Uninformed Search Strategies

Alan Mackworth

UBC CS 322 – Search 2

January 11, 2013

Textbook §3.5

Today's Lecture

- Lecture 4 (2-Search1) Recap
- Uninformed search + criteria to compare search algorithms
  - Depth first
  - Breadth first

Recap

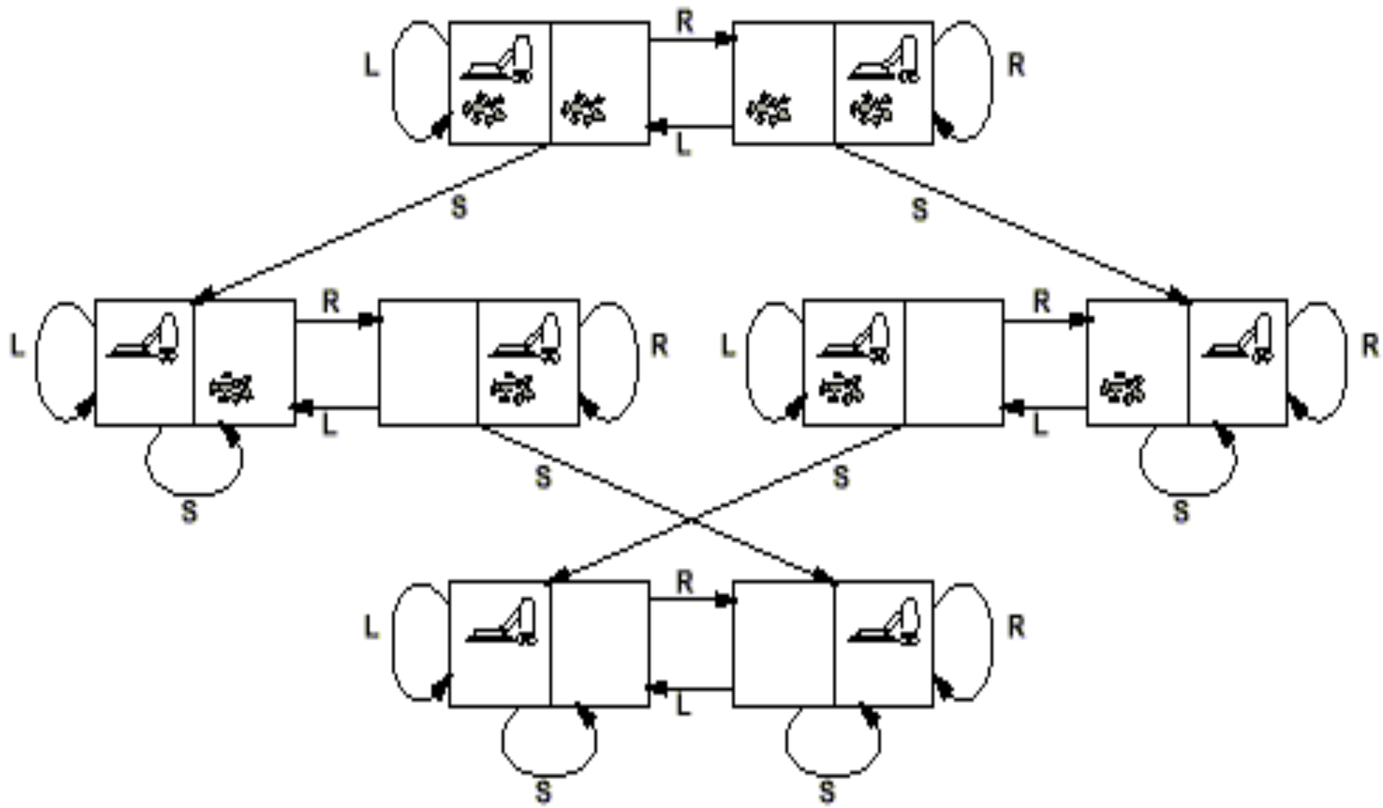
- Search is a key computational mechanism in many AI agents
- We will study the basic principles of search on the simple deterministic goal-driven search agent model

- Generic search approach:
  - Define a search space graph
  - Initialize the frontier with an empty path
  - incrementally expand frontier until goal state is reached

- Frontier:
  - The set of paths which could be explored next

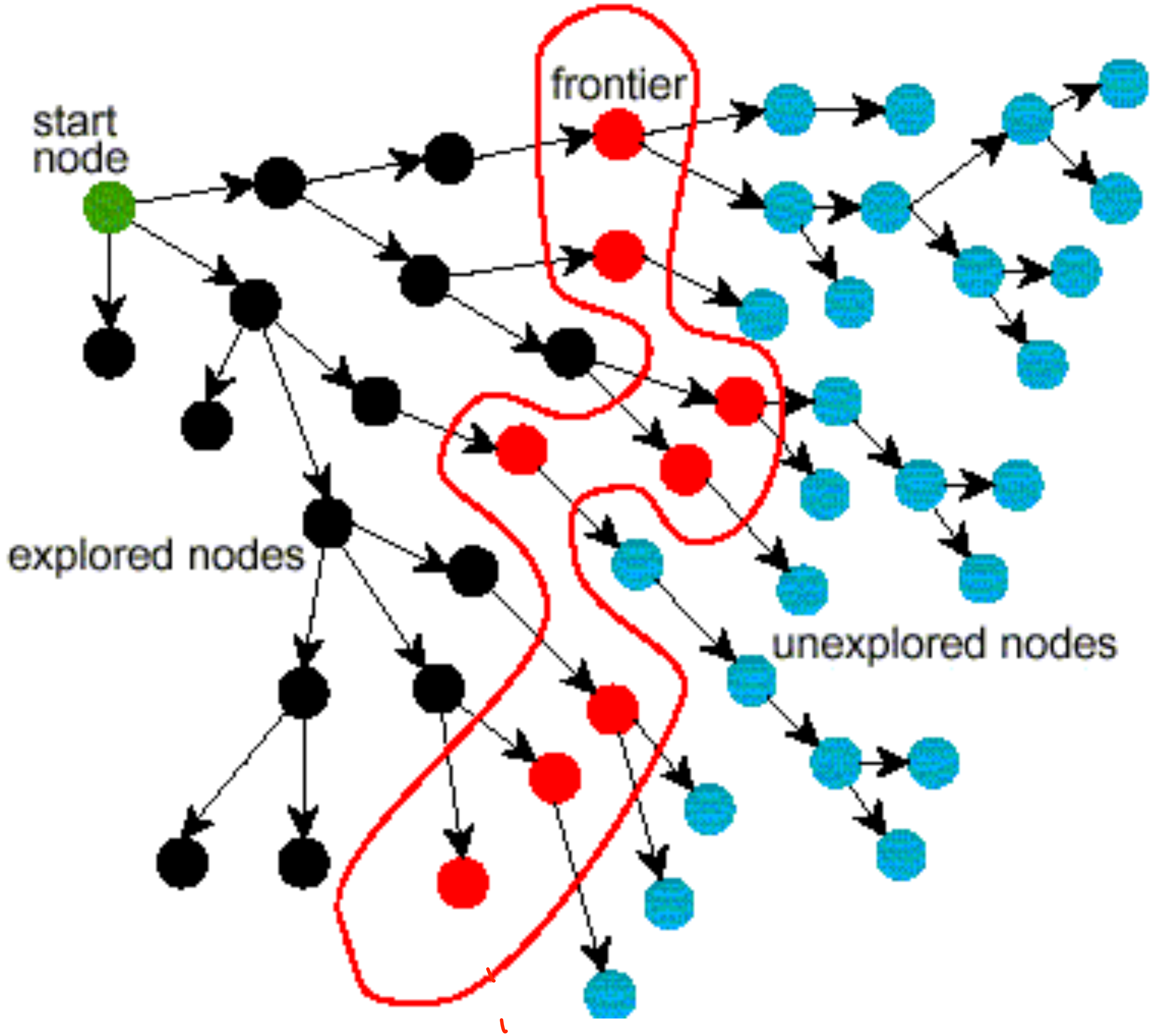
- The way in which the frontier is expanded defines the search strategy

Search Space: example



- **Operators** –*left, right, suck*
  - Successor states in the graph describe the effect of each action applied to a given state
- **Possible Goal** – no dirt

Problem Solving by Graph Searching



Bogus version of Generic Search Algorithm

```
Input: a graph
        a set of start nodes
        Boolean procedure goal(n) that tests if n is a goal node
frontier:= [<g>: g is a goal node];
While frontier is not empty:
    select and remove path <n_o,....,n_k> from frontier;
    If goal(n_k)
        return <n_o,....,n_k>;
     Find a neighbor n of n_k
        add <n> to frontier;
     end
```

- There are several bugs in this version here: help me find them!

Bogus version of Generic Search Algorithm

Input: a graph
a set of start nodes
Boolean procedure goal(n) that tests if n is a goal node
frontier:= [<g>: g is a goal node];
While frontier is not empty:
select and **remove** path $<n_o,\ldots,n_k>$ from frontier;
If goal(n_k)
return $<n_o,\ldots,n_k>$;
Find **a** neighbor n of n_k
add <n> to frontier;
end

- Start at the start node(s)
- Add all neighbours of n_k to the frontier
- Add path(s) to frontier, NOT just the node(s)

Generic Search Algorithm

Input: a graph

a set of start nodes

Boolean procedure goal(n) testing if n is a goal node

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

While frontier is not empty:

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

If goal(n_k)

Then return $<n_o,.....,n_k>$;

Else

For every neighbor n of n_k,

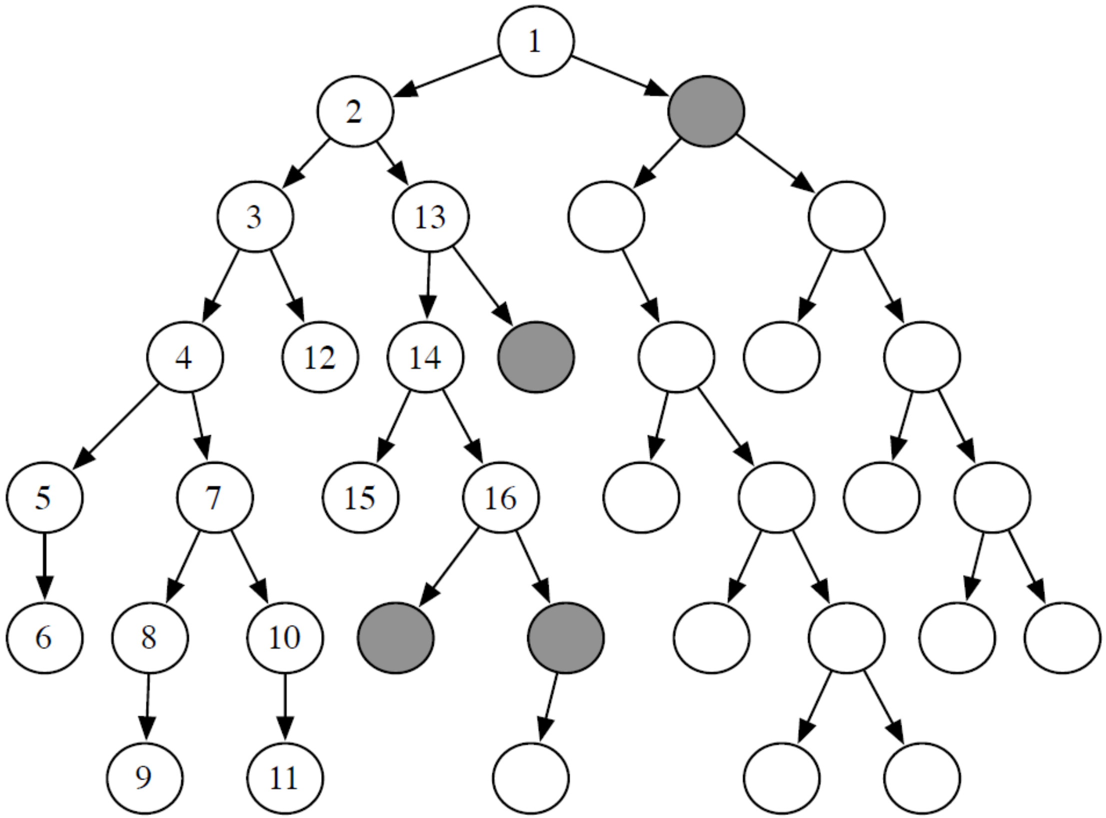
add $<n_o,.....,n_k, n>$ to frontier;

end

Today's Lecture

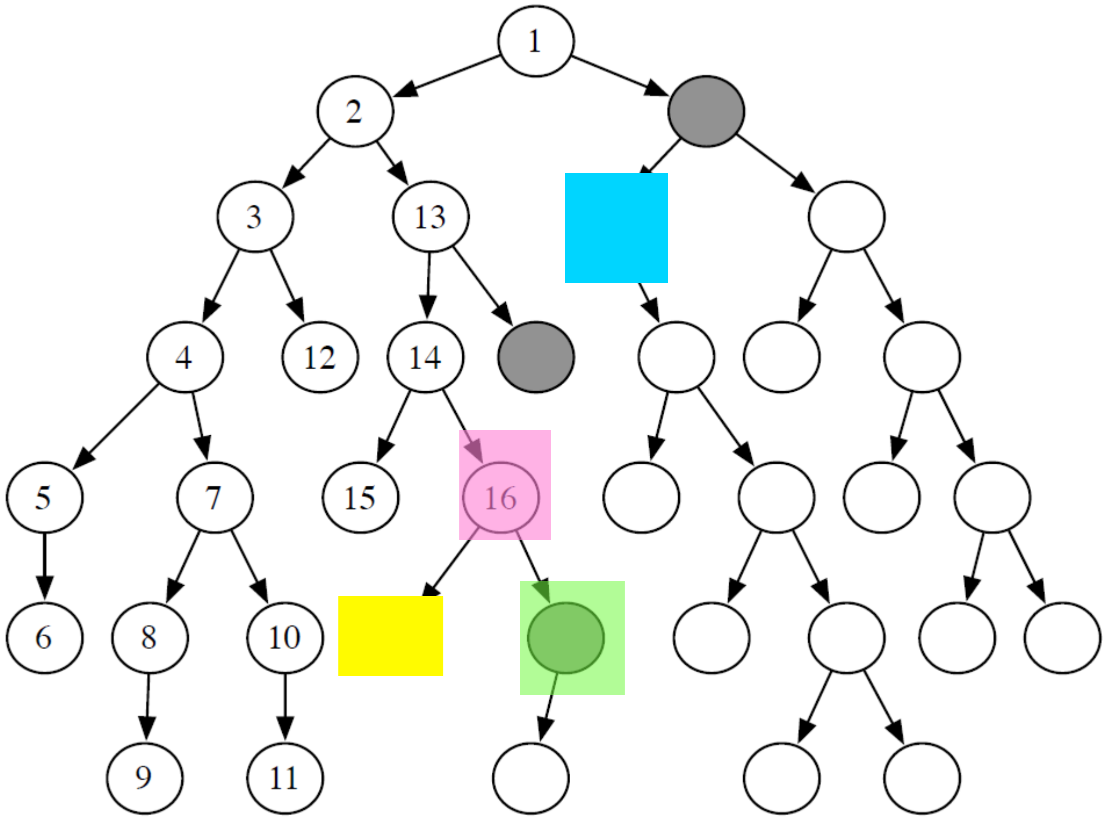
- Lecture 4 Recap
- Uninformed search + criteria to compare search algorithms
  - Depth-first
  - Breadth-first

Depth-first search (DFS)



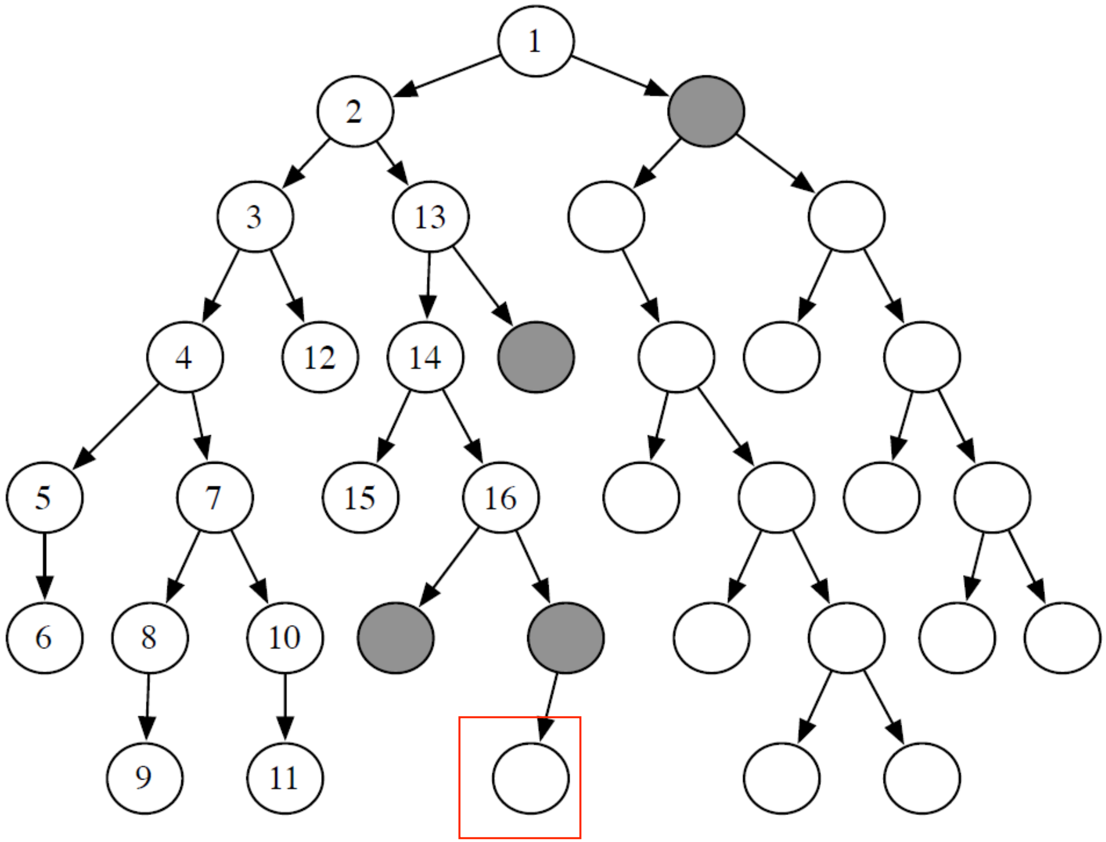
- Frontier: shaded nodes

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")

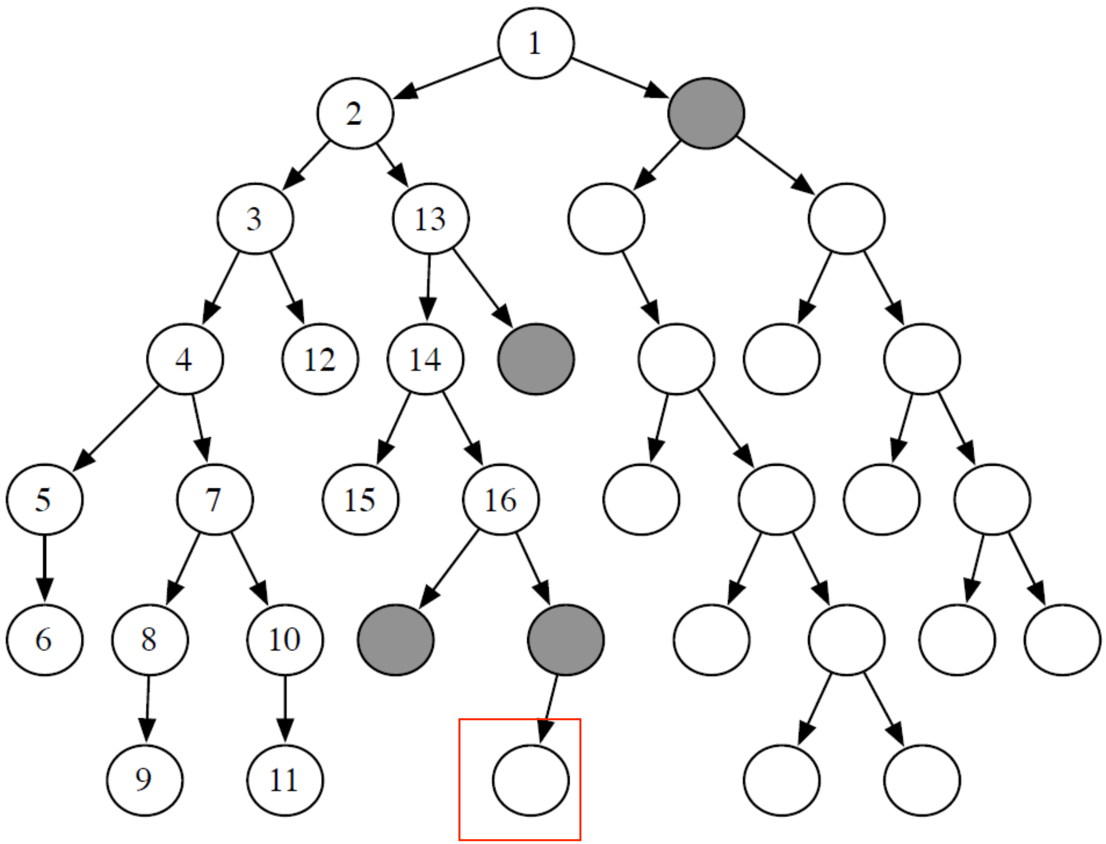
Depth-first search (DFS)



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

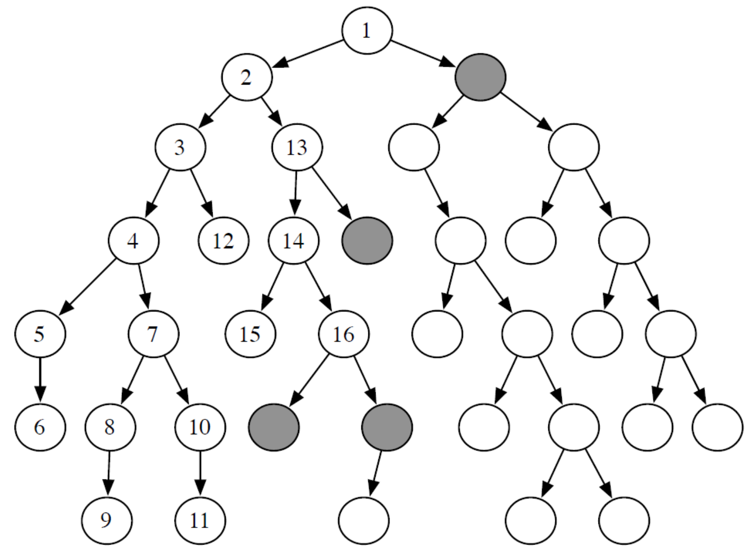
1 2 3 4

Depth-first search (DFS)



- Say, node in red box is a goal
- How many more nodes will be expanded?
  - 3: you only return once the goal is being expanded!
  - Not when a goal is put onto the frontier!

DFS as an instantiation of the Generic Search Algorithm



Input: a graph

a set of start nodes

Boolean procedure goal(n)

testing if n is a goal node

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

While frontier is not empty:

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

If goal(n_k)

Then return $<n_o,....,n_k>$;

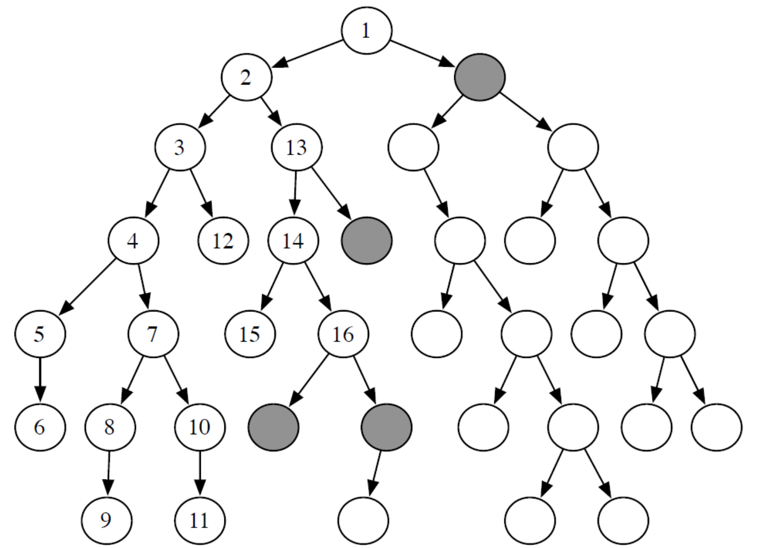
Else

For every neighbor n of n_k,

add $<n_o,....,n_k, n>$ to frontier;

end

DFS as an instantiation of the Generic Search Algorithm



Input: a graph

a set of start nodes

Boolean procedure goal(n) testing 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)

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

Else

For every neighbor n of n_k,

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

end

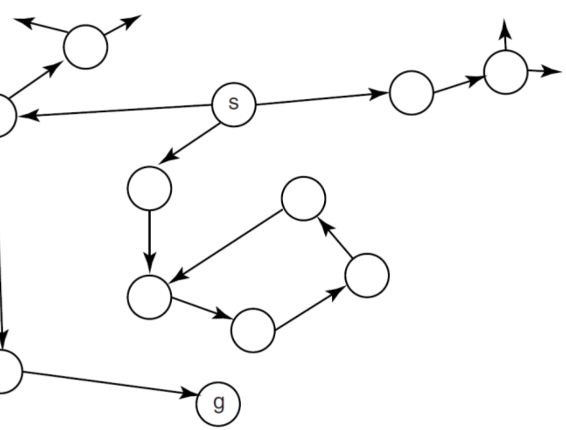
In DFS, the frontier is a last-in-first-out stack

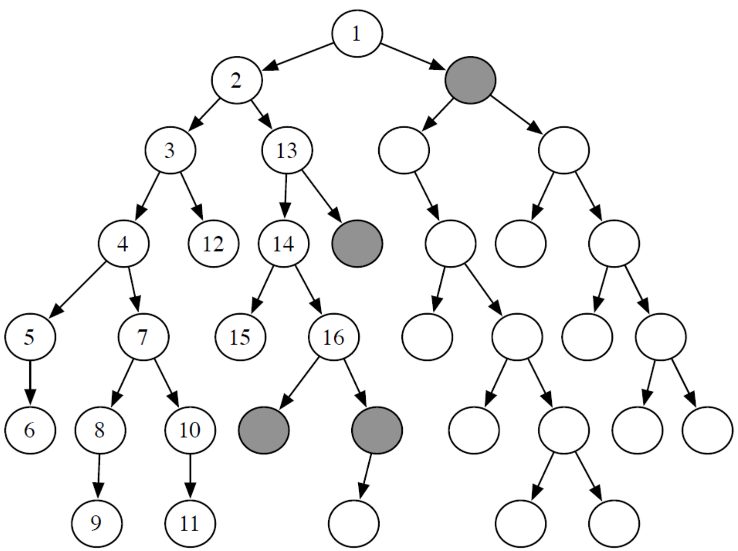
Analysis of DFS

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.

Is DFS complete? Yes No



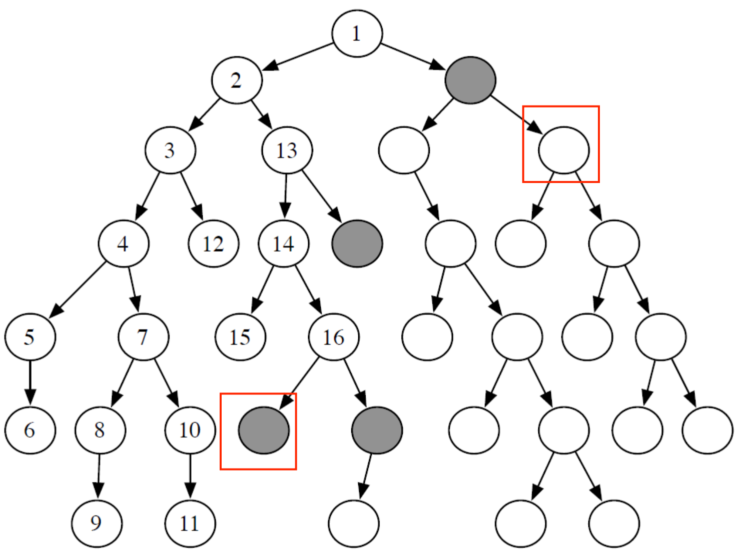


Analysis of DFS

Def.: A search algorithm is optimal if
when it finds a solution, it is the best one

Is DFS optimal? Yes No

- E.g., goal nodes: red boxes



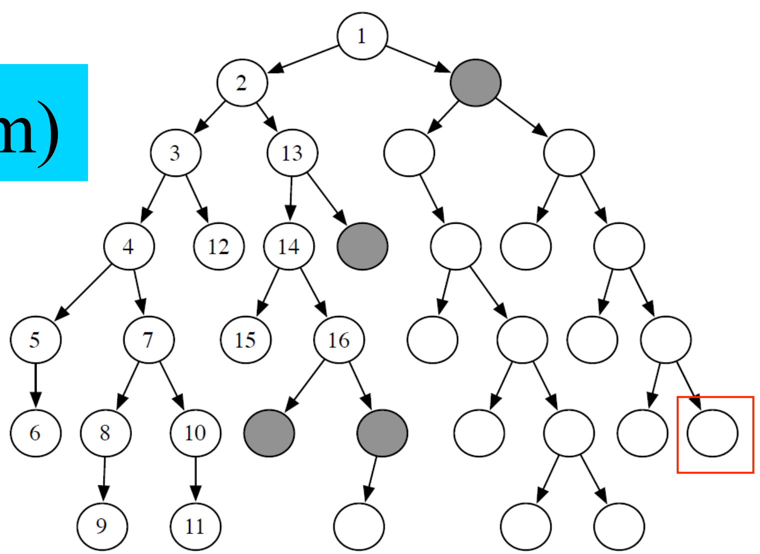
Analysis of DFS

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*.

- 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



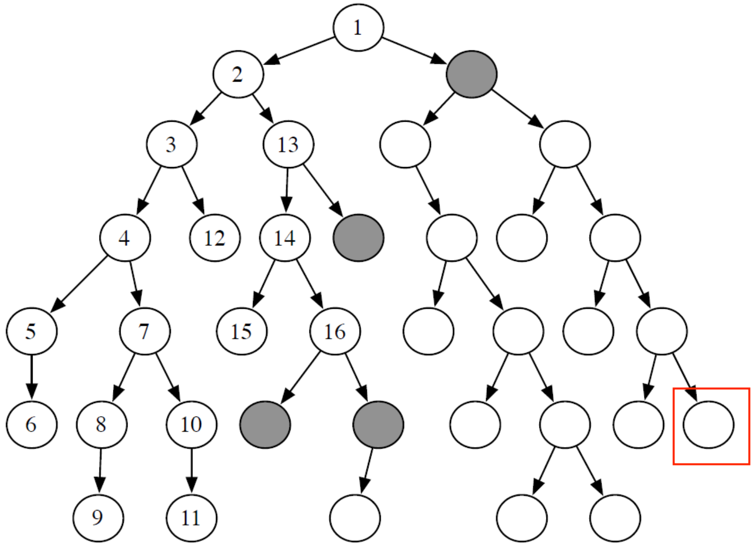
Analysis of DFS

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*.

- What is DFS's time complexity, in terms of m and b ?

$O(b^m)$

- E.g., single goal node: red box

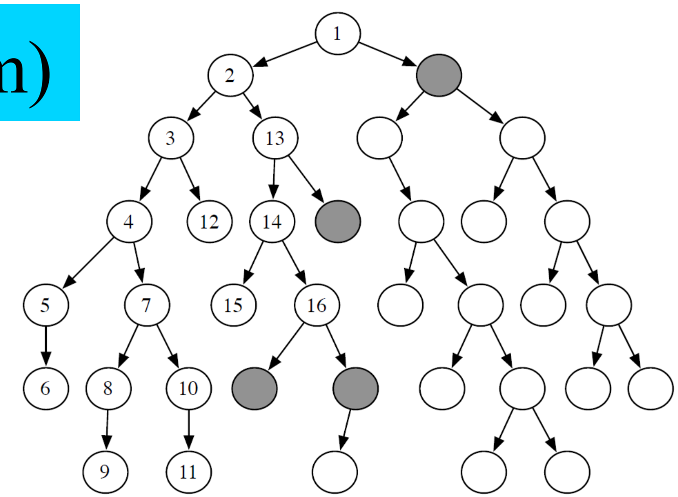


Analysis of DFS

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*.

- What is DFS's space complexity, in terms of m and b ?

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



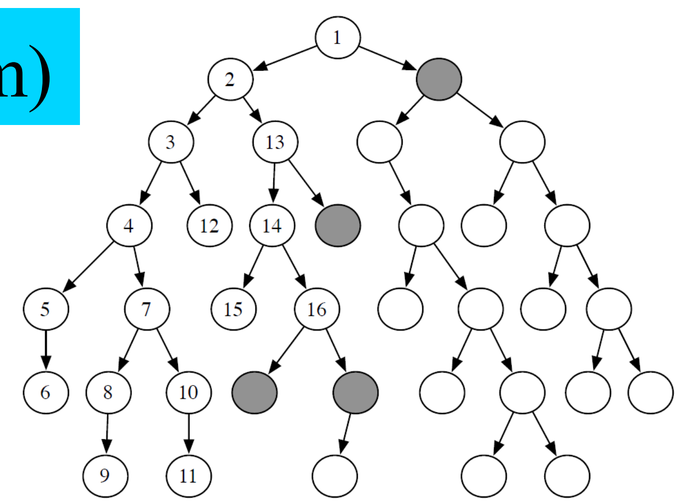
Analysis of DFS

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*.

- What is DFS's space complexity, in terms of m and b ?

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

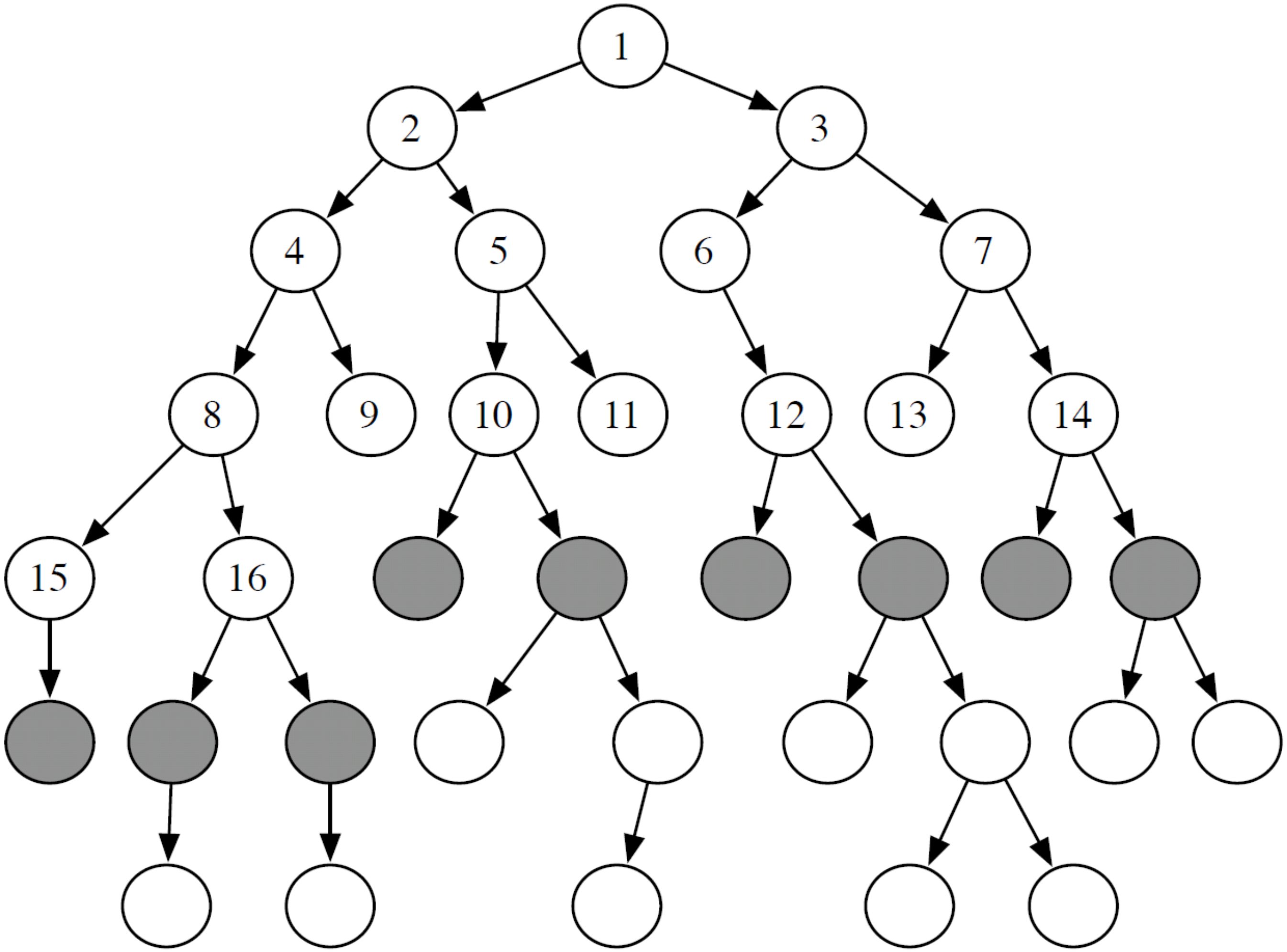


- O(bm)
- The longest possible path is m, and for every node in that path must maintain a fringe of size b

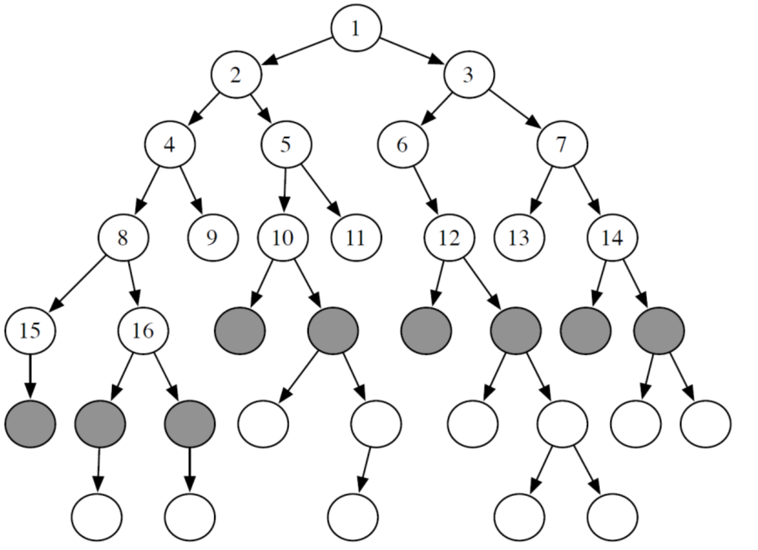
Today's Lecture

- Lecture 4 Recap
- Uninformed search + criteria to compare search algorithms
  - Depth first
  - Breadth first

Breadth-first search (BFS)



BFS as an instantiation of the Generic Search Algorithm



Input: a graph

a set of start nodes

Boolean procedure goal(n)
testing 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)

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

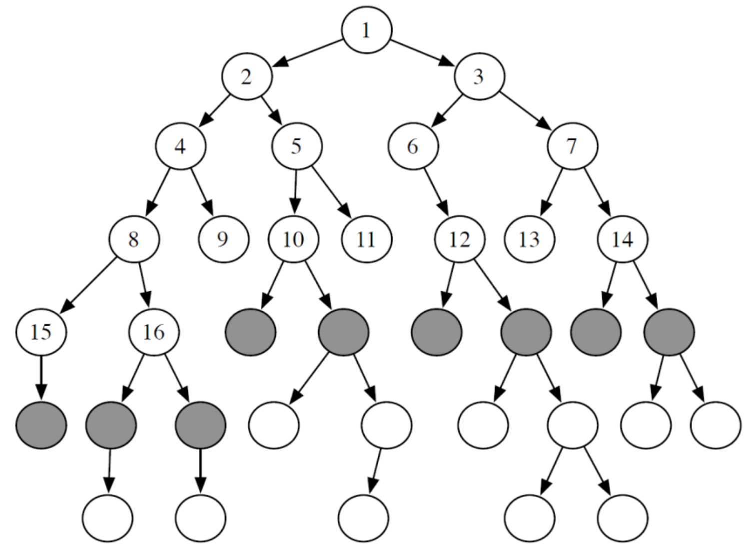
Else

For every neighbor n of n_k,

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

end

BFS as an instantiation of the Generic Search Algorithm



Input: a graph
a set of start nodes
Boolean procedure goal(n)
testing if n is a goal node

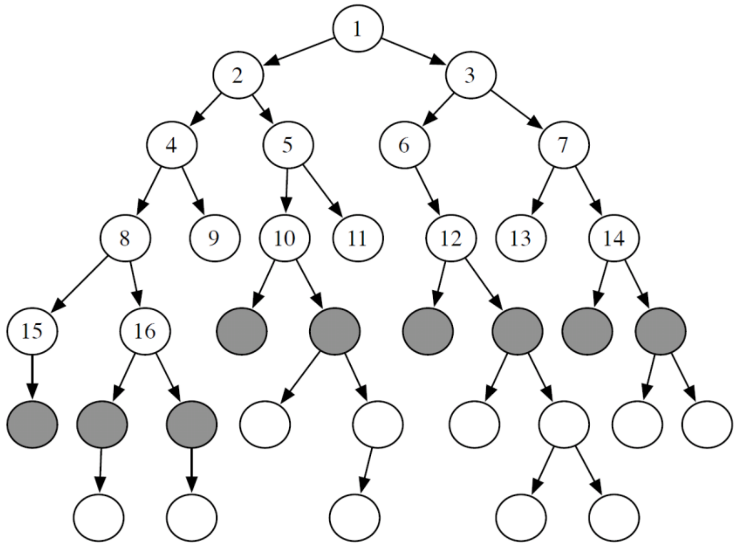
frontier:= [<s>: s is a start node];
While frontier is not empty:
select and **remove** path <$n_o, \ldots, n_k$> from frontier;
If goal(n_k)
Then return <$n_o, \ldots, n_k$>;
Else
For every neighbor n of n_k,
add <$n_o, \ldots, n_k$, n> to frontier;
end

In BFS, the frontier is a first-in-first-out queue

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.

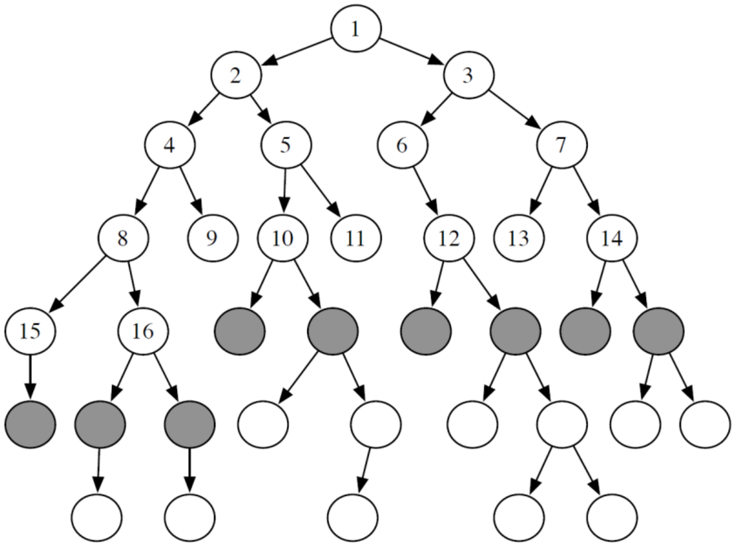
Is BFS complete? Yes No



Analysis of BFS

Def. : A search algorithm is 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



- Proof sketch?

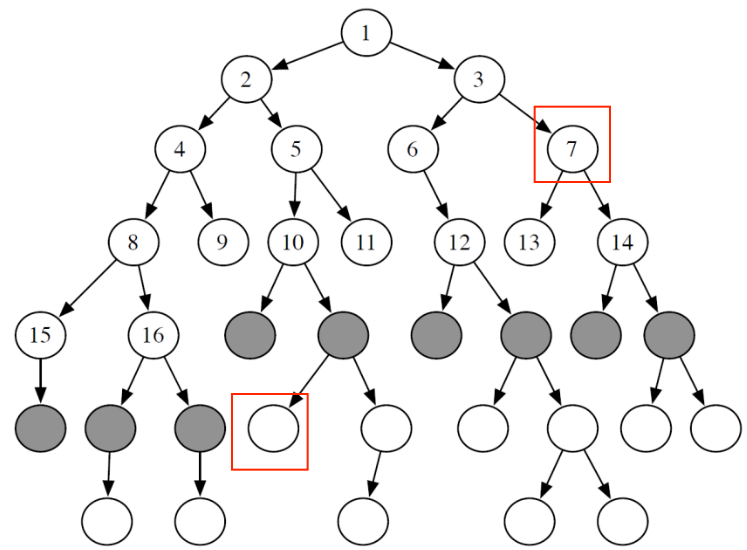
Analysis of BFS

Def.: A search algorithm is optimal if
when it finds a solution, it is the best one

Is BFS optimal?

Yes

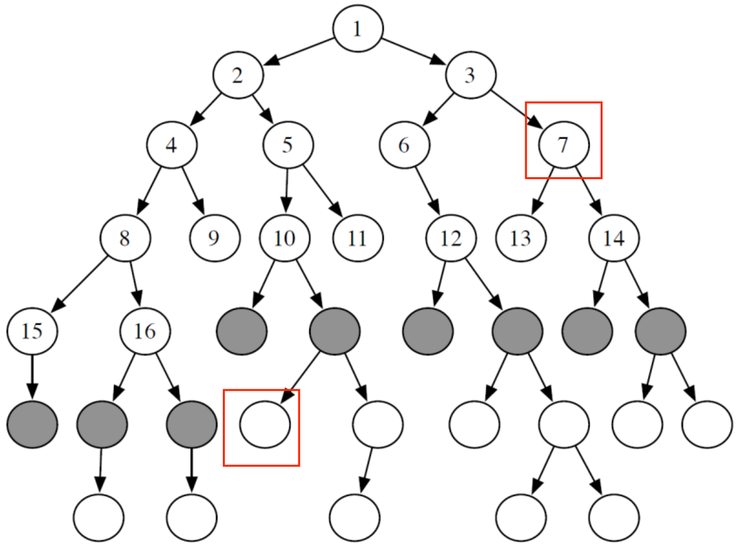
No



Analysis of BFS

Def.: A search algorithm is optimal if
when it finds a solution, it is the best one

Is BFS optimal? **Yes**



- Proof sketch?

Analysis of BFS

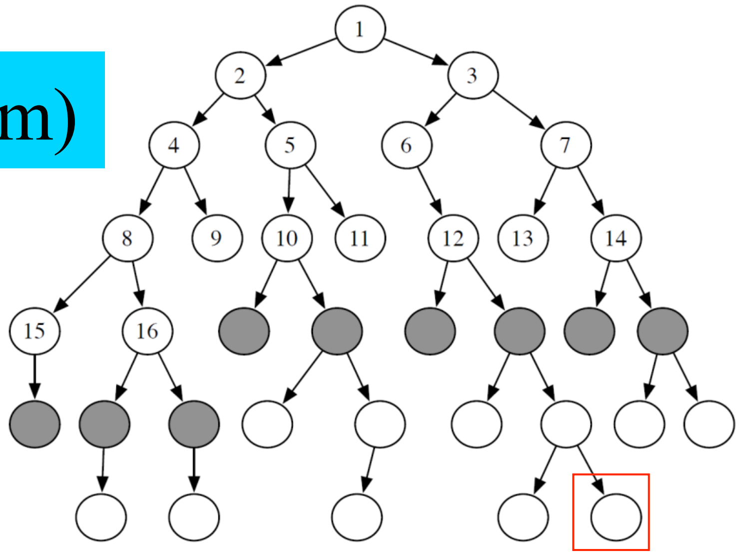
Def.: The time complexity of a search algorithm is the worst-case amount of time it will take to run, expressed in terms of

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

- 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

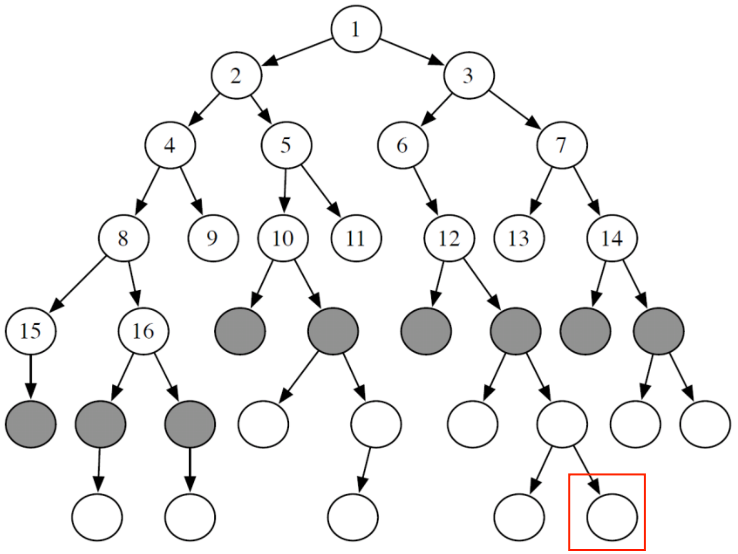
Def.: The time complexity of a search algorithm is the worst-case amount of time it will take to run, expressed in terms of

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

- What is BFS's time complexity, in terms of m and b ?

$O(b^m)$

- E.g., single goal node: red box

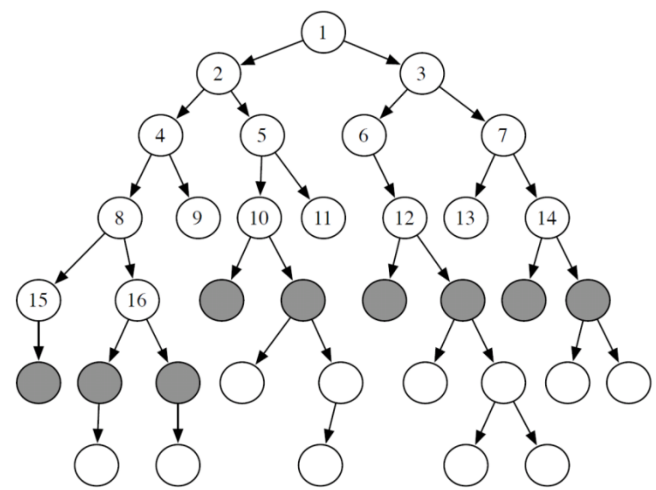


Analysis of BFS

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.

- 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

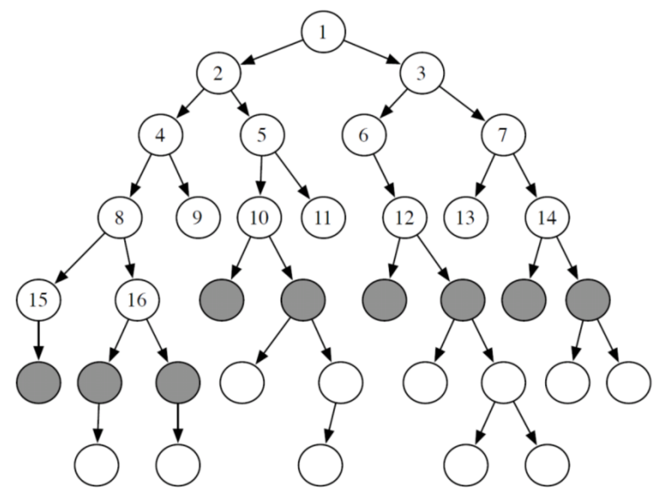
Def.: The space complexity of a search algorithm is the worst case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of
- maximum path length m
- maximum forward branching factor b.

- What is BFS's space complexity, in terms of m and b ?

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

$O(b^m)$

- How many nodes at depth m?



When to use BFS vs. DFS?

- The search graph has cycles or is infinite

  BFS DFS

- We need the shortest path to a solution

  BFS DFS

- There are only solutions at great depth

  BFS DFS

- There are some solutions at shallow depth: the other one

- No way the search graph will fit into memory

  BFS DFS

Real Example: Solving Sudoku

Sudoku Puzzle

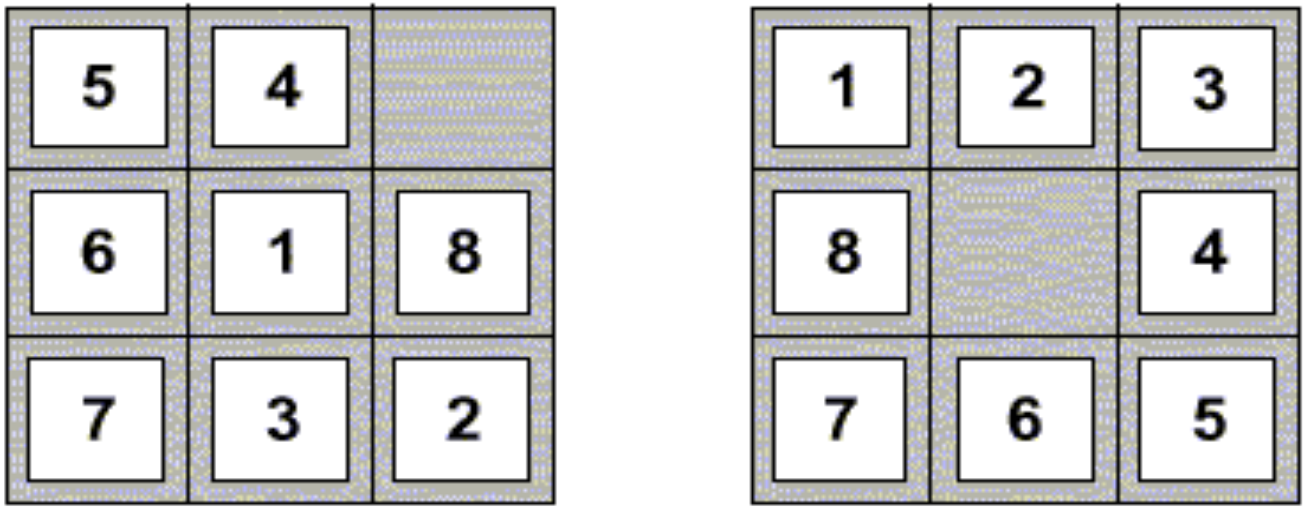
	9	3	6	2	8	1	4	
	6						5	
	3			1			9	
	5		8		2		7	
	4			7			6	
	8						3	
	1	7	5	9	3	4	2	

- E.g. start state on the left
- Operators:
  fill in an allowed number
- Solution: all numbers filled in, with constraints satisfied

- Which method would you rather use?

BFS DFS

Real Example: Eight Puzzle. DFS or BFS?



- Which method would you rather use?

BFS DFS

Learning Goals for today's class

- Apply basic properties of search algorithms:
  - completeness
  - optimality
  - time and space complexity of search algorithms
- Select the most appropriate search algorithms for specific problems.
  - Depth-First Search vs. Breadth-First Search

Coming up …

- Read Section 3.6, Heuristic Search