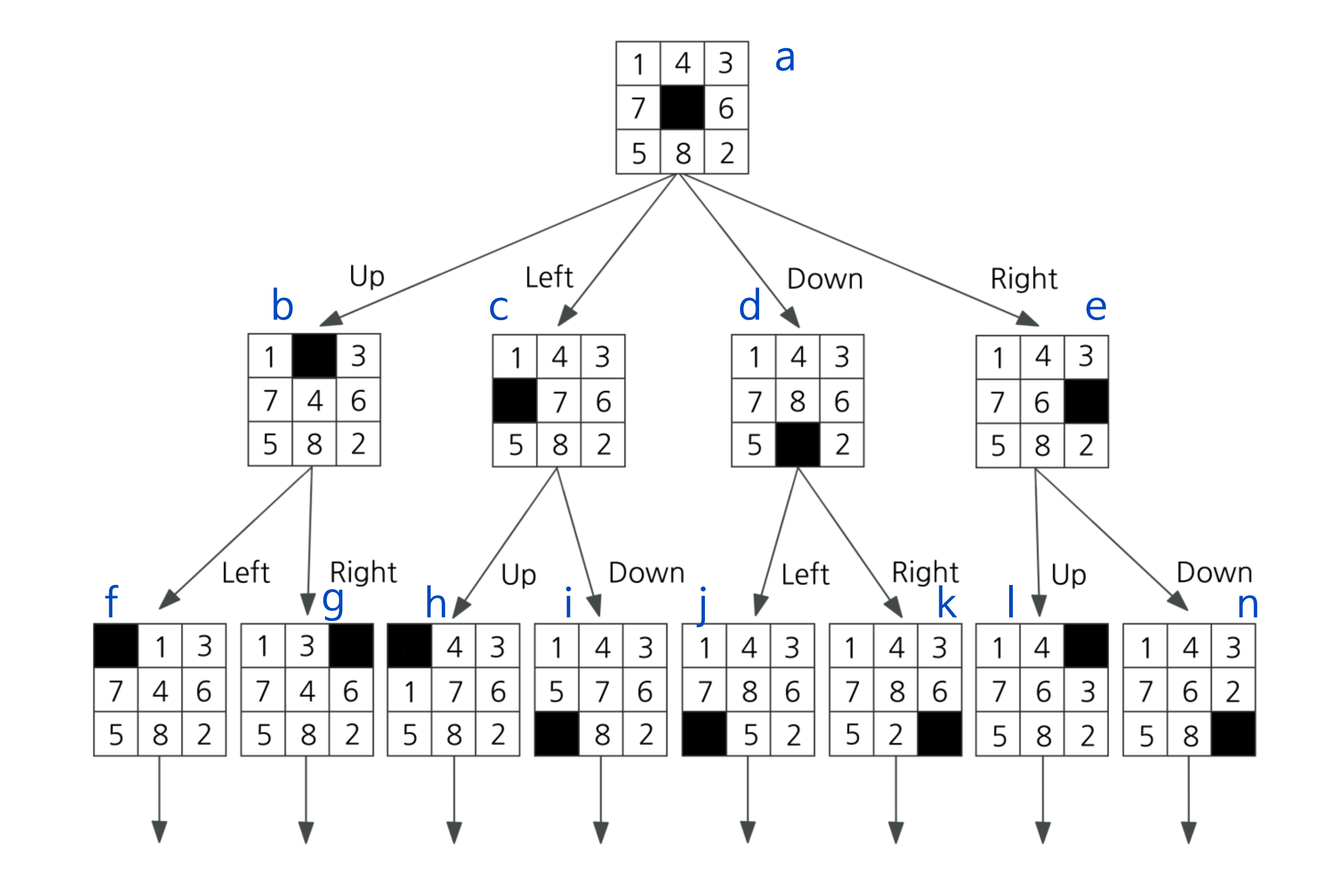
path: A ***path*** is a sequence of nodes $n_0, n_1, n_2, \ldots, n_k$ such that $\langle n_{i-1}, n_i \rangle \in$ A.

Search Graph(2/2)

cycle	A *cycle* is a non-empty path such that the start node is the same as the end node
directed acyclic graph	A *directed acyclic graph* (DAG) is a graph with no cycles

“ Given a start node and goal nodes,
a **solution** is a path from a start node to a goal node. ”

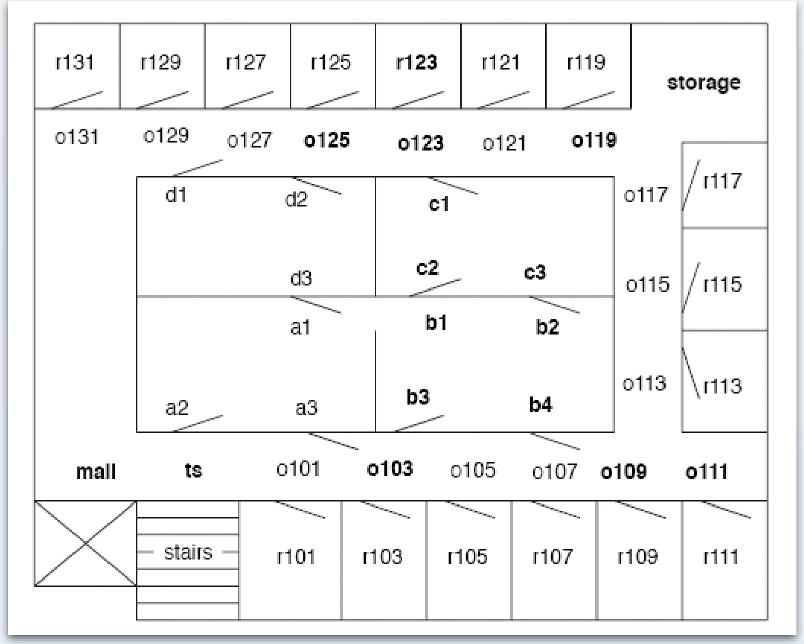
Examples for Graph Formal Def.

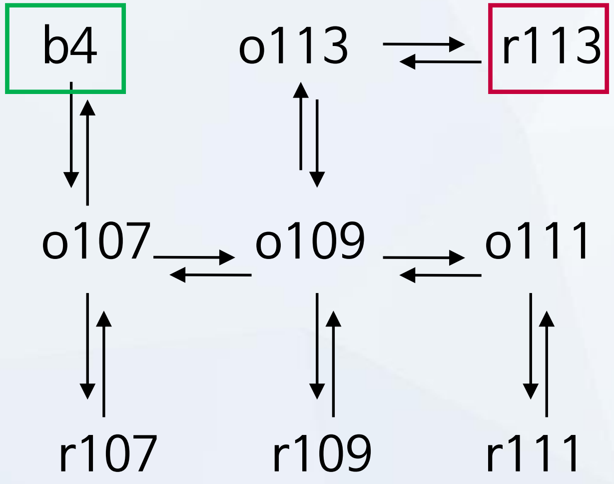


3. Search Procedure

Solution

- Start state b4, goal r113
- Solution ⟨b4, o107, o109, o113, r113⟩



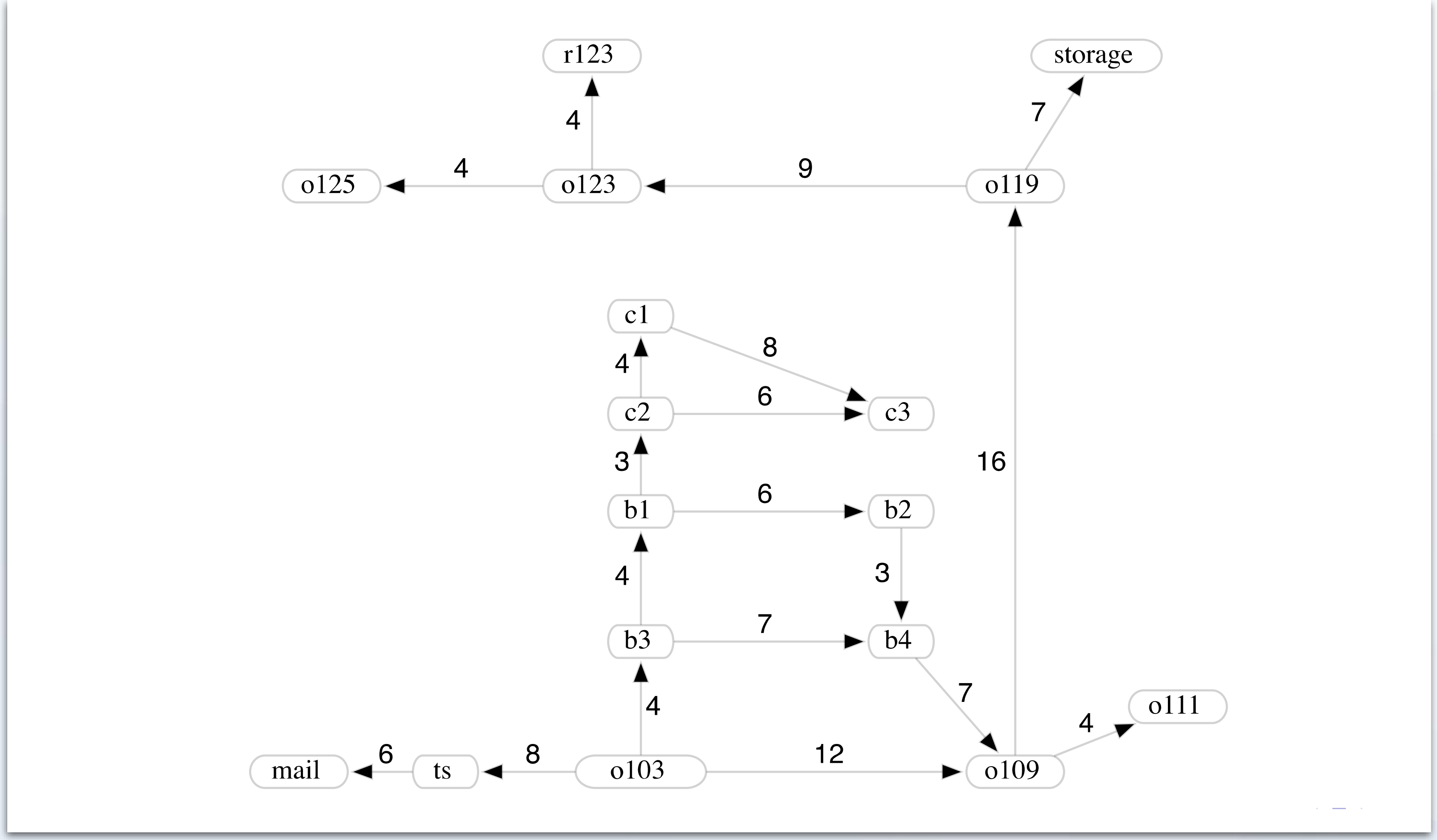


Graph Searching

Generic search algorithm	given a graph, start node, and goal node(s), incrementally explore paths from the start node(s).
frontier of paths	Maintain a **frontier of paths** from the start node that have been explored.

- As search proceeds, the frontier expands into the unexplored nodes until (hopefully!) a goal node is encountered.
- The way in which the frontier is expanded defines the search strategy.

Graph for the delivery robot



3. Search Procedure

Generic Search Algorithm

Input: a graph, a start node, Boolean procedure *goal(n)* that tests if *n* is a goal node

frontier:= [<*s*>: *s* is a start node];

While *frontier* is not empty:

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

    If $goal(n_k)$

        return $<n_o,\ldots,n_k>$;

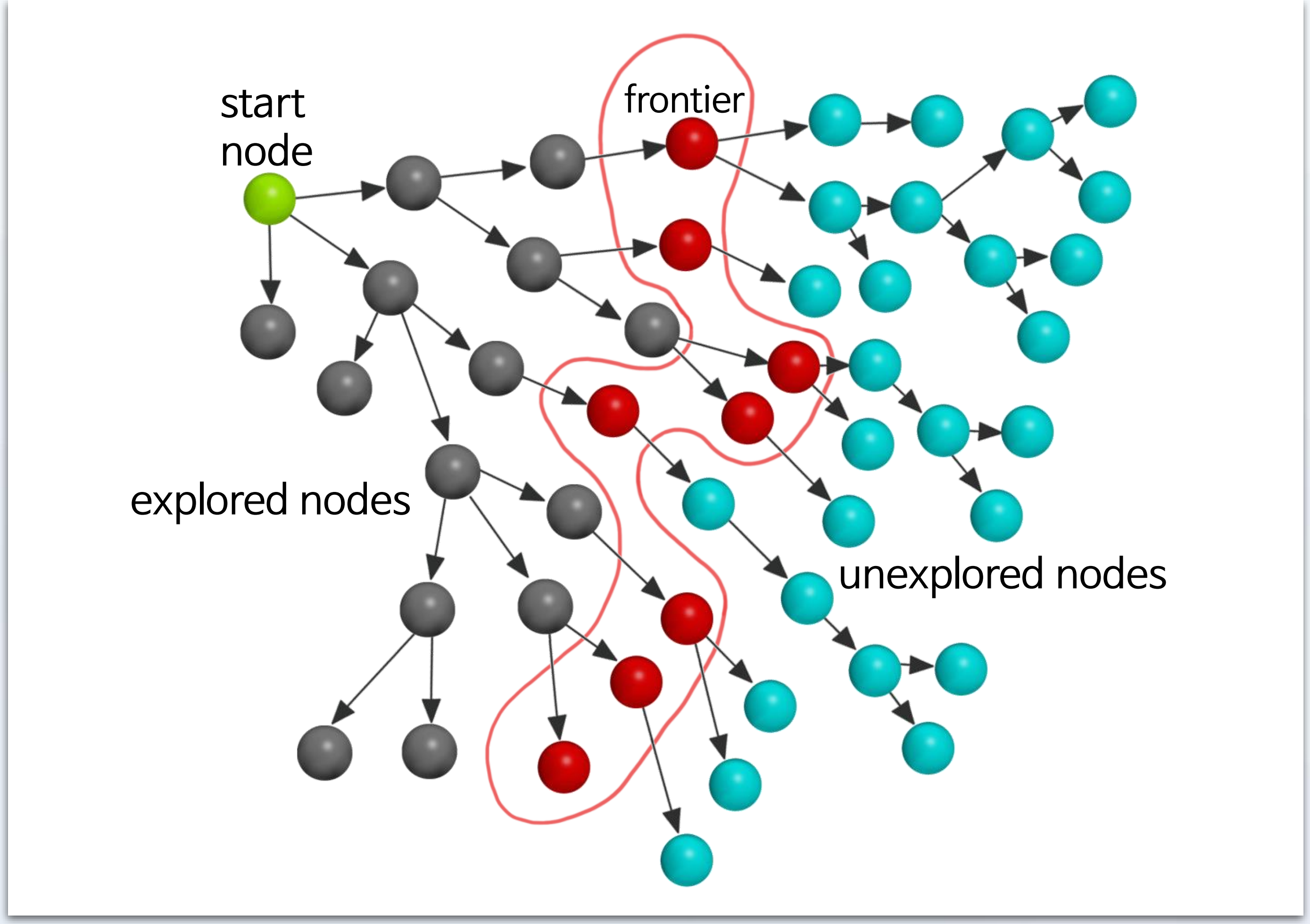
    For every neighbor *n* of n_k

        add $<n_o,\ldots,n_k, n>$ to *frontier;*

end

AI+X : DEEP LEARNING

Problem Solving by Graph Searching



Branching Factor

forward branching factor	backward branching factor
The *forward branching factor* of a node is the number of arcs going out of the node	The *backward branching factor* of a node is the number of arcs going into the node

Q. If the forward branching factor of any node is *b* and the graph is a tree, how many nodes are *n* steps away from a node?

nb b^n n^b n/b

Summary Generic Search Approach

1. Search is a key computational mechanism in many AI agents

2. We will study the basic principles of search on the simple deterministic planning agent model

Summary Generic Search Approach

Generic search approach	▪ define a search space graph, ▪ start from current state, ▪ incrementally explore paths from current state until goal state is reached.

" The way in which the frontier is expanded defines the search strategy. "

AI+X :

DEEP LEARNING

4. Criteria to compare Search Strategies

Comparing Searching Algorithms: will it find a solution? the best one?

Def. (complete)

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.

Def. (optimal)

A search algorithm is **optimal** if, when it finds a solution, it is the best solution.

Comparing Searching Algorithms: Complexity

Def. (time 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*****.**

Def. (space complexity)

The **space complexity** of a search algorithm is an expression for the **worst-case** amount of memory that the algorithm will use (*number of nodes*),

→ Also expressed in terms of ***m*** **and** ***b***.

AI+X :
DEEP LEARNING

5. Simple (Uninformed) Search Strategies

Depth-first Search: DFS

- It treats the frontier as a stack.
- It always selects one of the last elements added to the frontier.

AI+X : DEEP LEARNING

Depth-first Search: DFS

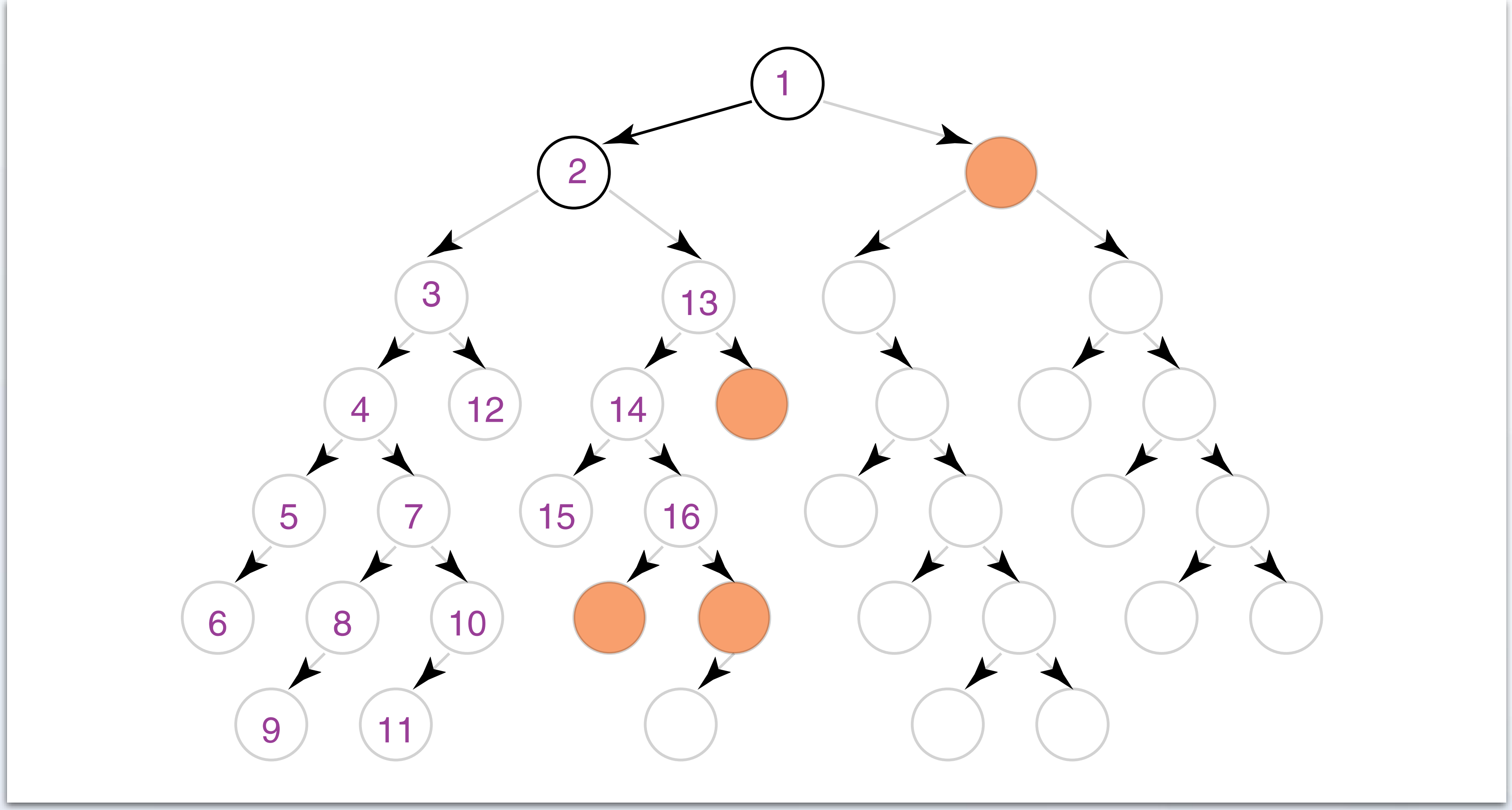
Eg.

- the frontier is $[p_1, p_2, \cdots, p_r]$
- neighbors of last node of p_1 (its end) are $\{n_1, \cdots, n_k\}$

What happens?

- p_1 is selected, and its end is tested for being a goal.
- New paths are created attaching $\{n_1, \ldots, n_k\}$ *to* p_1.
- These **"replace"** p_1 at the beginning of the frontier.
- Thus, the frontier is now $[(p_1, n_1), \ldots, (p_1, n_k), p_2, \ldots, p_r]$.
- ***NOTE:*** p_2 is only selected when all paths extending p_1 have been explored.

DFS



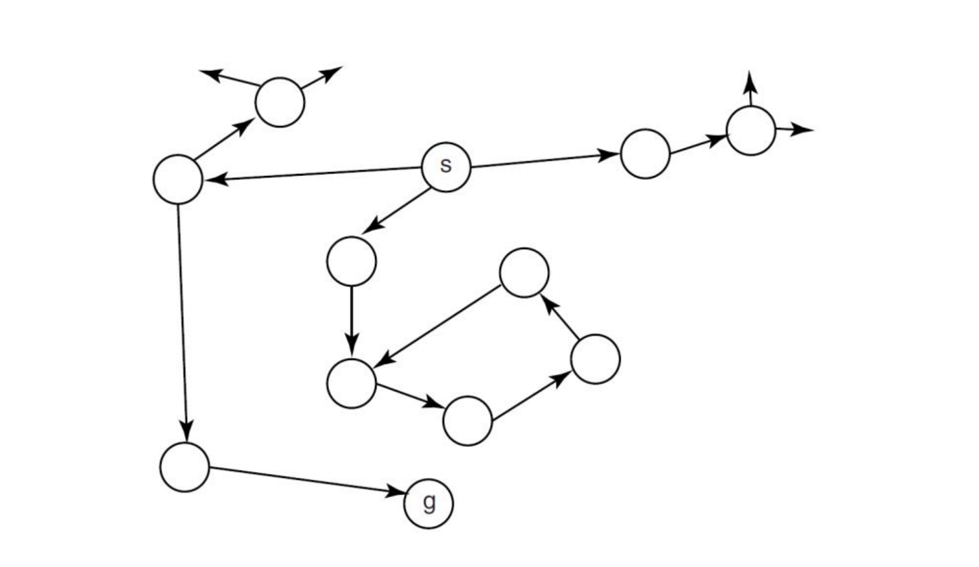
Analysis of DFS

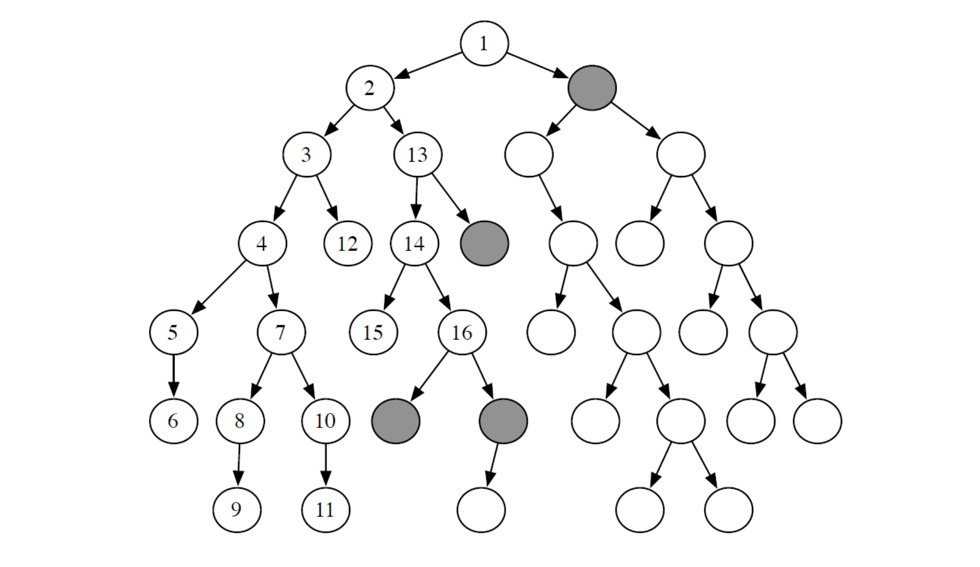
Def.

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

Analysis of DFS

Is DFS complete? NO





- If there are cycles in the graph, DFS may get "stuck" in one of them
- See AISpace by loading "Cyclic Graph Examples" (e.g., http://www.aispace.org/downloads.shtml)

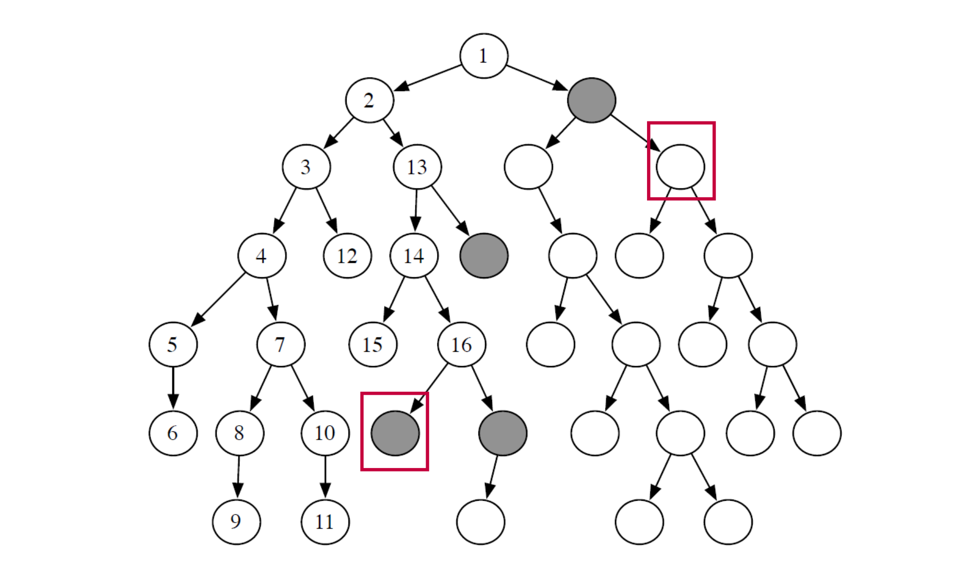
Analysis of DFS

Def.

A search algorithm is **optimal** if when it finds a solution, it **is the best one (e.g., the shortest)**

Is DFS optimal? YES NO

→ E.g., goal nodes: red boxes



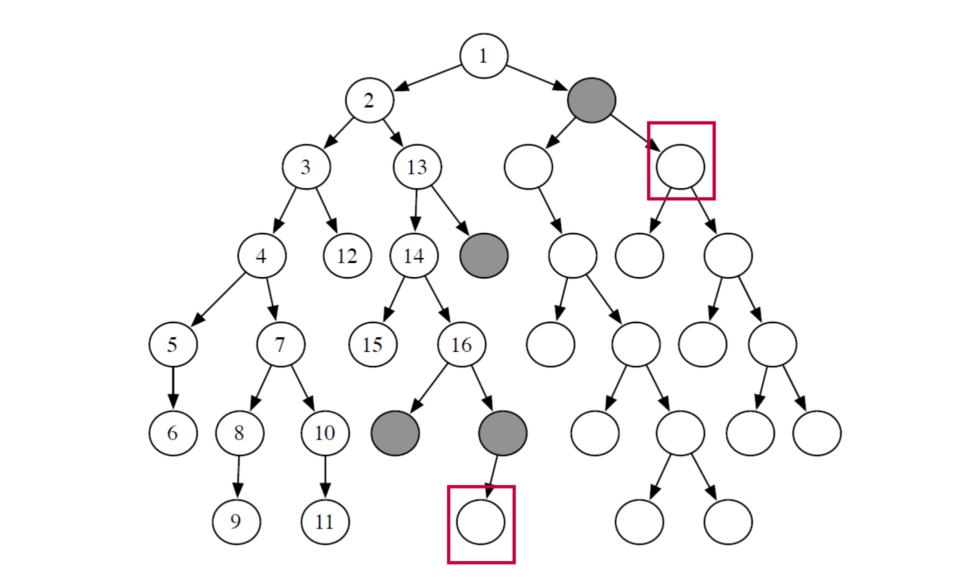
Analysis of DFS

Def.

A search algorithm is **optimal** if when it finds a solution, it **is the best one (e.g., the shortest)**

Is DFS optimal? **NO**

- It can "stumble" on longer solution paths before it gets to shorter ones.
- E.g., goal nodes: red boxes



Analysis of DFS

Def.

The **time complexity** of a search algorithm is the **worst-case** amount of time it will take to run, expressed in terms of

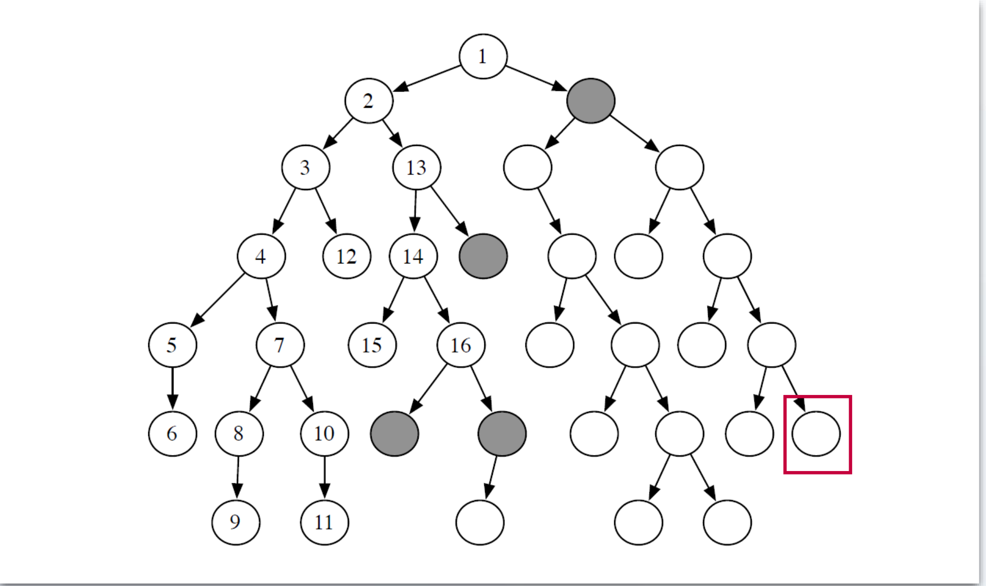
- maximum path length *m*
- maximum forward branching factor *b*.

Analysis of DFS

Q. What is DFS's *time complexity*, in terms of *m* and *b*?

$O(b^m)$	$O(m^b)$	$O(bm)$	$O(b+m)$

→ E.g.,
single goal node : red box

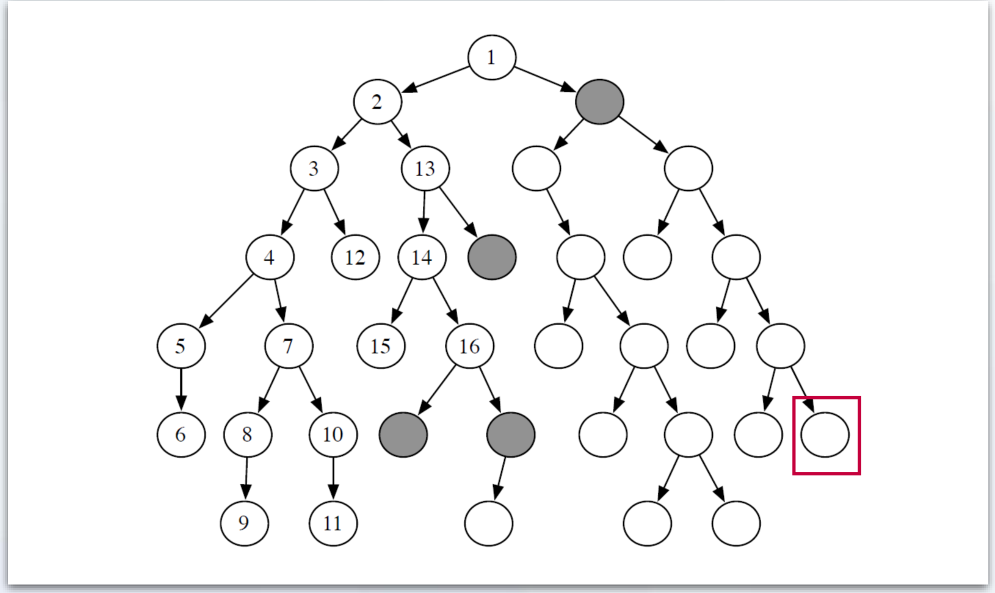


Analysis of DFS

Q. What is DFS's *time complexity*, in terms of *m* and *b*?

$O(b^m)$

- In the worst case, must examine every node in the tree
- E.g., single goal node : red box



Analysis of DFS

Def.

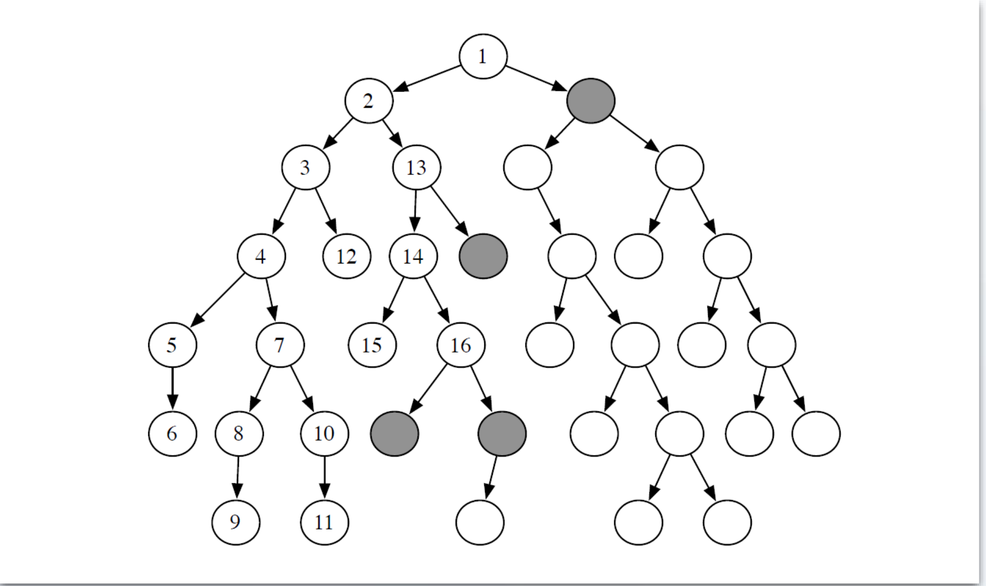
The space complexity of a search algorithm is the worst-case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

- maximum path length m
- maximum forward branching factor b.

Analysis of DFS

Q. What is DFS's *space complexity*, in terms of *m* and *b*?

- $O(b^m)$
- $O(m^b)$
- $O(bm)$
- $O(b+m)$

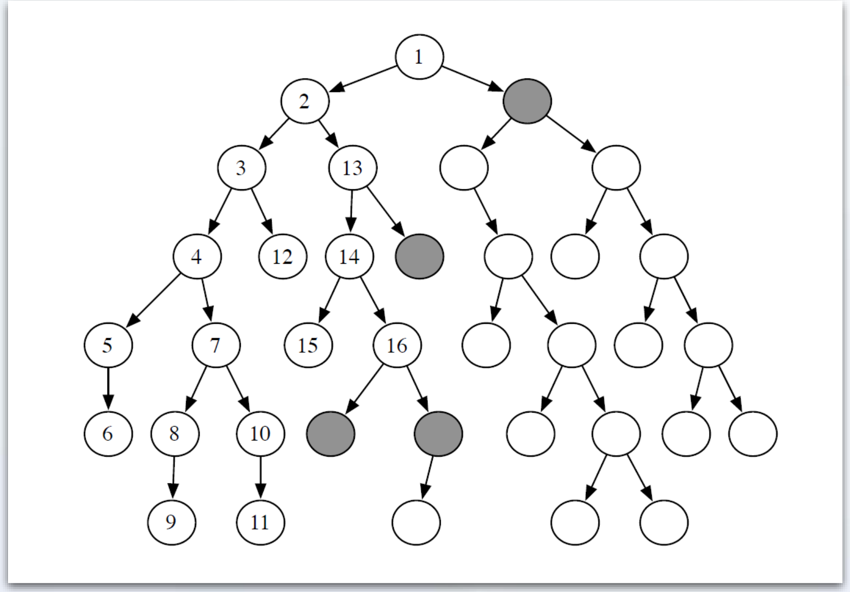


Analysis of DFS

Q. What is DFS's *space complexity*, in terms of *m* and *b*?

O(bm)

- For every node in the path currently explored, DFS maintains a path to its unexplored siblings in the search tree
  - Alternative paths that DFS needs to explore
- The longest possible path is m, with a maximum of b−1 alterative paths per node



Analysis of DFS: Summary

1.

Is DFS complete? NO

- Depth-first search isn't guaranteed to halt on graphs with cycles.
- However, DFS is complete for **finite acyclic graphs.**

2.

Is DFS optimal? NO

- It can "stumble" on longer solution paths before it gets to shorter ones.

Analysis of DFS: Summary

3.

What is the time complexity, if the maximum path length is *m* and the maximum branching factor is *b* ?

- $O(b^m)$: must examine every node in the tree.
- Search is unconstrained by the goal until it happens to stumble on the goal.

4.

What is the space complexity?

- $O(bm)$
- the longest possible path is *m*, and for every node in that path must maintain a fringe of size *b*.

Depth-First Search: When it is appropriate?

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

Why DFS need to be studied and understood?

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

Breadth-first Search: BFS

Breadth-first search	▪ It treats the frontier as a **queue**. ▪ it always selects one of the earliest elements added to the frontier.

AI+X : DEEP LEARNING

Breadth-first Search: BFS

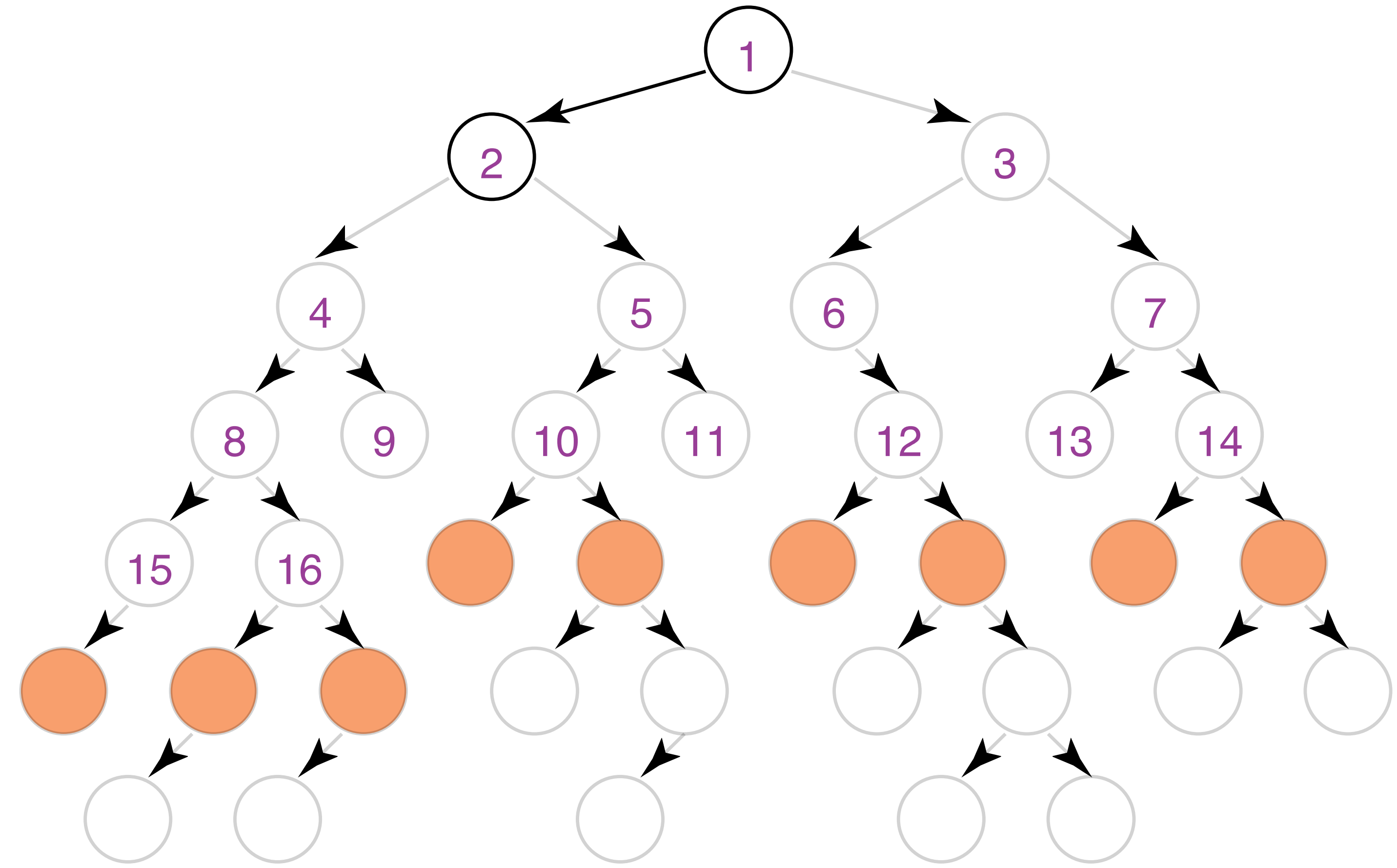
Eg.

- the frontier is $[p_1, p_2, \ldots, p_r]$
- neighbors of the last node of p_1 are $\{n_1, \ldots, n_k\}$

What happens?

- p_1 is selected, and end tested for being a path to the goal.
- New paths are created attaching $\{n_1, \ldots, n_k\}$ *to* p_1.
- These follow p_r at the end of the frontier.
- Thus, the frontier is now $[p_2, \ldots, p_r, (p_1, n_1), \ldots, (p_1, n_k)]$.
- p_2 is selected next.

BFS

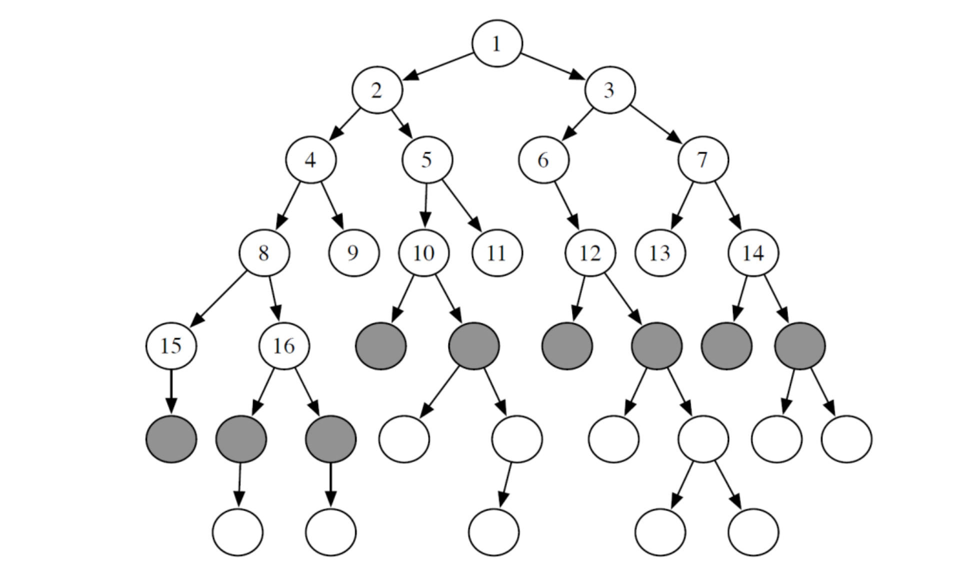


Analysis of BFS

Def.

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

Is BFS complete? **YES** **NO**



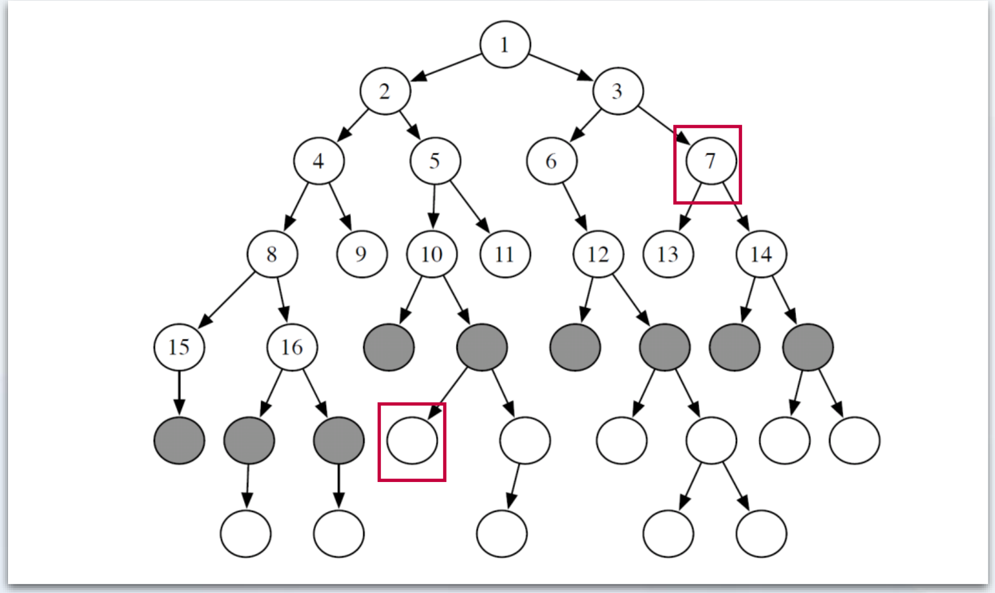
Analysis of BFS

Def.

A search algorithm is **optimal** if when it finds a solution, it is **the best one**

Is BFS optimal? YES NO

→ E.g.,
two goal nodes : red boxes



Analysis of BFS

Def.

The **time complexity** of a search algorithm is the **worst-case** amount of time it will take to run, expressed in terms of

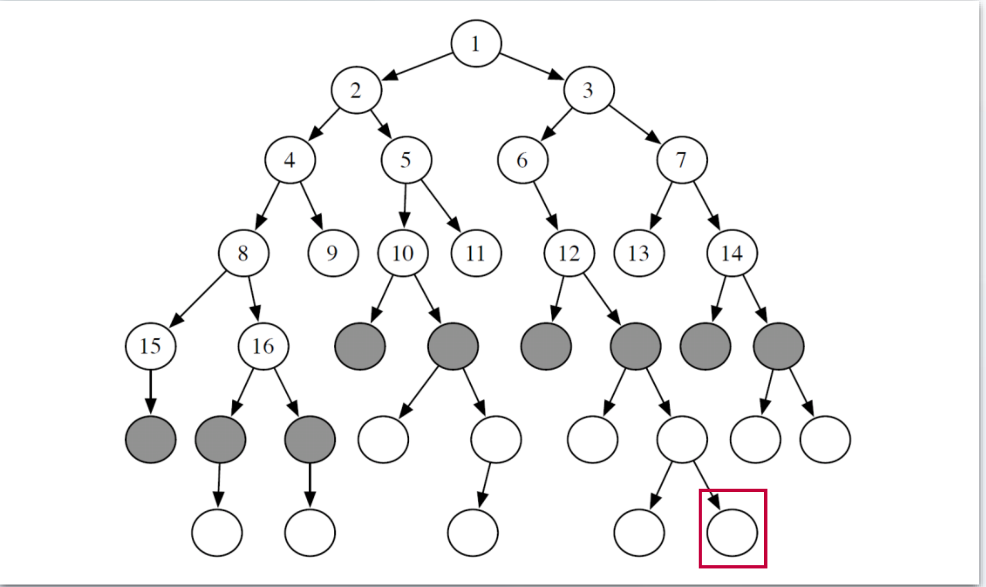
- maximum path length m
- maximum forward branching factor b.

Analysis of BFS

Q. What is BFS's *time complexity*, in terms of *m* and *b* ?

$O(b^m)$	$O(m^b)$	$O(bm)$	$O(b+m)$

→ E.g.,
single goal node : red box



Analysis of BFS

Def.

The **space complexity** of a search algorithm is the **worst-case** amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

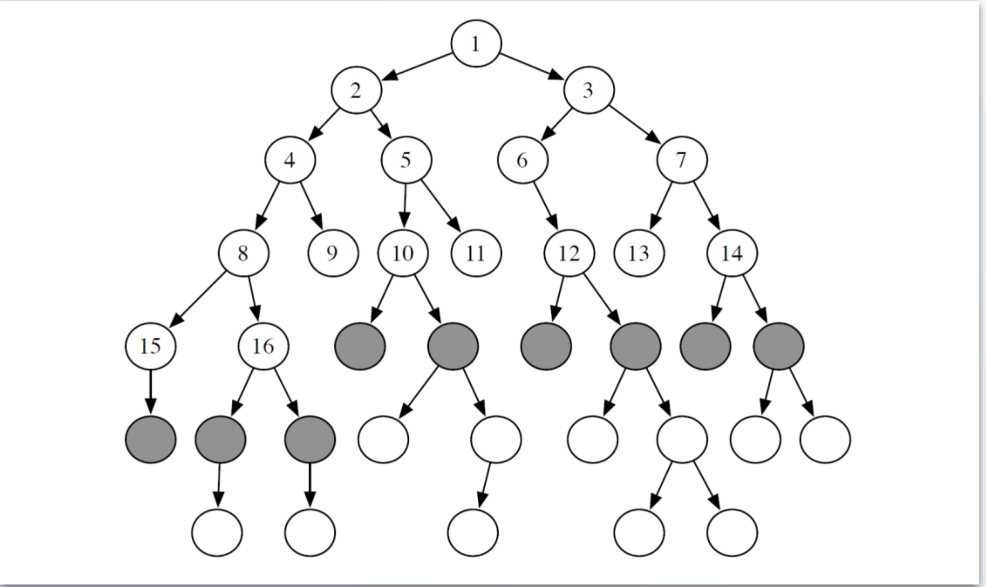
- maximum path length m
- maximum forward branching factor b.

AI+X : DEEP LEARNING

Analysis of BFS

Q. What is BFS's *space complexity*, in terms of *m* and *b*?

$O(b^m)$ | $O(m^b)$ | $O(bm)$ | $O(b+m)$



→ How many nodes at depth m?

Analysis of BFS: Summary

1.

Is BFS complete? YES

- In fact, BFS is guaranteed to find the path that involves the fewest arcs.

2.

What is the time complexity, if the maximum path length is *m* and the maximum branching factor is *b* ?

- The time complexity is $O(b^m)$ must examine every node in the tree.
- The order in which we examine nodes (BFS or DFS) makes no difference to the worst case: search is unconstrained by the goal.

Analysis of BFS: Summary

4.

What is the space complexity?

- Space complexity is $O(b^m)$

Using Breadth-First Search : When it is appropriate?

Appropriate	Inappropriate
▪ 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	▪ space is limited ▪ all solutions tend to be located deep in the tree ▪ the branching factor is very large

What have we done so far?

1.

GOAL

- study search, a set of basic methods underlying many intelligent agents

→ AI agents can be very complex and sophisticated.

→ Let's start from a very simple one, the deterministic, goal-driven agent for which: the sequence of actions and their appropriate ordering is the solution

What have we done so far?

2.

We have looked at two search strategies DFS and BFS

- To understand key properties of a search strategy
- They represent the basis for more sophisticated (heuristic/intelligent) search

AI+X :
DEEP LEARNING

6. Uninformed Iterative Deepening (IDS)

Iterative Deepening

Q. How can we achieve an acceptable (linear) space complexity maintaining completeness and optimality?

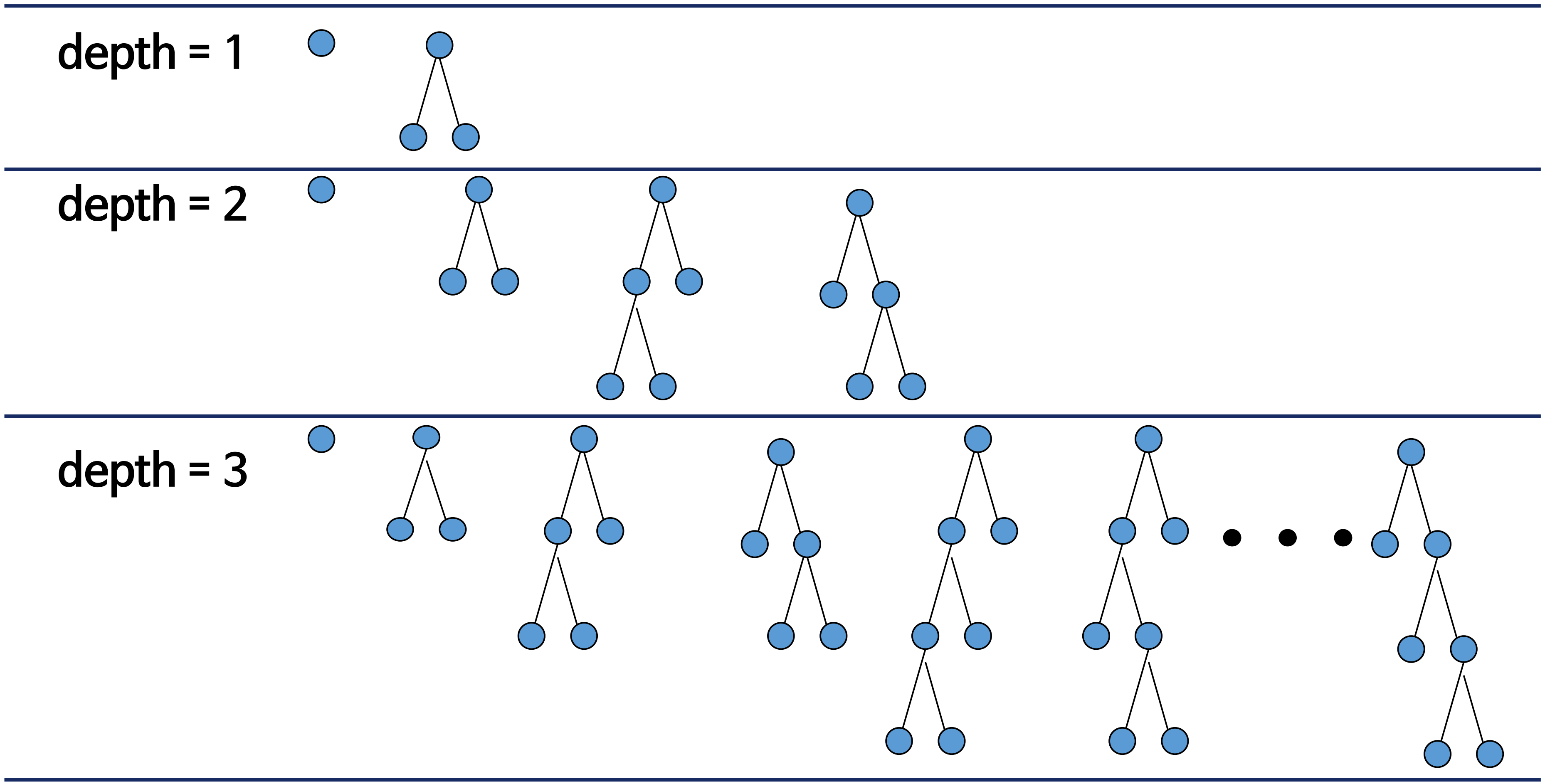
	Complete	Optimal	Time	Space
DFS	NO	NO	b^m	mb
BFS	YES	YES	b^m	b^m
?	YES	YES	b^m	mb

→ **Key Idea**: let's re-compute elements of the frontier rather than saving them.

Iterative Deepening in Essence

- Look with DFS for solutions at depth 1, then 2, then 3, etc.
- If a solution cannot be found at depth *D*, look for a solution at depth *D + 1*.
- You need a depth-bounded depth-first searcher.
- Given a bound B you simply assume that paths of length B cannot be expanded.

Iterative Deepening in Essence



AI+X : DEEP LEARNING

(Time) Complexity of Iterative Deepening

Complexity of solution at depth *m* with branching factor *b*

Total # of paths generated

$$b^m + 2\,b^{m-1} + 3\,b^{m-2} + .. + mb =$$

$$b^m\,(1 + 2\,b^{-1} + 3\,b^{-2} + .. + m\,b^{1-m}) \leq$$

$$b^m\left(\sum_{i=1}^{\infty} i b^{1-i}\right) = b^m\left(\frac{b}{b-1}\right)^2 = O(b^m)$$

Further Analysis of Iterative Deepening DFS (IDS) : Summary

1. Space complexity $O(bm)$
   - DFS scheme, only explore one branch at a time

2. Complexity? YES

Further Analysis of Iterative Deepening DFS (IDS) : Summary

3.

Only paths up to depth m, doesn't explore longer paths

- cannot get trapped in infinite cycles, gets to a solution first

4.

Optimal? YES

AI+X : DEEP LEARNING

7. Search with Costs

Search with Costs

- Sometimes there are costs associated with arcs.

Definition (cost of a path)

The cost of a path is the sum of the costs of its arcs:

$$\text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle)$$

Search with Costs

- In this setting we often don't just want to find just any solution
  - → we usually want to find the solution that **minimizes cost**

Definition (optimal algorithm)

A search algorithm is **optimal** if it is complete, and only returns cost-minimizing solutions.

Lowest-Cost-First Search

- At each stage, lowest-cost-first search selects a path on the frontier with **lowest cost.**
  - The frontier is a priority queue ordered by path cost
  - We say “a path” because there may be ties

Lowest-Cost-First Search

Eg.

- the frontier is $[\langle p_2, 5\rangle, \langle p_3, 7\rangle, \langle p_1, 11\rangle,]$
- p_2 is the lowest-cost node in the frontier
- "neighbors" of p_2 are $\{\langle p_9, 10\rangle, \langle p_{10}, 15\rangle\}$

What happens?

→ p_2 is selected, and tested for being a goal.

→ Neighbors of p_2 are inserted into the frontier

→ Thus, the frontier is now $[\langle p_3, 7\rangle, \langle p_9, 10\rangle, \langle p_1, 11\rangle, \langle p_{10}, 15\rangle]$.

→ etc. etc.

Lowest-Cost-First Search

When arc costs are equal LCFS is equivalent to..

- DFS
- BFS
- IDS
- None of the above

Analysis of Lowest-Cost Search

Q. Is LCFS complete?

- not in general: for instance, a cycle with zero or negative arc costs could be followed forever.
- yes, as long as arc costs are strictly positive $\geq \varepsilon > 0$

Q. Is LCFS optimal?

YES | NO | IT DEPENDS

Analysis of Lowest-Cost Search

Q. Is LCFS complete?

- not in general : a cycle with zero or negative arc costs could be followed forever.
- yes, as long as arc costs are strictly positive

Q. Is LCFS optimal?

- Not in general. Why not?
- Arc costs could be negative : a path that initially looks high-cost could end up getting a "refund".
- However, LCFS is optimal if arc costs are guaranteed to be non-negative.

Analysis of Lowest-Cost Search

Q. What is the time complexity, if the maximum path length is m and the maximum branching factor is b?

- The time complexity is $O(b^m)$: must examine every node in the tree.
- Knowing costs doesn't help here.

Q. What is the space complexity?

- Space complexity is $O(b^m)$: we must store the whole frontier in memory.

Learning Goals for Search

Apply basic properties of search algorithms

completeness, optimality, time and space complexity of search algorithms.

	Complete	Optimal	Time	Space
DFS	NO	NO	b^m	bm
BFS	YES	YES	b^m	b^m
IDS	YES	YES	B^m	bm
LCFS	NO	NO	b^m	b^m