Lecture 4

Uninformed Search Strategies, Search with Costs

(Ch 3.1-3.5, 3.7.3)

Announcements

- Assignment 1 out this week
- Use first edition of the textbook (2010), but feel free to check corresponding sections on the new edition

Today's Lecture

- Recap from Previous lectures

- Depth first search
- Breadth first search
- Iterative deepening
- Search with costs
- Intro to heuristic search (time permitting)

Bogus Version of Generic Search Algorithm

Input: a graph a set of

start nodes

Boolean procedure goal(n)that tests if nis a goal node

frontier:= [<g>: gis 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 nof n_k

add n to frontier; end

- How many bugs?

A. One

B. Two

C.

Three D. Four

Bogus Version of Generic Search Algorithm

Input: a graph a set of

start nodes

Boolean procedure goal(n)that tests if nis a goal node

frontier:= [<g>: gis 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 nof n_k

add n to frontier; end

- Start at the start node(s), NOT at the goal

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

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

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.

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

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 branching factor b.

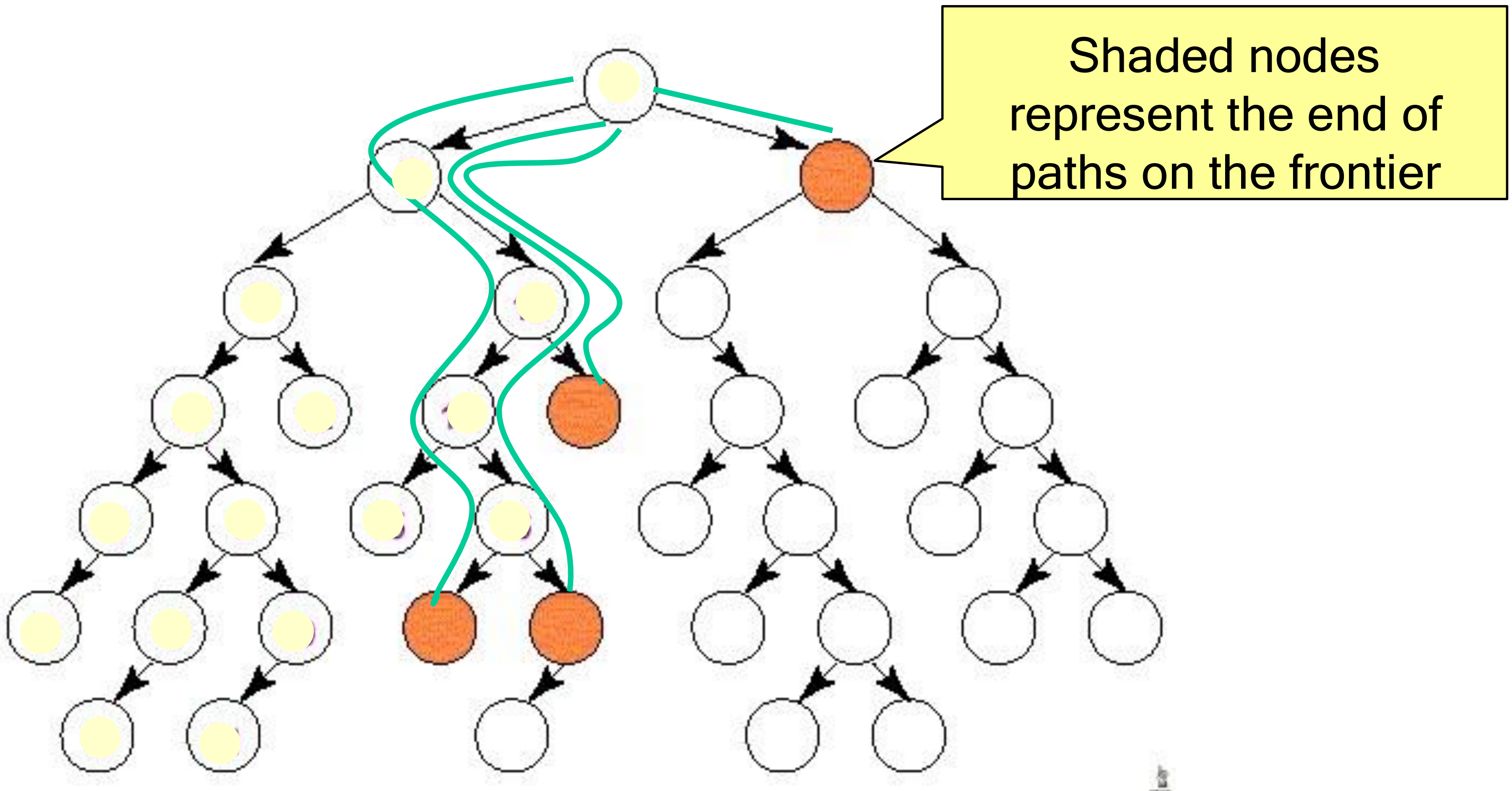
Def.: The space complexity of a search algorithm is the worst-case amount of memory that the algorithm will use (i.e., the maximum number of nodes on the frontier), also expressed in terms of m and b. Slide 6

Today's Lecture

- Recap from Previous lectures
- Depth first search
- Breadth first search
- Iterative deepening

- Search with costs
- Intro to heuristic search (time permitting)

Illustrative Graph: DFS



DFS explores each path on the frontier until its end (or until a goal is found) before considering any other path.

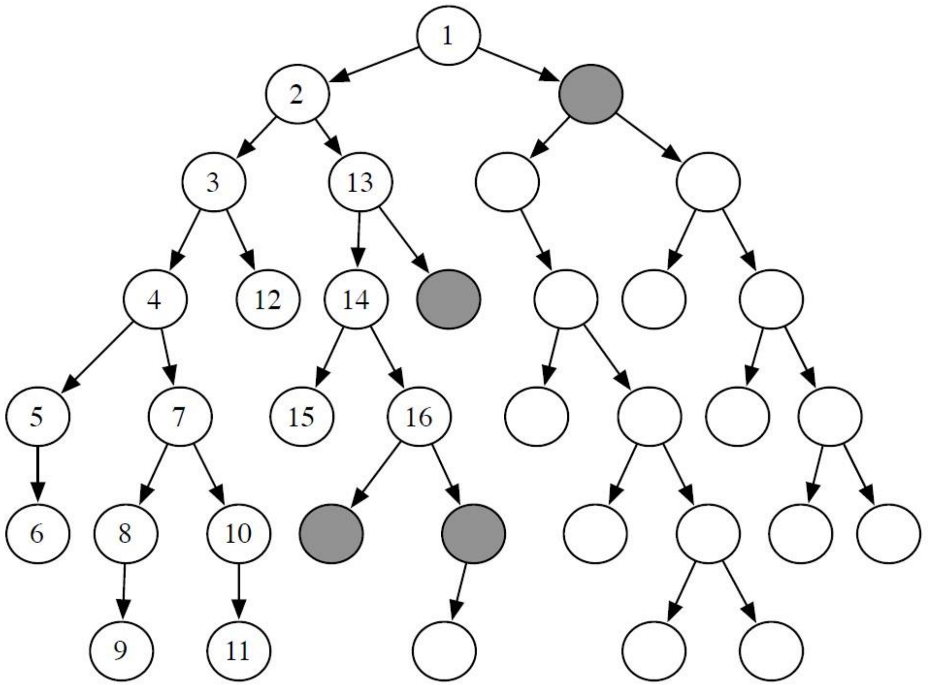
Input: a graph a set
of start
nodes
Boolean procedure goal(n)
testing if n is a goal node

frontier:= [<s>: s is a start node]; While frontier is not empty:

select and remove path $\langle n_o,\ldots,n_k\rangle$ from frontier;

If goal(n_k)

return $\langle n_o,\ldots,n_k\rangle$;



For every neighbor n of $n_{k,}$

add <n_o,….,n_k, n> to

frontier;

end

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

DFS as an instantiation of the Generic Search Algorithm

Let's see how this works in

DFS in AI Space

- Go to: http://www.aispace.org/mainTools.shtml
- Click on “Graph Searching” to get to the Search Applet
- Select the “Solve” tab in the applet
- Select one of the available examples via “File -> Load Sample Problem (good idea to start with the “Simple Tree” problem)
- Make sure that “Search Options -> Search Algorithms” in the toolbar is set to “Depth-First Search”.
- Step through the algorithm with the “Fine Step” or

“Step” buttons in the toolbar • The panel above the graph panel verbally describes what is happening during each step

- The panel at the bottom shows how the frontier evolves

See available help pages and video tutorials for more details on how to use the Search applet (http://www.aispace.org/search/index.shtml)

Depth-first Search: DFS

Example:

- the frontier is $[p_1, p_2, \ldots, p_r]$ -each p_k is a path
- neighbors of last node of p_1 are $\{n_1, \ldots, n_k\}$
- What happens?
- p_1 is selected, and its last node is tested for being a goal. If not

Analysis of DFS

- Knew paths are created by adding each of $\{n_1, \ldots, n_k\}$ to p_1 • These Knew paths replace p_1 at the beginning of the frontier.
- Thus, the frontier is now: $[(p_1, n_1), \ldots, (p_1, n_k), p_2, \ldots, p_r]$.

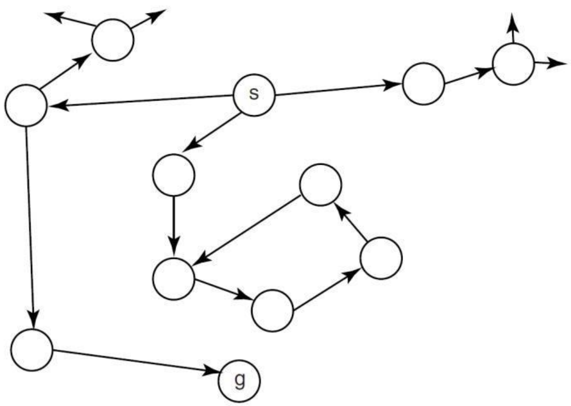
NOTE: p_2 is only selected when all paths extending p_1 have been explored.

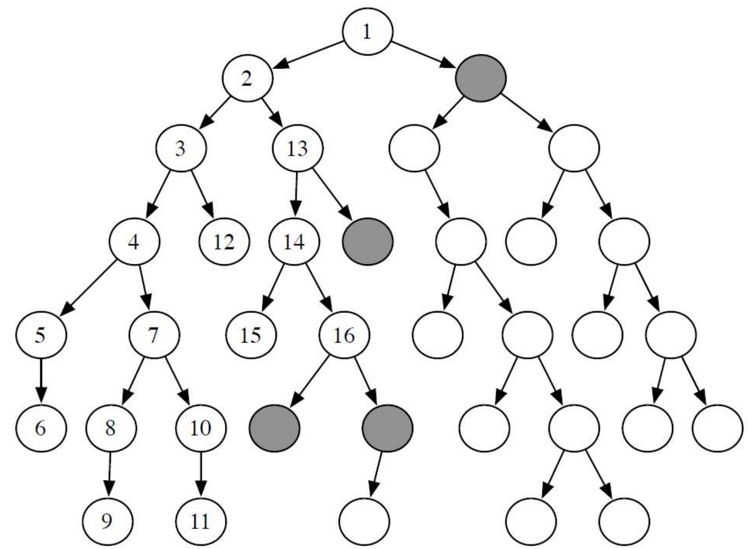
- You can get a much better sense of how DFS works by looking at the Search Applet in AI Space

Slide 11

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?

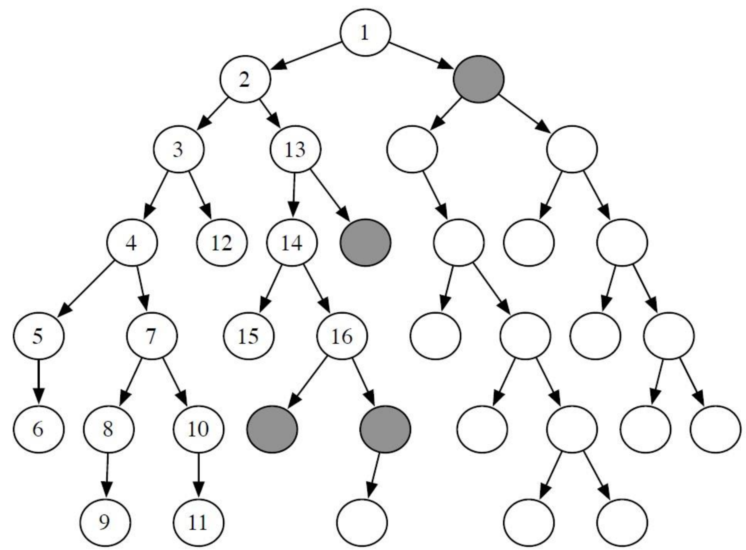




Analysis of DFS

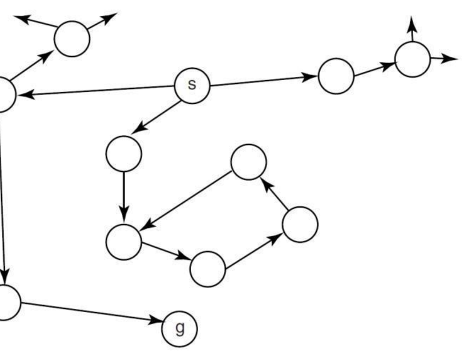
Analysis of DFS

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.



Is DFS complete?

No

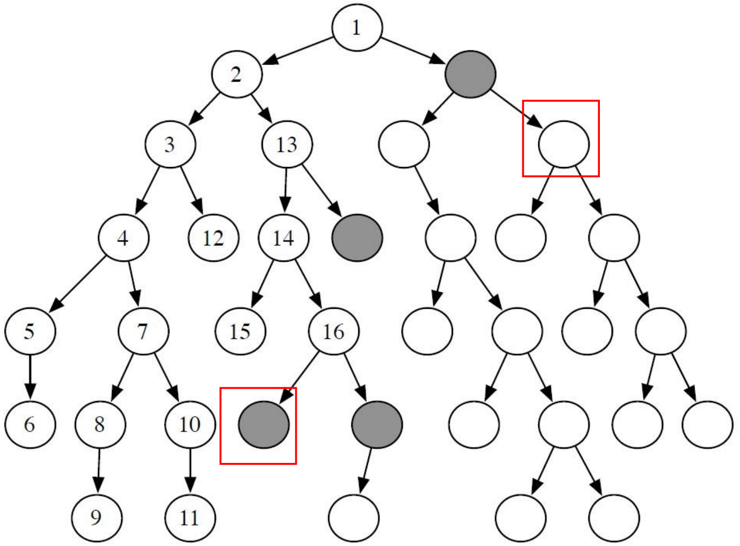


- If there are cycles in the graph, DFS may get “stuck” in one of them
- see this in AISpace by loading “Cyclic Graph Examples” or by adding a cycle to “Simple Tree”
- e.g., click on “Create” tab, create a new edge from N7 to N1, go back to “Solve” and see what happens

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

Analysis of DFS

Is DFS optimal?



15

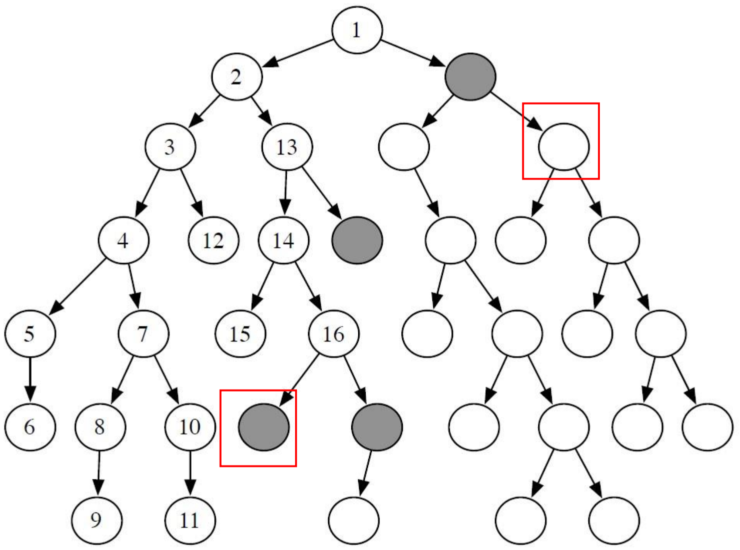
- E.g., goal nodes: red boxes

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

Is DFS optimal?

A Yes B No

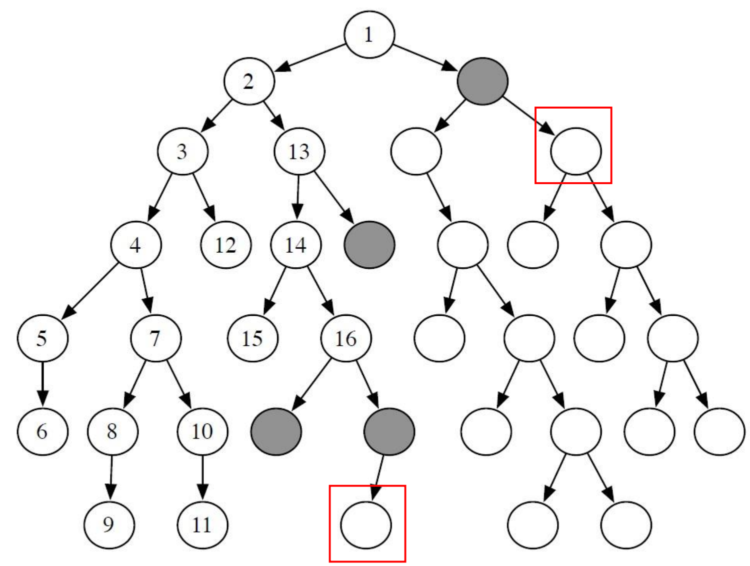
- E.g., goal nodes: red boxes



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
- see this in AISpace by loading "Extended Tree Graph" and set N6 as a goal
- e.g., click on "Create" tab, right-click on N6 and select "set as a goal node"

Analysis of DFS

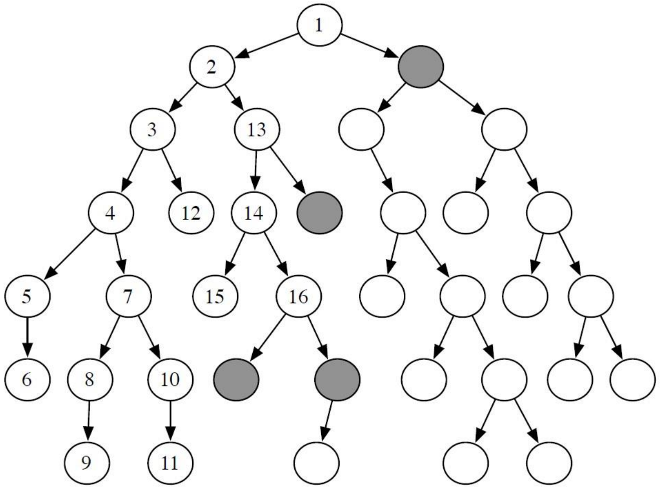
Def.: The space complexity of a search algorithm is the worst-case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

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

Analysis of DFS

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

See how this works in



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

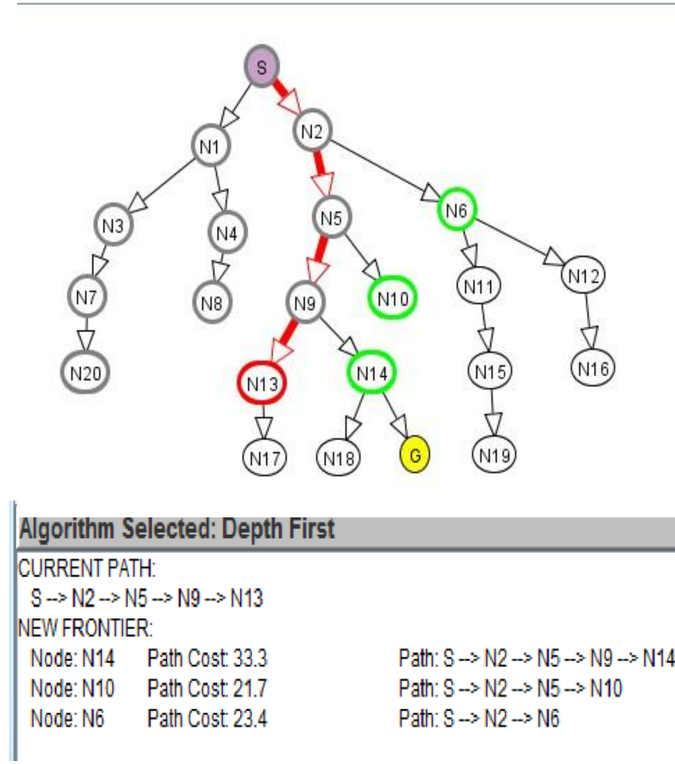
- maximum path length m
- maximum forward branching factor b.

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

Analysis of DFS

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 along the path



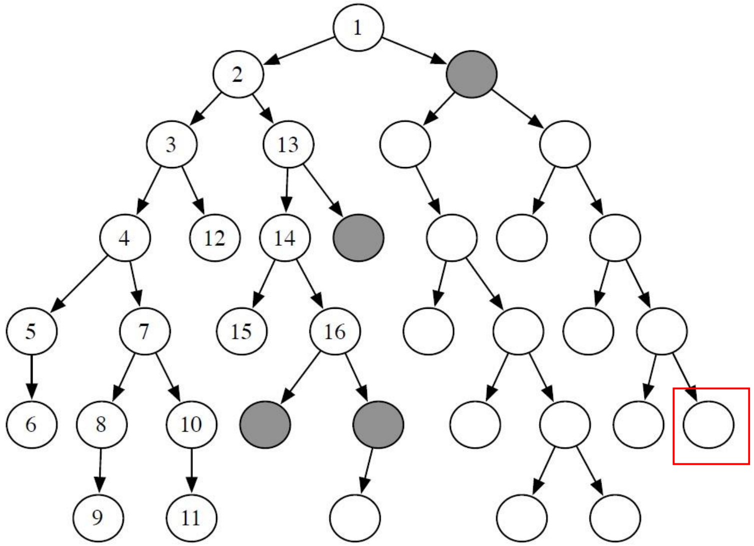
Analysis of DFS

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?

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

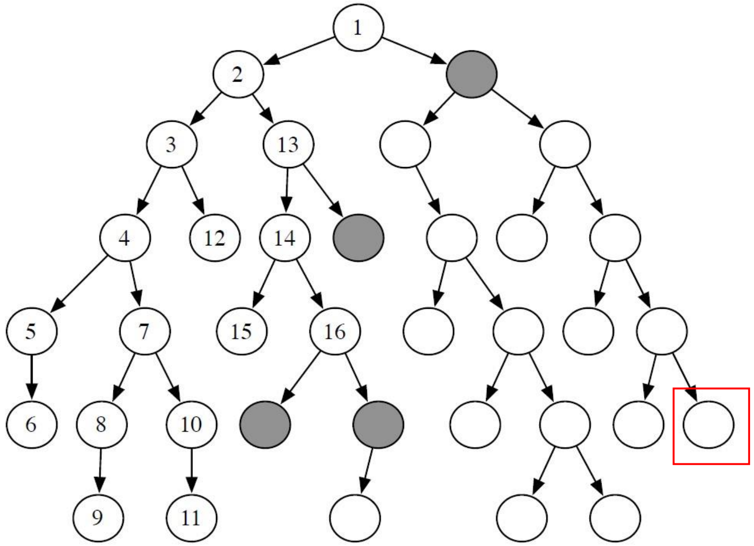


Analysis of DFS

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?



A. $O(b^m)$

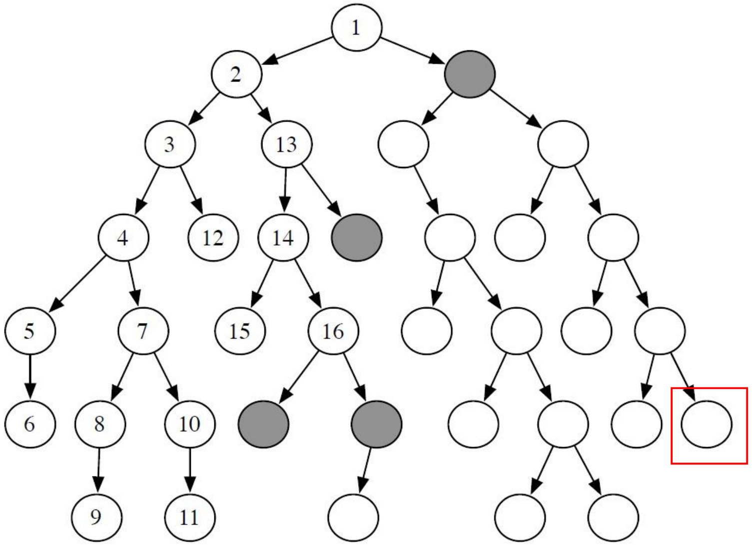
B. $O(m^b)$

C. $O(bm)$

D. $O(b+m)$

- E.g., single goal node -> red box
- Hint: think about how many nodes are in a search tree m steps away from the start

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.

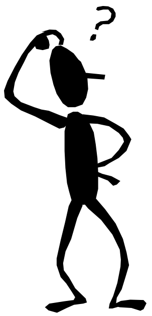
- What is DFS's time complexity, in terms of m and b?

$O(b^m)$

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

To Summarize

	Complete	Optimal	Time	Space
DFS	NO	NO	$O(b^m)$	$O(bm)$



Analysis of DFS: Summary

- Is DFS complete? NO
- Depth-first search isn't guaranteed to halt on graphs with cycles.
- However, DFS iscomplete for finite acyclic graphs.
- Is DFS optimal? NO
- It can “stumble” on longer solution paths before it gets to shorter ones.
- What is the worst-case time complexity, if the maximum path length is mand 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.
- What is the worst-case space complexity?
- O(bm)
- the longest possible path is m, and for every node in that path must maintain a fringe of size b. Slide

Analysis of DFS (cont.)

DFS is appropriate when

- Space is restricted
- Many solutions, with long path length

It is a poor method when

- There are cycles in the graph
- There are sparse solutions at shallow depth

Why DFS need to be studied and understood?

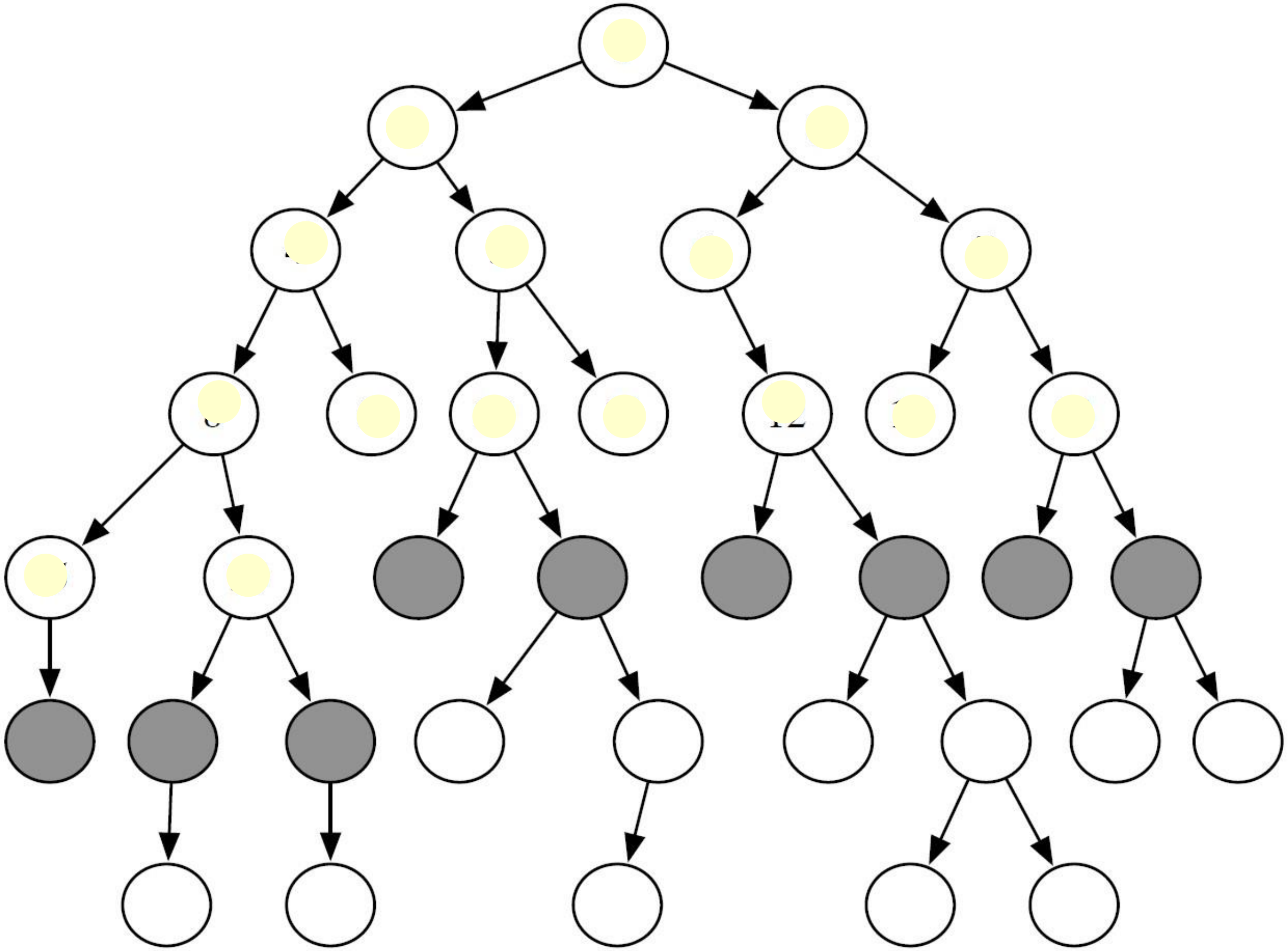
- It is simple enough to allow you to learn the basic aspects of searching

- It is the basis for a number of more sophisticated and useful search algorithms (e.g. Iterative Deepening)

Today’s Lecture

- Recap from Previous lectures
- Depth first search - analysis
- Breadth first search
- Iterative deepening
- Search with costs
- Intro to heuristic search (time permitting)

- BFS explores all paths of length l on the frontier, before looking at path of length l + 1



Breadth-first search (BFS)

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 $<n_o,\dots,n_k>$ from frontier;

If goal(n_k) return

$<n_o,\dots,n_k>$;

Else

For every neighbor n of n_k, add

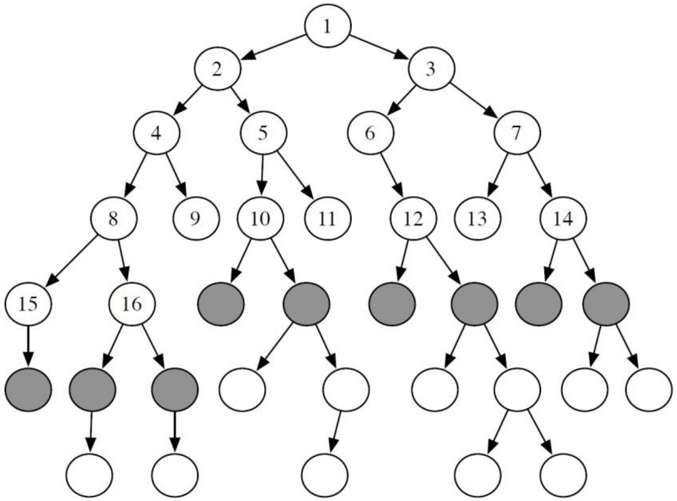
$<n_o,\dots,n_k, n>$ to frontier;

end

Let's see how this works in AIspace

in the Search Applet toolbar, set “Search Options -> Search Algorithms” to “Breadth-First Search”.

BFS as an instantiation of the Generic Search Algorithm



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

Breadth-first Search: BFS

Example:

- 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 its end node is tested for being a goal. If not
- New k paths are created attaching each of $\{n_1, \ldots, n_k\}$ to p_1
- These follow p_r at the end of the frontier.

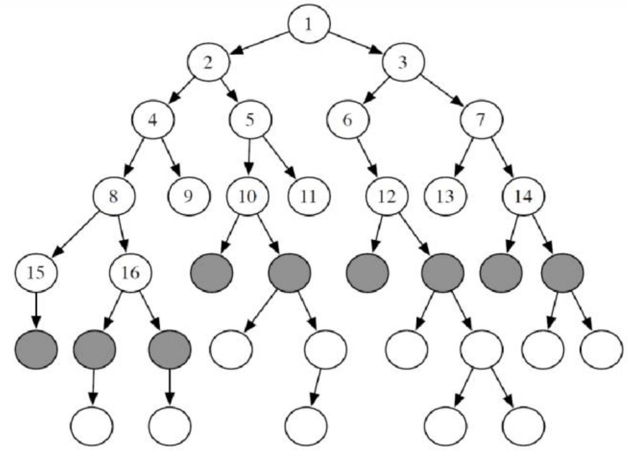
- Thus, the frontier is now $[p_2, \ldots, p_r, (p_1, n_1), \ldots, (p_1, n_k)]$.
- p_2 is selected next.

As for DFS, you can get a much better sense of how BFS works by looking at the Search Applet in AI Space

Slide

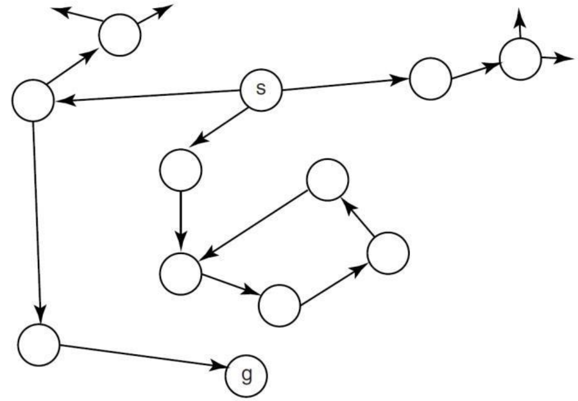
Analysis of BFS

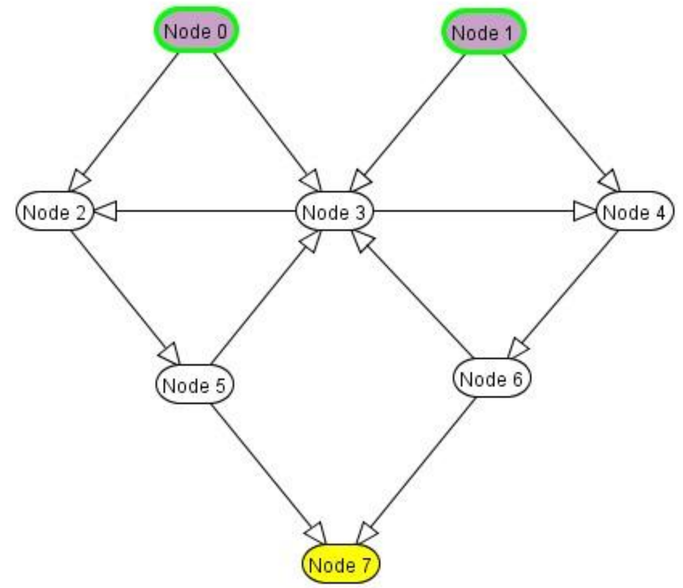
Def. : A search algorithm is complete if whenever there is at least one solution, the algorithm is guaranteed to find it within a finite amount of time.



Is BFS complete?

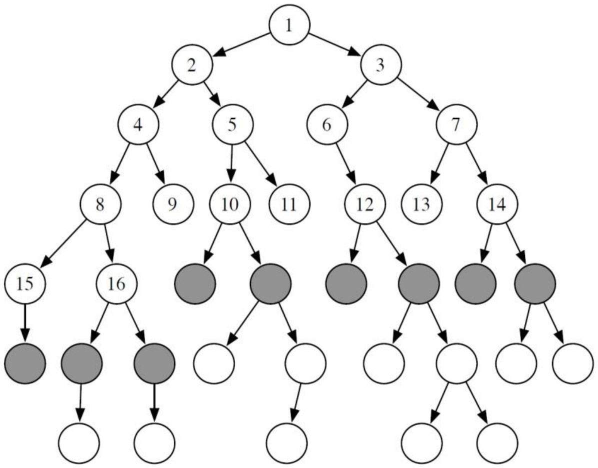
Analysis of BFS





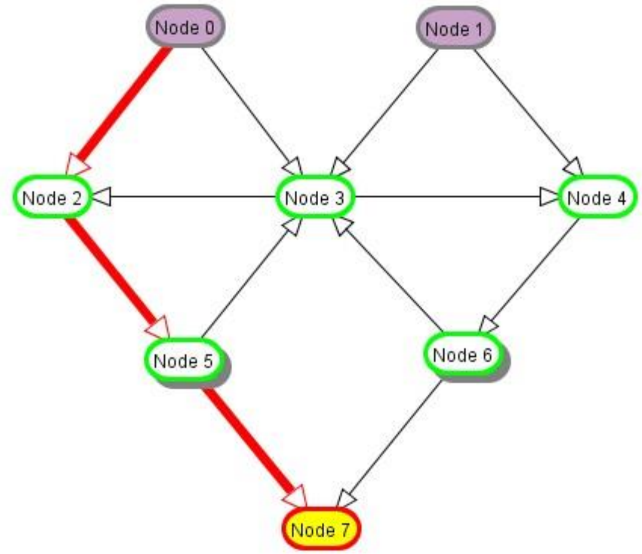
Analysis of BFS

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.



Analysis of BFS

Is BFS complete? Yes



- If a solution exists at level l, the path to it will be explored before any other path of length l + 1
- impossible to fall into an infinite cycle
- see this in AISpace by loading “Cyclic Graph Examples” or by adding a cycle to “Simple Tree”

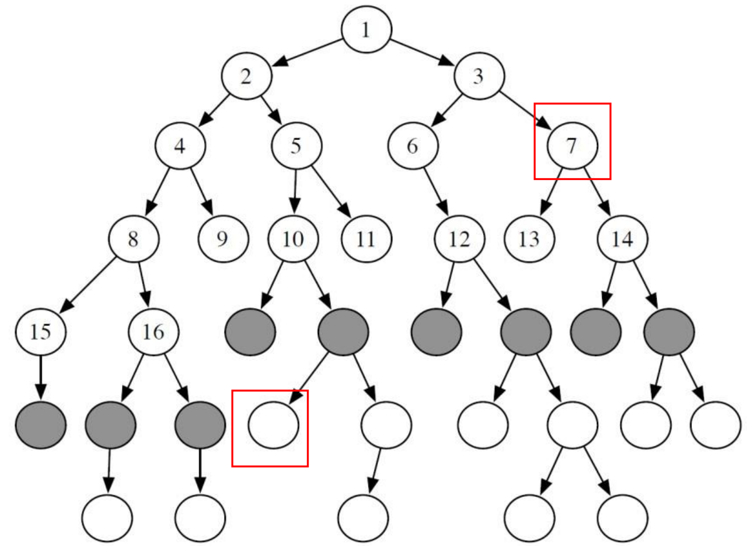
Def.: A search algorithm is optimal if when it finds a solution, it is the best one

Analysis of BFS

Is BFS optimal?

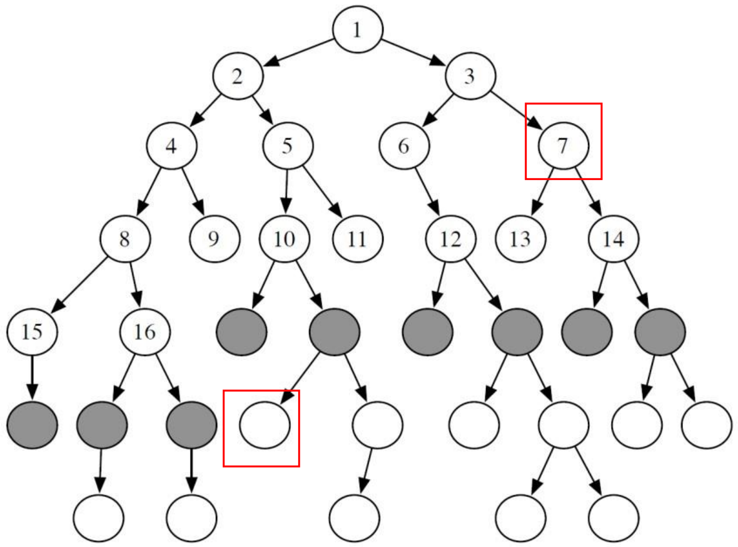
- E.g., two goal nodes: red boxes

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



Analysis of BFS

Is BFS optimal? Yes



- E.g., two goal nodes: red boxes
- Any goal at level l(e.g. red box N7) will be reached before goals at lower levels

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.

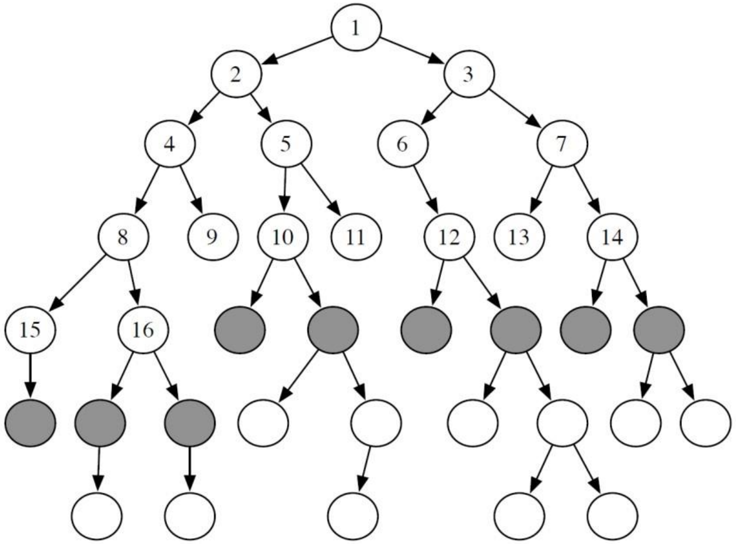
- What is BFS's time complexity, in terms of m and b ?

A. $O(b^m)$

B. $O(m^b)$

Analysis of BFS

C. O(bm)



D. O(b+m)

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

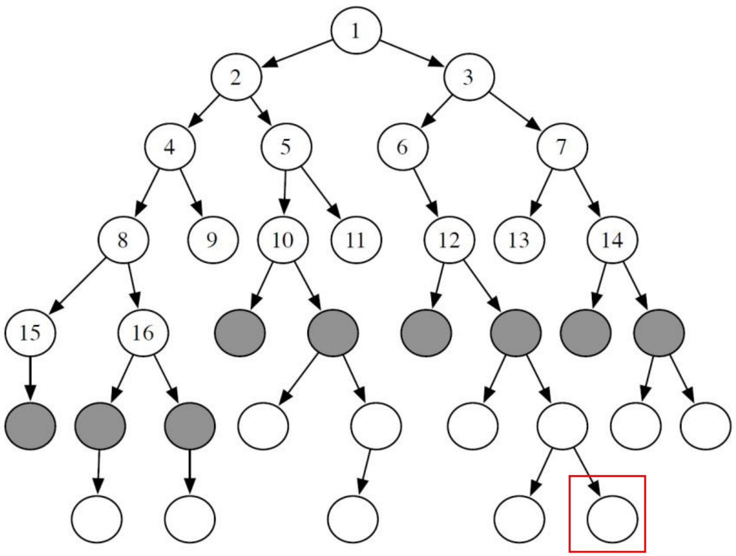
- maximum path length m
- maximum forward branching factor b.

$O(b^m)$

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

Analysis of BFS

- Like DFS, in the worst case BFS must examine every node in the tree
- E.g., single goal node -> red box

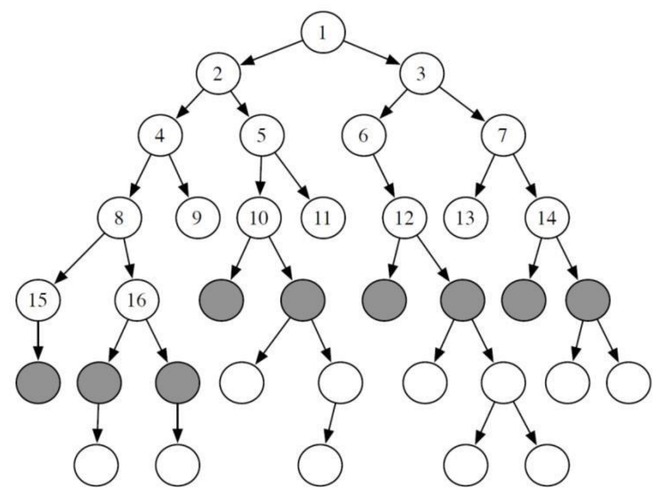


Analysis of BFS

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 ?



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.

A. $O(b^m)$

B. $O(m^b)$

C. $O(bm)$

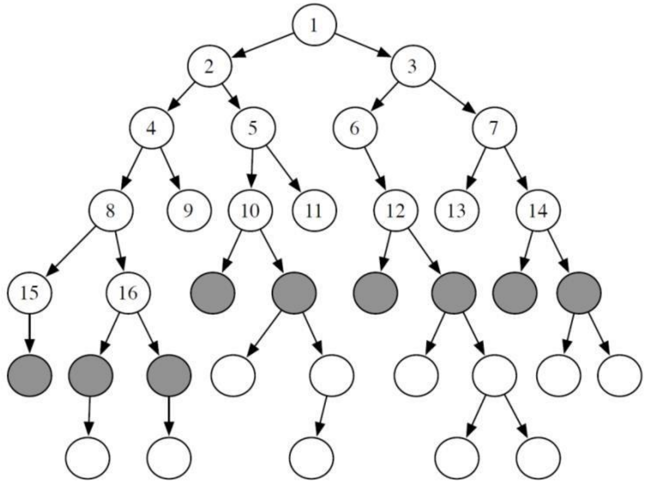
D. $O(b+m)$

Analysis of BFS

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

$O(b^m)$

- BFS must keep paths to all the nodes al level m



When to use BFS vs. DFS?

- The search graph has cycles or is infinite

- We need the shortest path to a solution

- There are only solutions at great depth

- There are some solutions at shallow depth

- No way the search graph will fit into memory

When to use BFS vs. DFS?

- The search graph has cycles or is infinite

BFS

- We need the shortest path to a solution

BFS

- There are only solutions at great depth

DFS

- There are some solutions at shallow depth

BFS

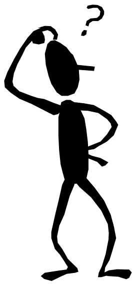
- No way the search graph will fit into memory

DFS

To Summarize

	Complete	Optimal	Time	Space
DFS	NO	NO	$O(b^m)$	$O(bm)$
BFS	YES	YES	$O(b^m)$	$O(b^m)$

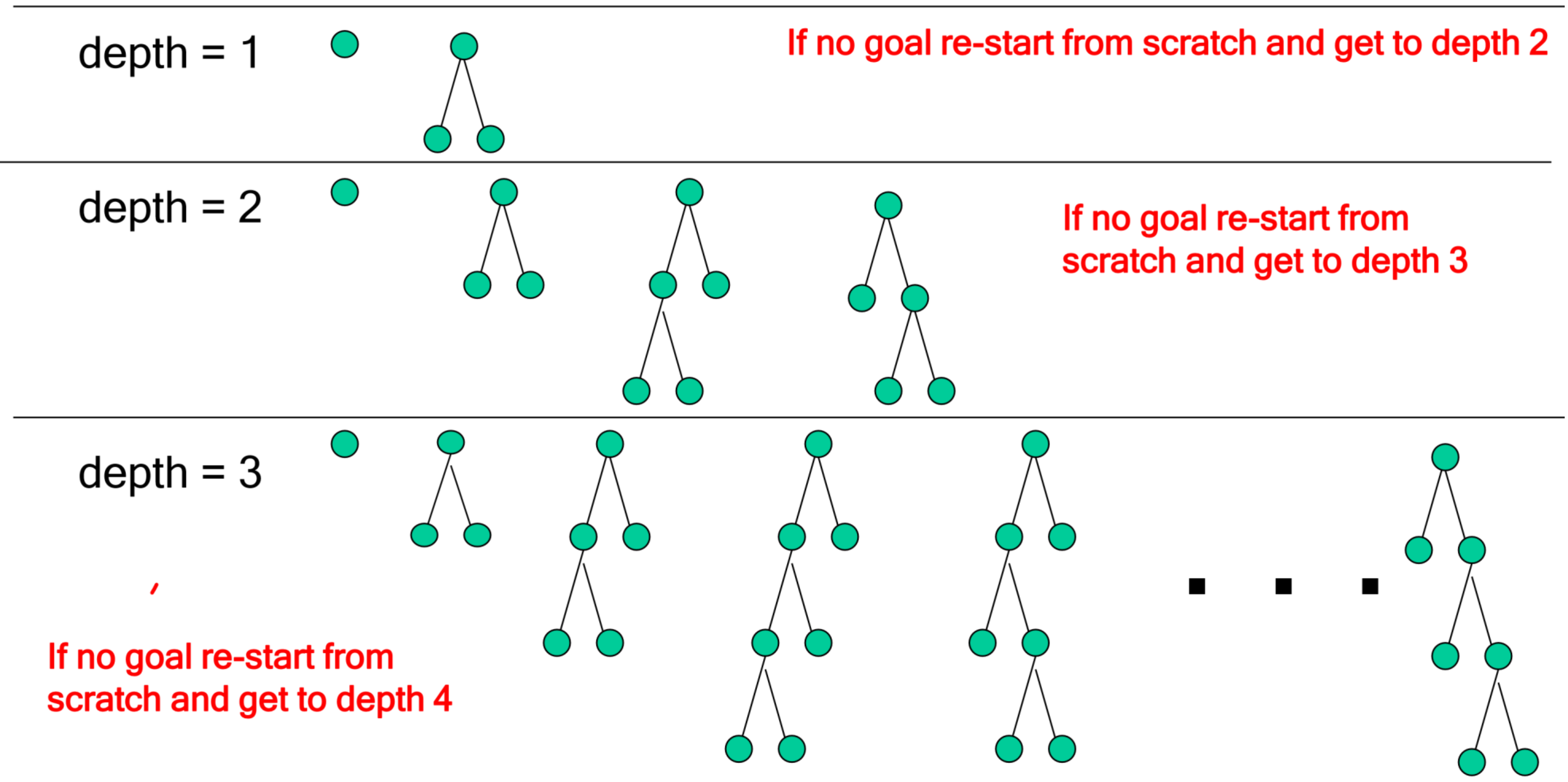
How can we achieve an acceptable (linear) space complexity while maintaining completeness and optimality?



Key Idea: re-compute elements of the frontier rather than saving them.

Iterative Deepening DFS (IDS) in a Nutshell

- Depth-bounded depth-first search



(Time) Complexity of IDS

- Use DFS to look for solutions at depth 1, then 2, then 3, etc
  - For depth D, ignore any paths with longer length

(Time) Complexity of IDS

- That sounds wasteful!
- Let's analyze the time complexity
- For a solution at depth m with branching factor b

(Time) Complexity of IDS

Depth	Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS	Total #paths for IDS
1				
2				
. . .	. . .	. . .	. . .	. . .
m-1				
m				

- That sounds wasteful!

(Time) Complexity of IDS

- Let's analyze the time complexity
- For a solution at depth mwith branching factor b

Depth	Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS	Total #paths for IDS
1	b	1	m	mb
2				
. . .				
m-1				
m				

(Time) Complexity of IDS

- That sounds wasteful!
- Let's analyze the time complexity
- For a solution at depth m with branching factor b

Depth	Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS	Total #paths for IDS
1	b	1	m	mb
2	b^2	1	m-1	$(m-1)\ b^2$
. . .				
m-1				
m				

(Time) Complexity of IDS

- That sounds wasteful!
- Let's analyze the time complexity
- For a solution at depth m with branching factor b

Depth	Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS	Total #paths for IDS
1	b	1	m	mb
2	b^2	1	m-1	$(m-1)\ b^2$
. . .	. . .	. . .	. . .	. . .
m-1	b^{m-1}	1	2	$2\ b^{m-1}$
m	b^m	1	1	b^m

(Time) Complexity of IDS

Solution at depth m, branching factor b

Total # of paths generated:

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

$= b^m\,(1\,b^0 + 2\,b^{-1} + 3\,b^{-2} + \ldots + m\,b^{1-m})$

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

Overhead factor

converges to

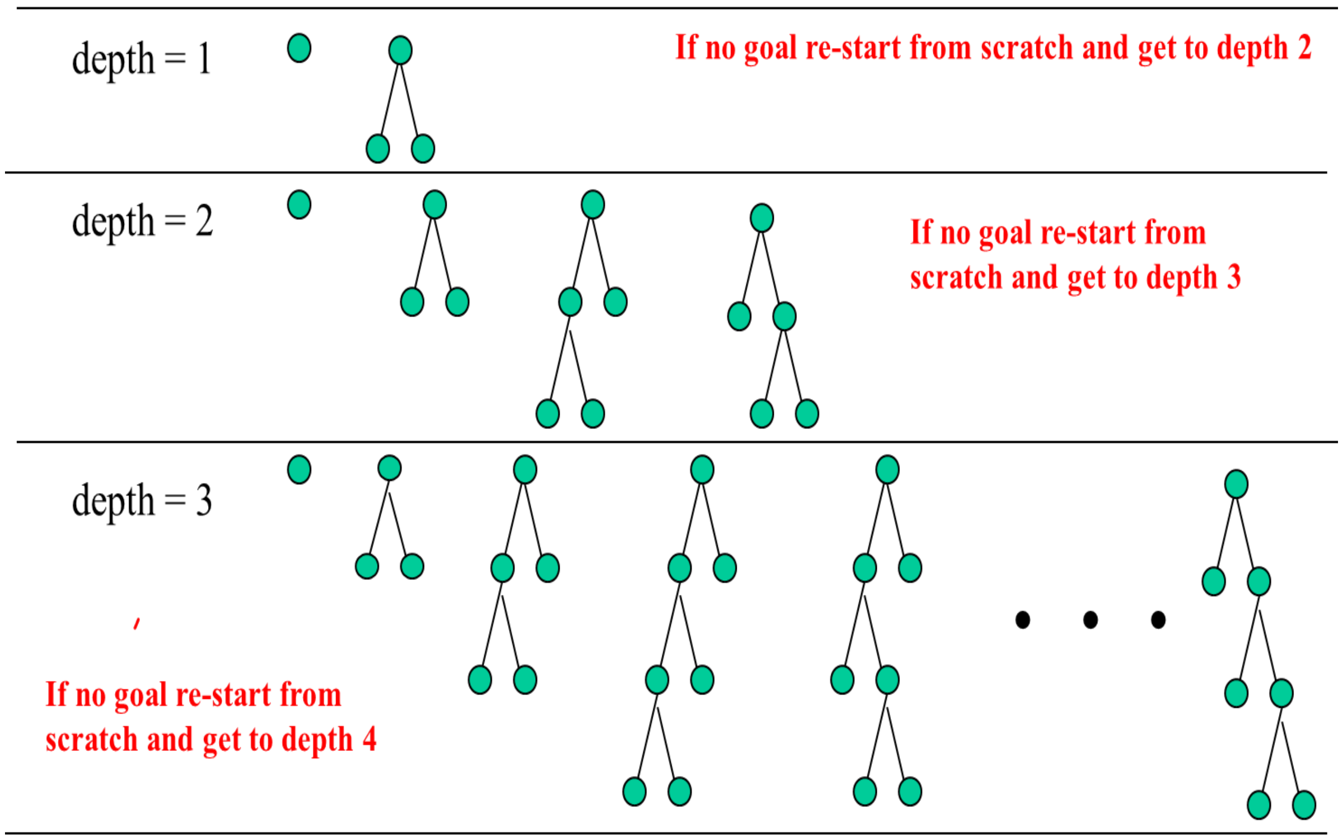
▪For b= 10, m = 5. BSF 111,111 and ID = 123,456 (only 11% more nodes)

(Time) Complexity of IDS

- The larger b the better, but even with b = 2 the search ID will take only 2 times as much as BFS

Further Analysis of Iterative Deepening DFS (IDS)

- Space complexity $O(mb)$
- Same s DFS!


depth = 1
If no goal re-start from scratch and get to depth 2
depth = 2
If no goal re-start from scratch and get to depth 3
depth = 3
If no goal re-start from scratch and get to depth 4

- Complete?

- Optimal?

Further Analysis of Iterative Deepening DFS (IDS)

- Space complexity

  O(mb)

- DFS scheme, only explore one branch at a time

- Complete?

Yes

- Only paths up to depth m, doesn't explore longer paths
  - cannot get trapped in infinite cycles, gets to a solution first

- Optimal? Yes

Summary of Uninformed Search

	Complete	Optimal	Time	Space

DFS	N	N	$O(b^m)$	O(mb)
BFS	Y	Y (shortest)	$O(b^m)$	$O(b^m)$
IDS	Y	Y (shortest)	$O(b^m)$	O(mb)
LCFS				

Learning Goals for today's class

- Select the most appropriate search algorithms for specific problems.
- Depth-First Search vs. Breadth-First Search vs. Iterative Deepening
- Define/read/write/trace/debug different search algorithms

TO DO

- Heuristic Search and A*: Ch 3.6, 3.6.1