Analysis of Uninformed Search Methods

Brian C. Williams

16.410-13

Sep 21st, 2004

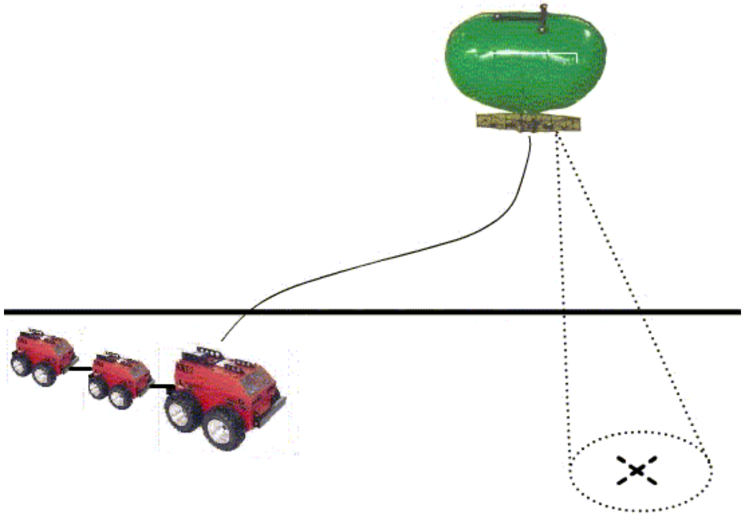
Slides adapted from:
6.034 Tomas Lozano Perez,
Russell and Norvig AIMA

Assignments

- Remember:
  Problem Set #2: Simple Scheme and Search
  due Monday, September 27th, 2004.
- Reading:
  - Uninformed search: more of AIMA Ch. 3

Outline

- Recap
- Analysis
  - Depth-first search
  - Breadth-first search
- Iterative deepening



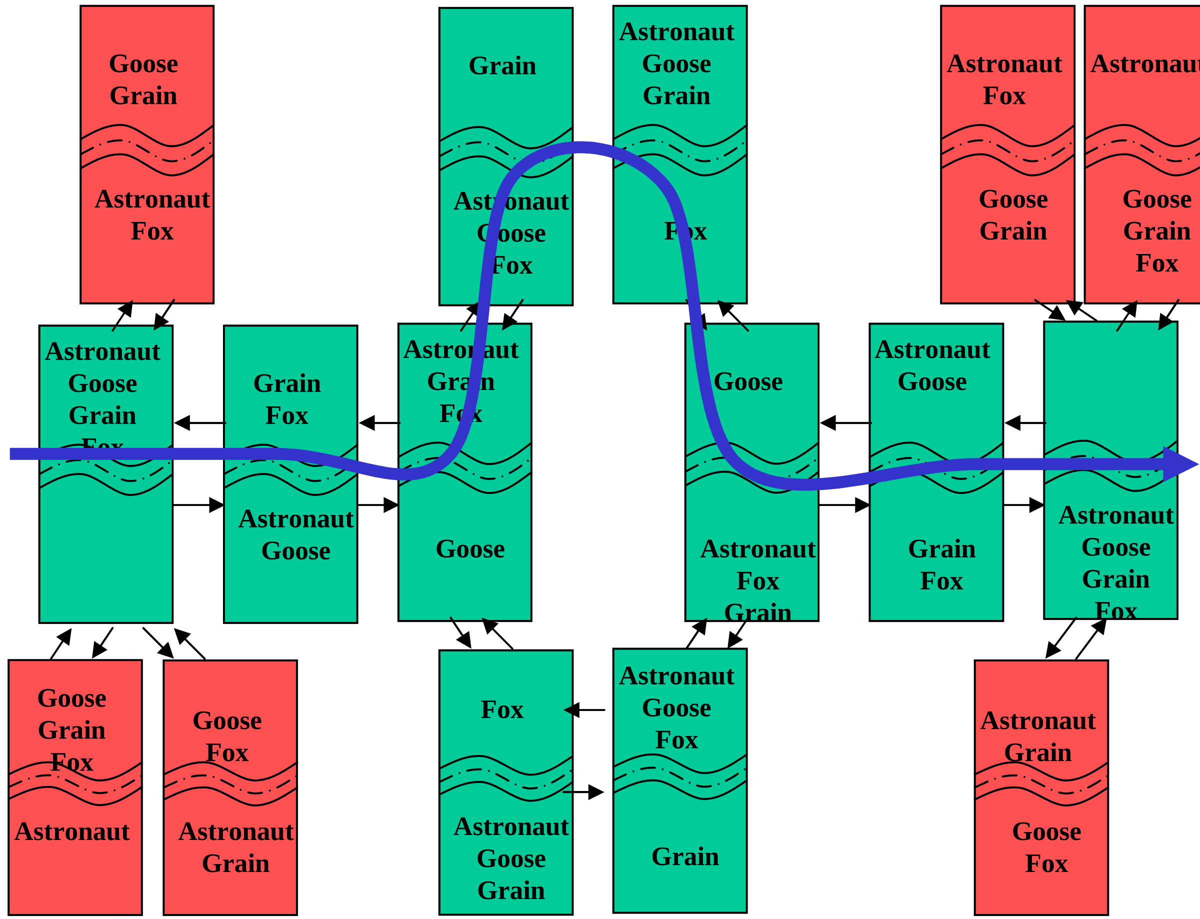
Complex missions must carefully:

- Plan complex sequences of actions
- Schedule tight resources
- Monitor and diagnose behavior
- Repair or reconfigure hardware.

⇨ Most AI problems, like these, may be formulated as state space search.

Problem Solving: Formulate Graph and Find Paths

- Formulate Goal
- Formulate Problem
  - States
  - Operators
- Generate Solution
  - Sequence of states

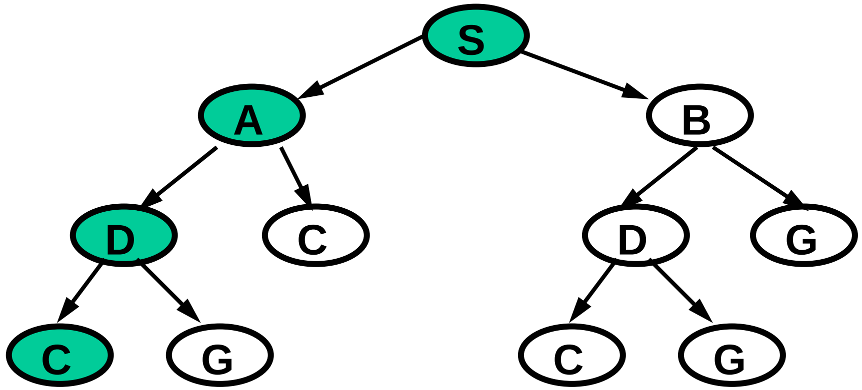


Elements of Algorithm Design

Description: (last Wednesday)

- stylized pseudo code, sufficient to analyze and implement the algorithm
- Implementation (last Monday).

Depth First Search (DFS)

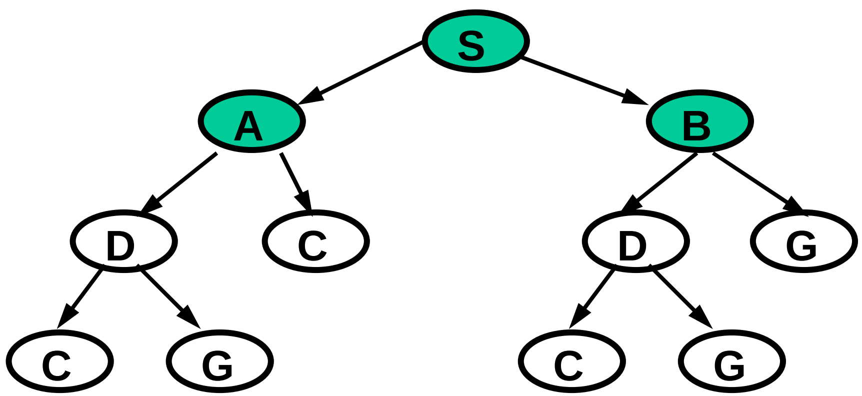


Depth-first:

Add path extensions to front of Q

Pick first element of Q

Breadth First Search (BFS)

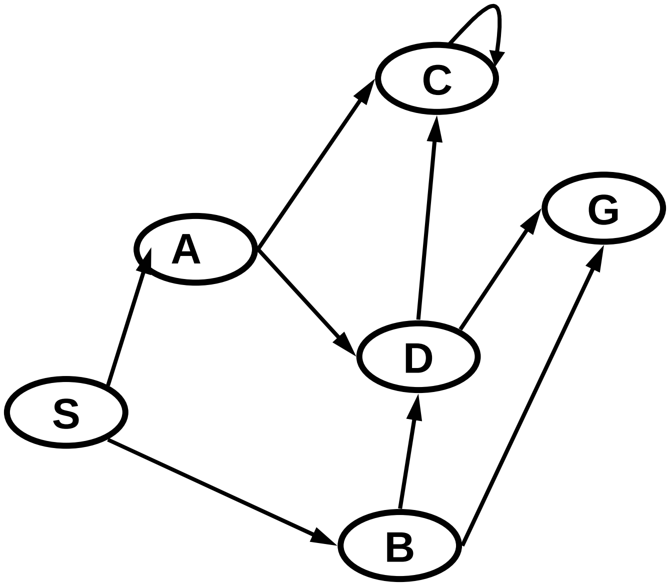


Breadth-first:

Add path extensions to back of Q

Pick first element of Q

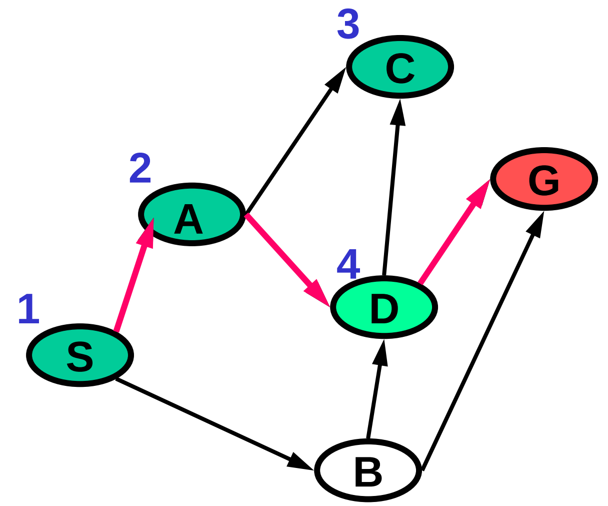
Issue: Starting at S and moving top to bottom, will depth-first search ever reach G?



Depth-First

Effort can be wasted in more mild cases

	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	(D A S) (B S)
5	(C D A S)(G D A S) (B S)
6	(G D A S)(B S)



- C visited multiple times
- Multiple paths to C, D & G

How much wasted effort can be incurred in the worst case?

How Do We Avoid Repeat Visits?

Idea:

- Keep track of nodes already visited.
- Do not place visited nodes on Q.

Does this maintain correctness?

- Any goal reachable from a node that was visited a second time would be reachable from that node the first time.

Does it always improve efficiency?

- Visits only a subset of the original paths, such that each node appears at most once at the head of a path in Q.

How Do We Modify Simple Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S) as only entry;
2. If Q is empty, fail. Else, pick some partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else
   a) Remove N from Q
   b) Find all children of head(N) and create all the one-step extensions of N to each child.
   c) Add to Q all the extended paths;
   d) Go to step 2.

Simple Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S) as only entry; set Visited = ( )
2. If Q is empty, fail. Else, pick some partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else
   a) Remove N from Q
   b) Find all children of head(N) not in Visited and create all the one-step extensions of N to each child.
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited
   e) Go to step 2.

Note: Testing for the Goal

- **Algorithm stops (in step 3) when head(N) = G.**
- **Could have performed test in step 6, as each extended path is added to Q.**
- **But, performing test in step 6 will be incorrect for optimal search, discussed later.**

⇨ We chose step 3 to maintain uniformity with these future searches.

Outline

- Analysis
  - Depth-first search
  - Breadth-first search
- Iterative deepening

Elements of Algorithm Design

Description: (last Wednesday)

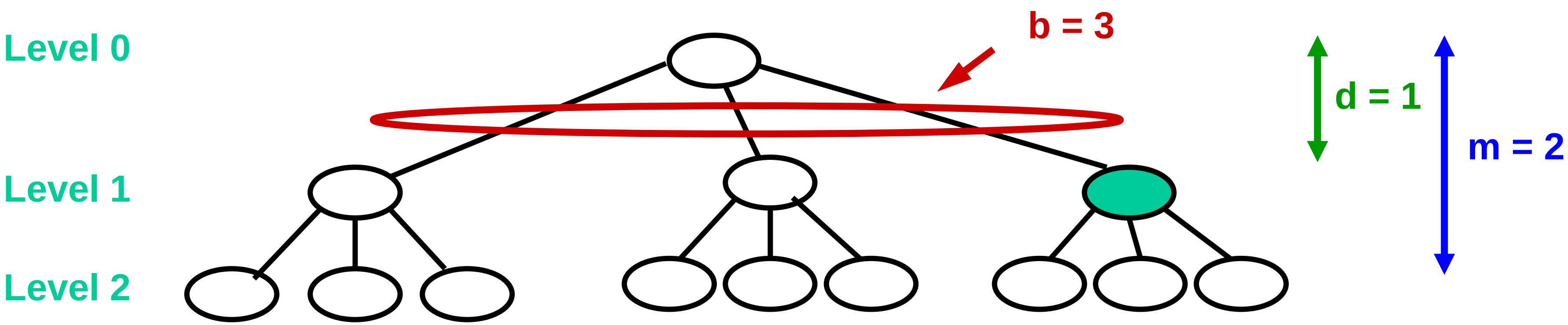
- stylized pseudo code, sufficient to analyze and implement the algorithm
- Implementation (last Monday).

Analysis: (today)

- Soundness:
  - when a solution is returned, is it guaranteed to be correct?
- Completeness:
  - is the algorithm guaranteed to find a solution when there is one?

- Time complexity:
  - how long does it take to find a solution?
- Space complexity:
  - how much memory does it need to perform search?

Characterizing Search Algorithms



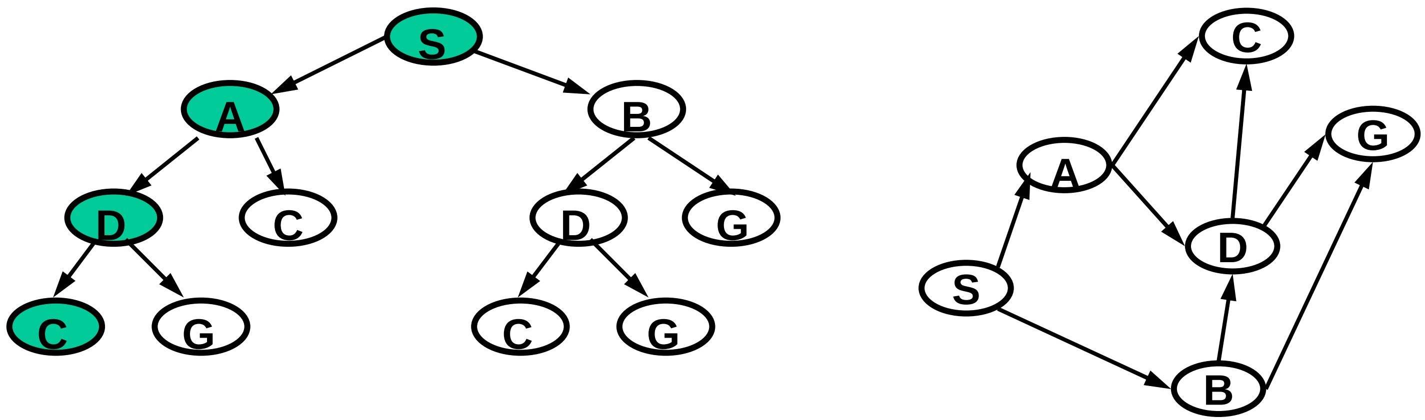
b = maximum branching factor, number of children

d = depth of the shallowest goal node

m = maximum length of any path in the state space

Cost and Performance

Which is better, depth-first or breadth-first?

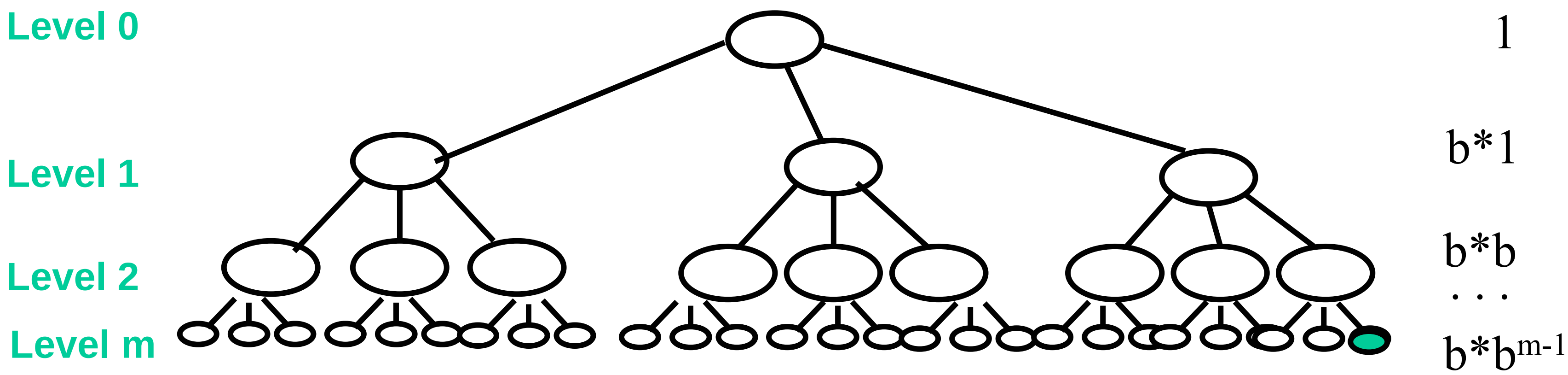


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first				
Breadth-first				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Worst Case Time for Depth-first

Worst case time T is proportional to number of nodes visited



$$T_{dfs} = \qquad [\cancel{b^m} + \ldots \cancel{b} + 1]*c_{dfs}$$

where cdfs is time per node

$$b * T_{dfs} = [b^{m+1} + \cancel{b^m} + \cancel{\ldots} b]*c_{dfs}$$

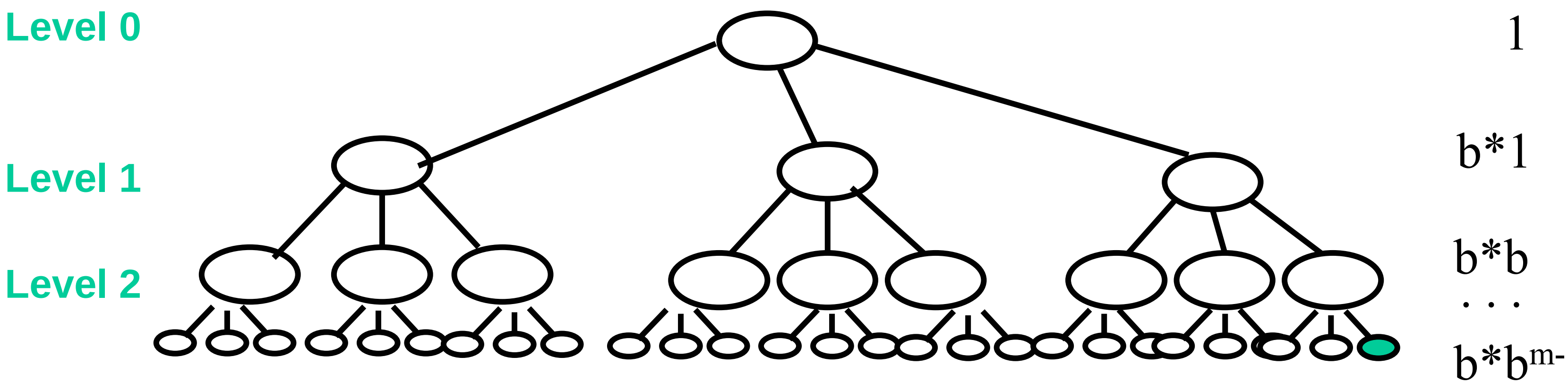
Solve recurrence

$$[b - 1] * T_{dfs} = [b^{m+1} - 1]*c_{dfs}$$

$$T_{dfs} = [b^{m+1} - 1] / [b - 1] *c_{dfs}$$

Cost Using Order Notation

Worst case time T is proportional to number of nodes visited



Order Notation

- $T = O(e)$ if $T \leq c * e$ for some constant c

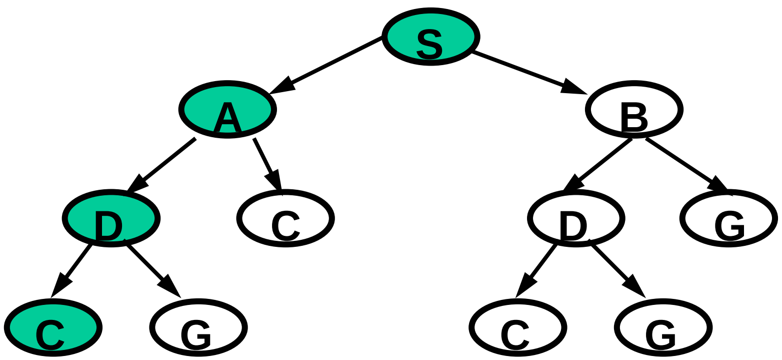
$T_{dfs} = [b^{m}+1 - 1] / [b - 1] * c_{dfs}$

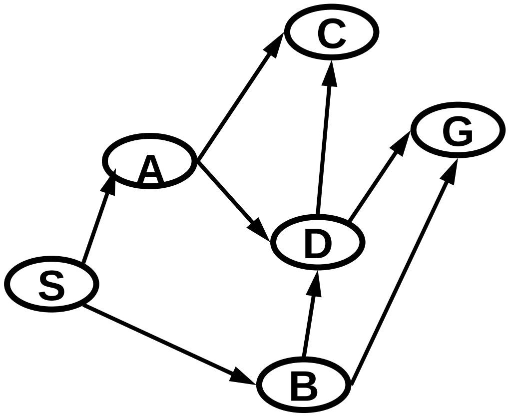
$\quad = O(b^{m+1})$

$\quad \sim O(b^{m})$ for large b

Cost and Performance

Which is better, depth-first or breadth-first?



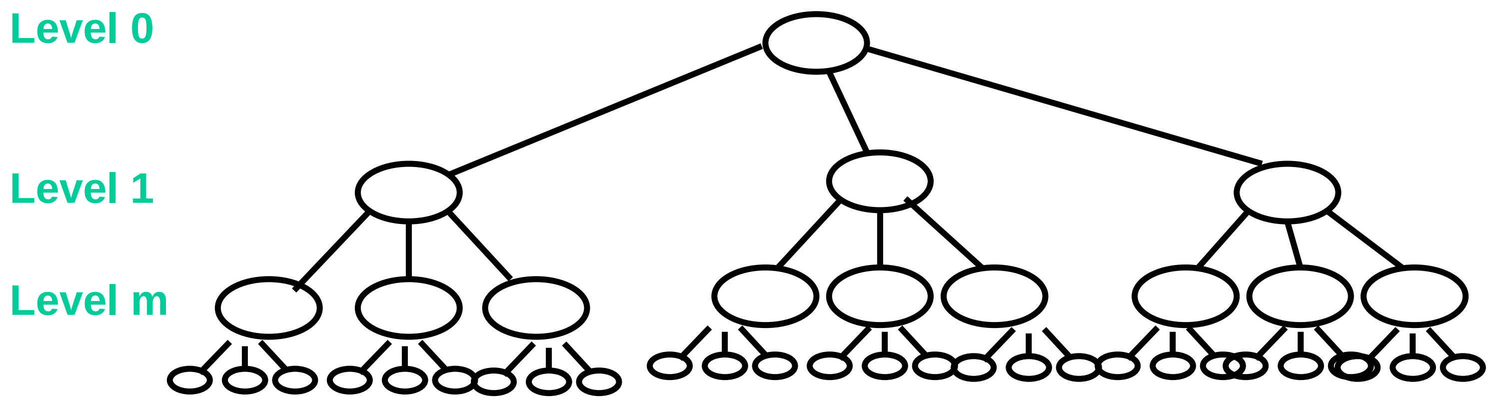


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$			
Breadth-first				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

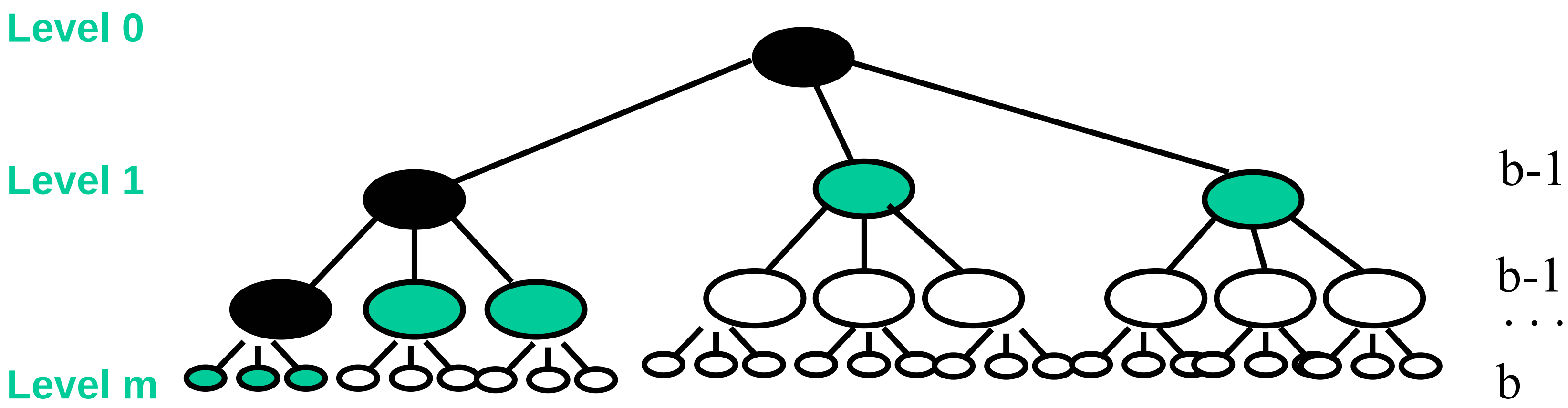
Worst Case Space for Depth-first

Worst case space S_{dfs} is proportional to maximal length of Q



Worst Case Space for Depth-first

Worst case space S_{dfs} is proportional to maximal length of Q



- If a node is queued its parent and siblings have been queued, and its parent dequeued.

  ➔ $S_{dfs} \geq [(b-1)*m+1] *c_{dfs}$ where cdfs is space per node

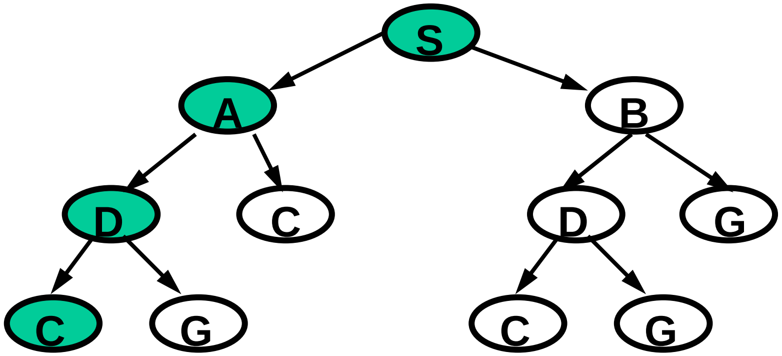
  The children of at most one sibling is expanded at each level.

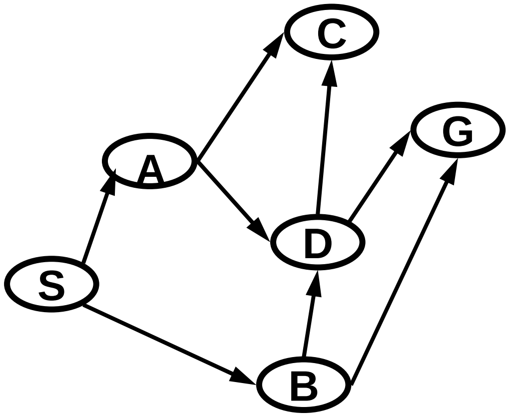
  ➔ $S_{dfs} = [(b-1)*m+1] *c_{dfs}$

- $S_{dfs} = O(b*m)$

Cost and Performance

Which is better, depth-first or breadth-first?



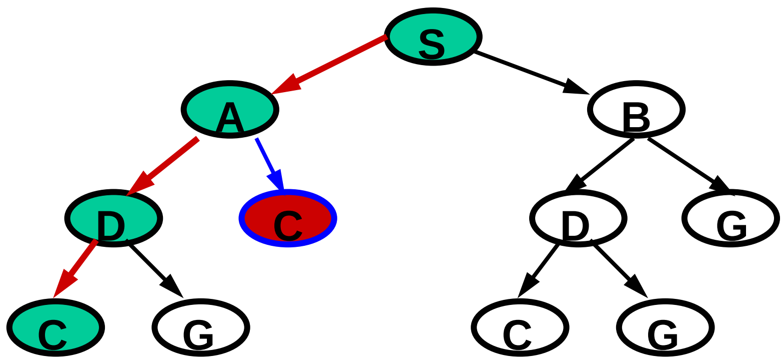


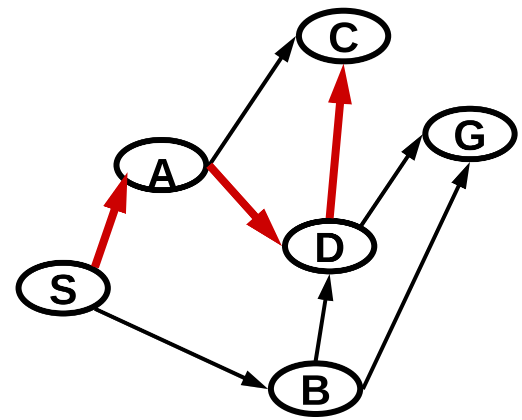
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$		
Breadth-first				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Cost and Performance

Which is better, depth-first or breadth-first?



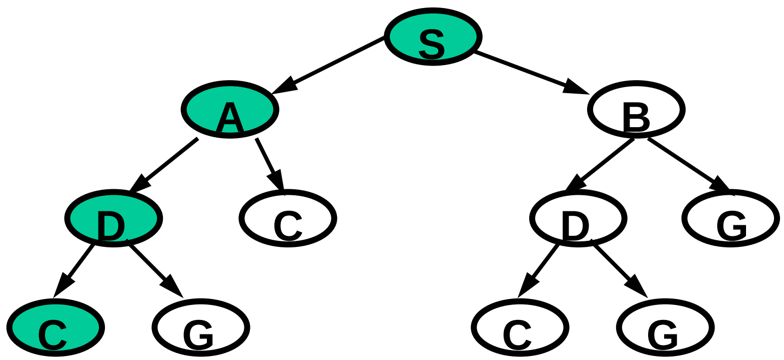


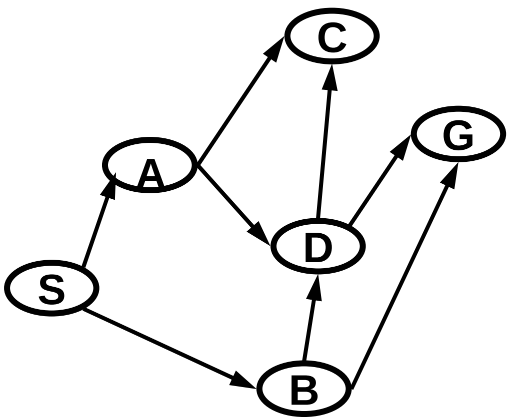
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	
Breadth-first				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Cost and Performance

Which is better, depth-first or breadth-first?



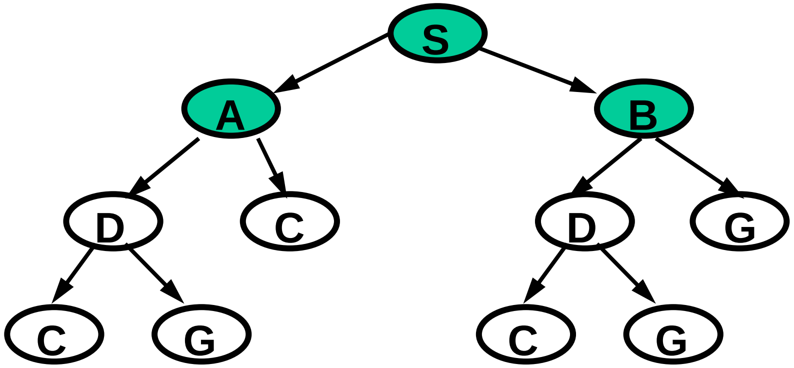


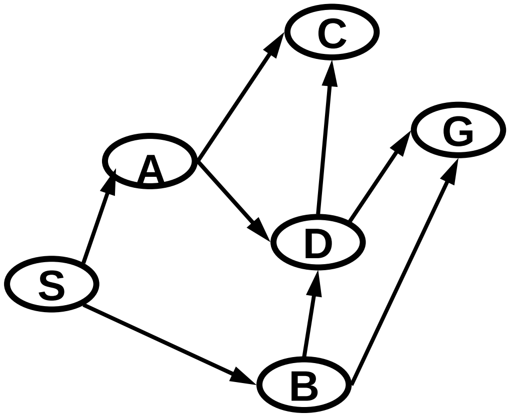
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Cost and Performance

Which is better, depth-first or breadth-first?



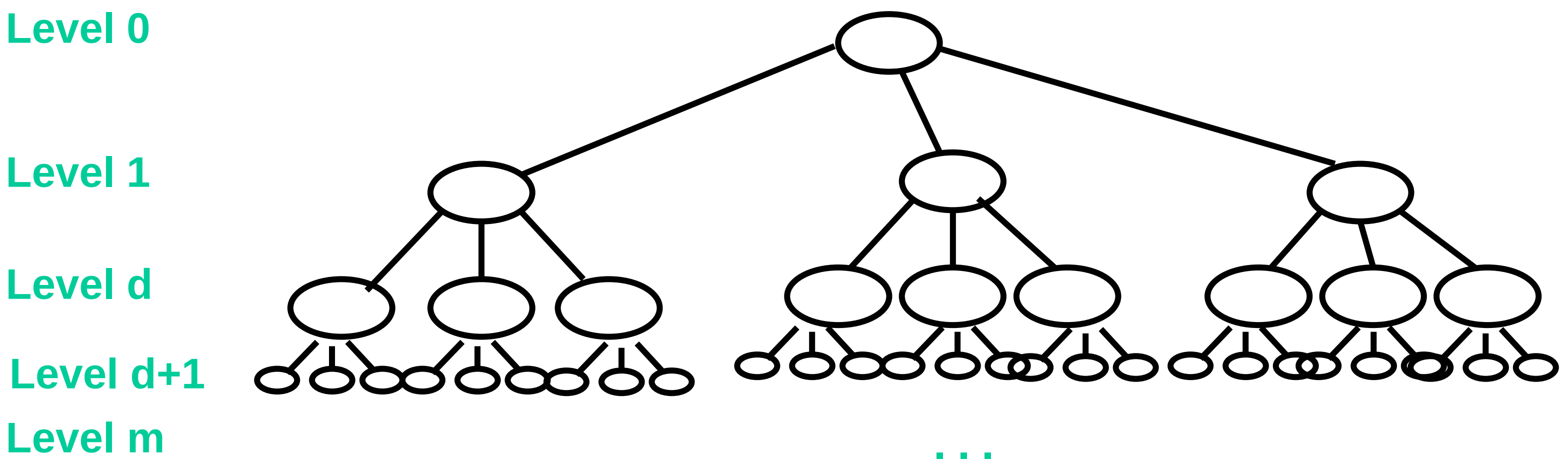


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Worst Case Time for Breadth-first

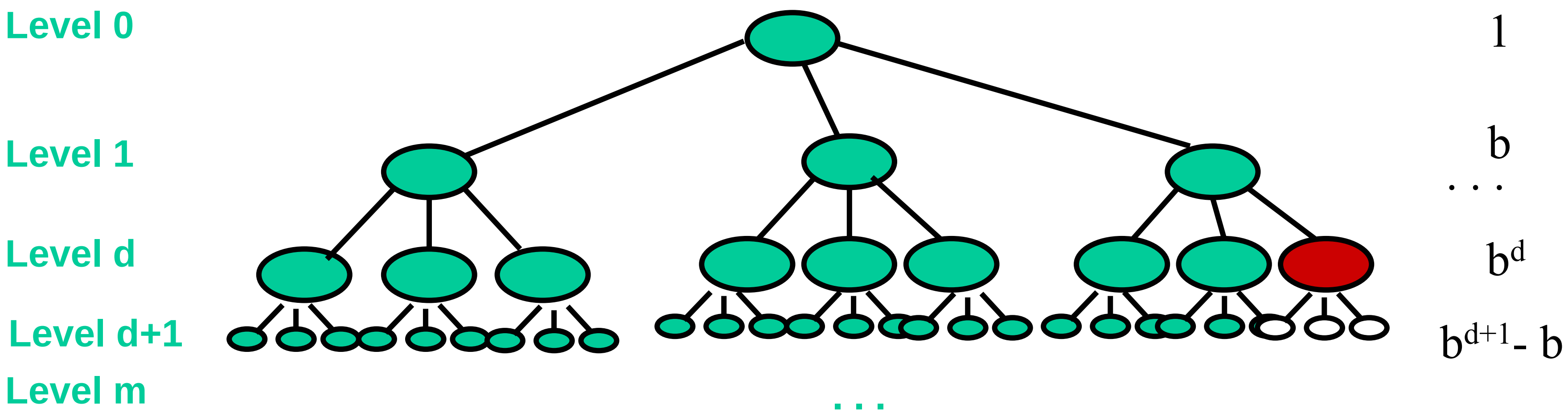
Worst case time T is proportional to number of nodes visited



Consider case where solution is at level d:

Worst Case Time for Breadth-first

Worst case time T is proportional to number of nodes visited



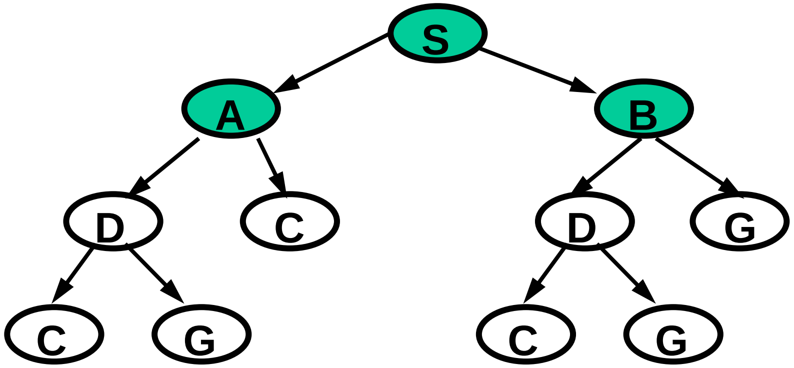
Consider case where solution is at level d:

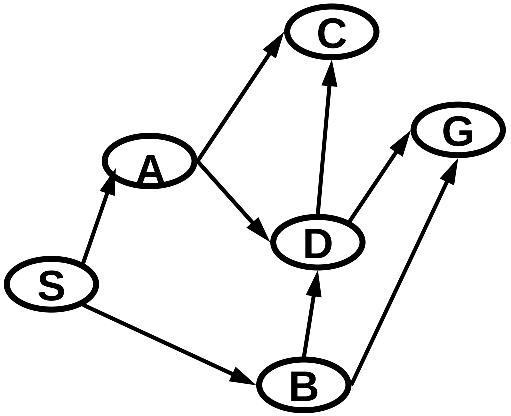
$T_{bfs} = [b^{d+1} + b^d + \ldots b + 1 - b]*c_{bfs}$

$\sim O(b^{d+1})$ for large b

Cost and Performance

Which is better, depth-first or breadth-first?



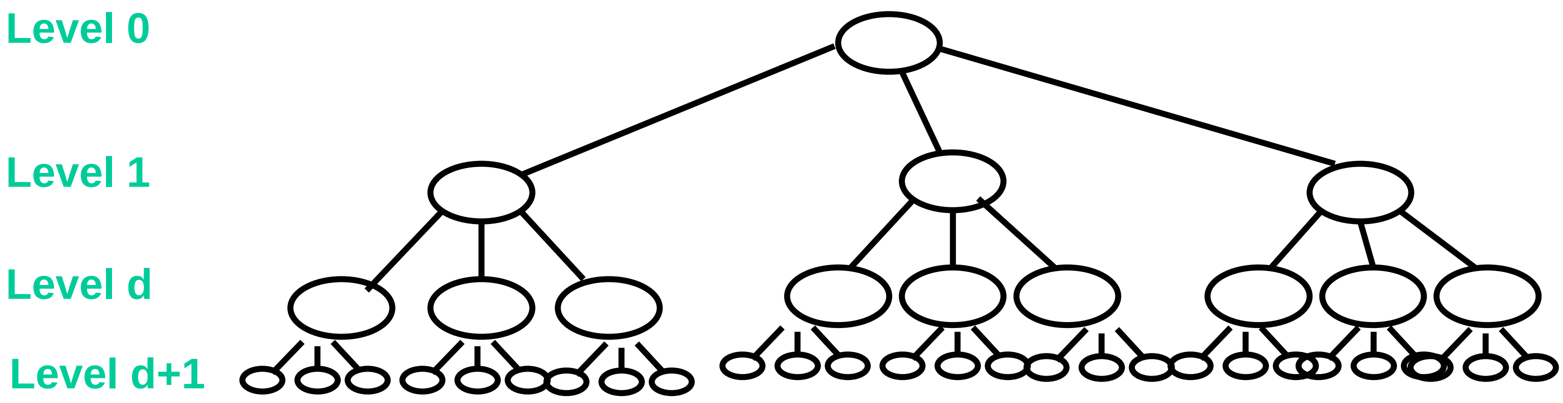


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first	$\sim b^{d+1}$			

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

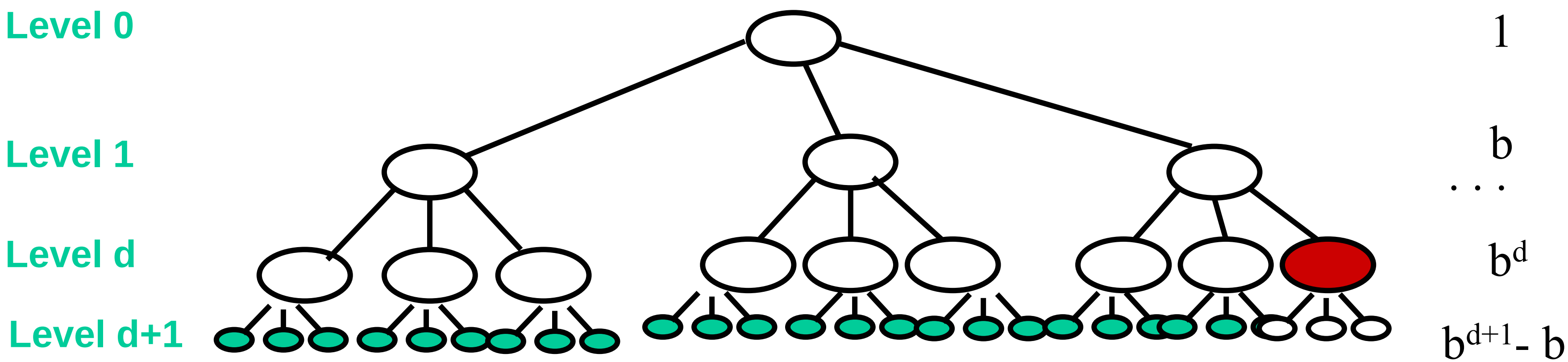
Worst Case Space for Breadth-first

Worst case space S_{dfs} is proportional to maximal length of Q



Worst Case Space for Breadth-first

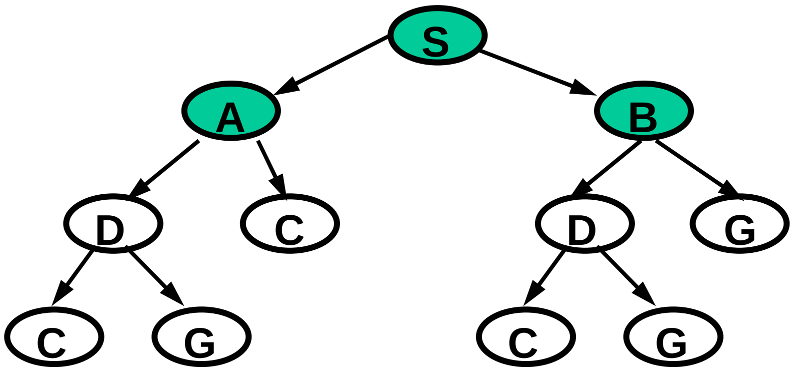
Worst case space S_{dfs} is proportional to maximal length of Q

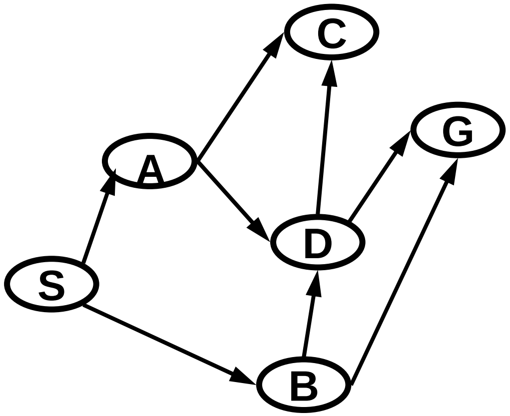


$$S_{bfs} = [b^{d+1} - b + 1]*c_{bfs}$$
$$= O(b^{d+1})$$

Cost and Performance

Which is better, depth-first or breadth-first?

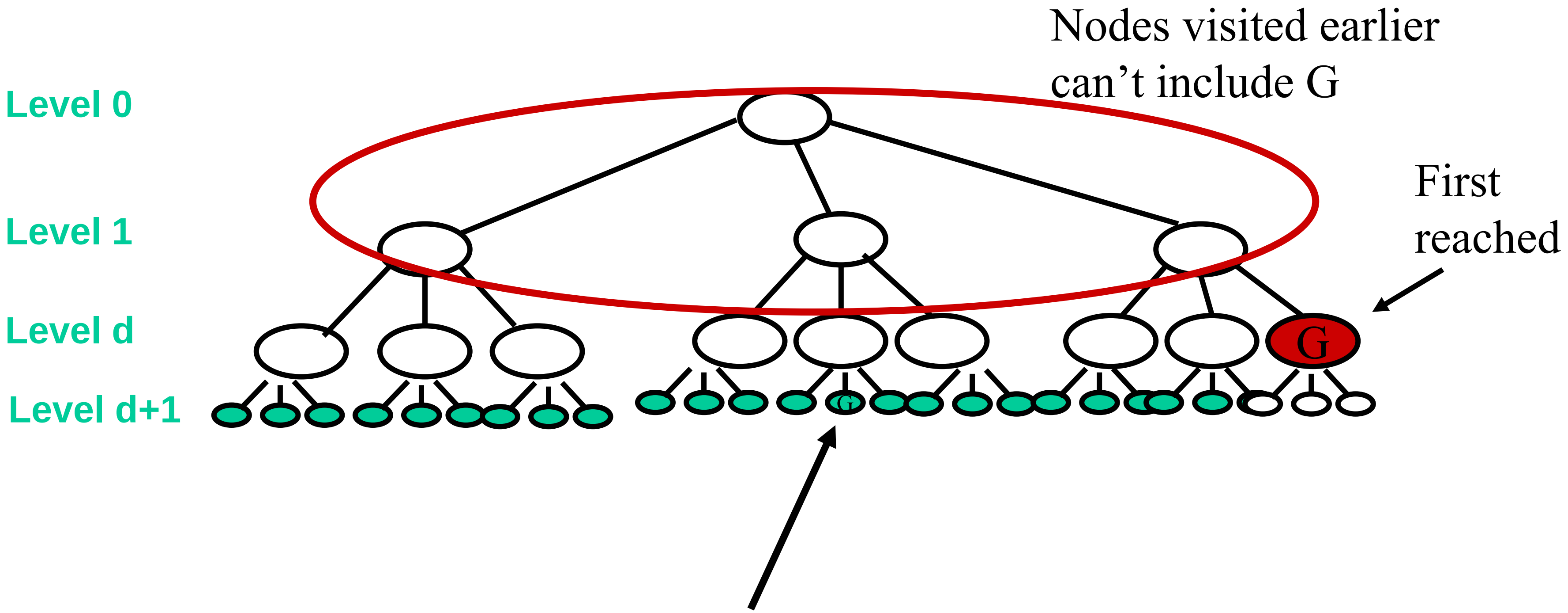




Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first	$\sim b^{d+1}$	b^{d+1}		

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

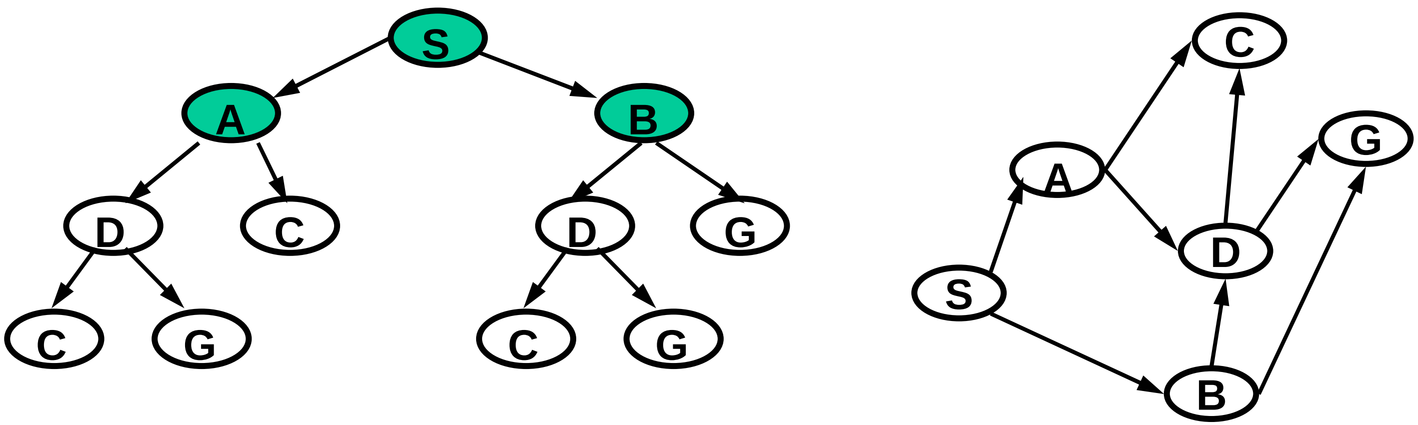
Breadth-first Finds Shortest Path



Assuming each edge is length 1,
other paths to G must be at least as long as first found

Cost and Performance

Which is better, depth-first or breadth-first?

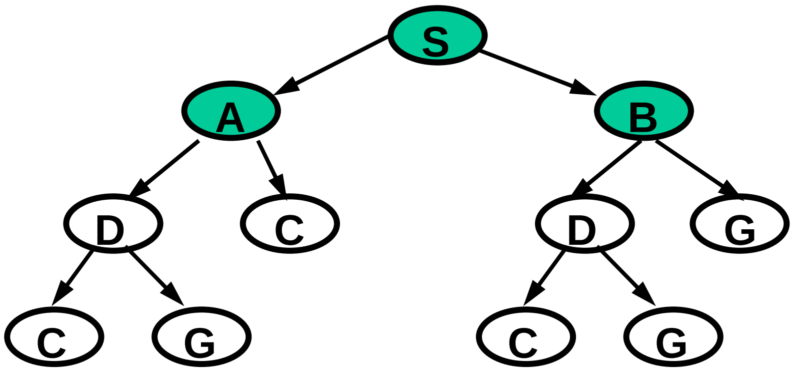


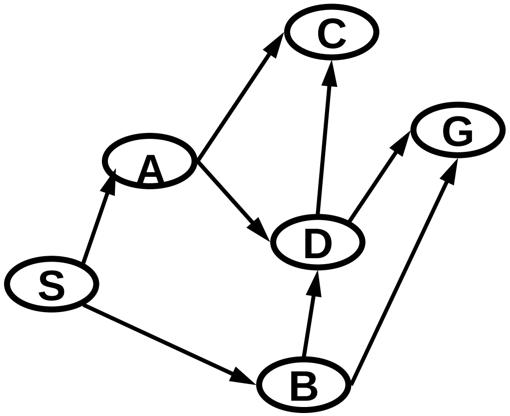
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first	$\sim b^{d+1}$	b^{d+1}	Yes unit lngth	

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Cost and Performance

Which is better, depth-first or breadth-first?



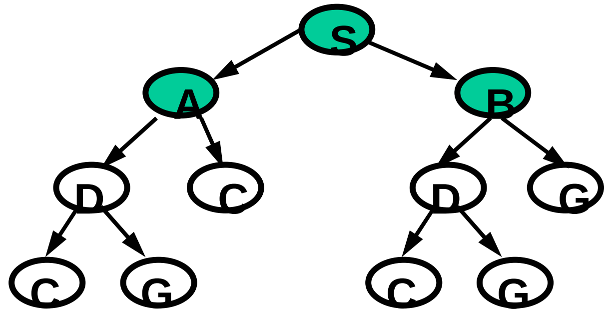


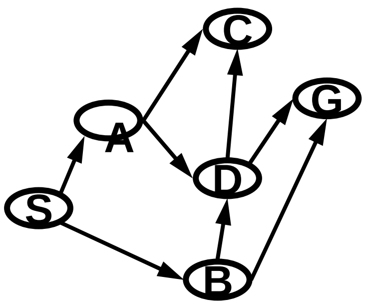
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first	$\sim b^{d+1}$	b^{d+1}	Yes unit lngth	Yes

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

The Worst of The Worst

Which is better, depth-first or breadth-first?





- Assume d = m in the worst case, and call both m.
- Take the conservative estimate: $b^{m+\cdots}1 = O(b^m+1)$

Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^{m+1}	$b*m$	No	Yes for finite graph
Breadth-first	b^{m+1}	b^m	Yes unit lngth	Yes

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

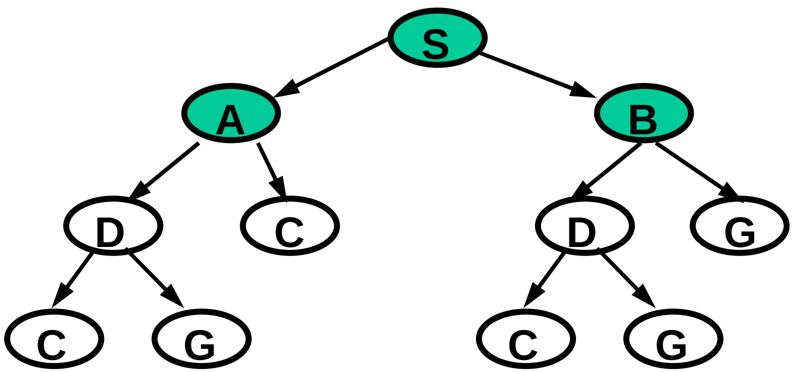
For best first search, which runs out first – time or memory?

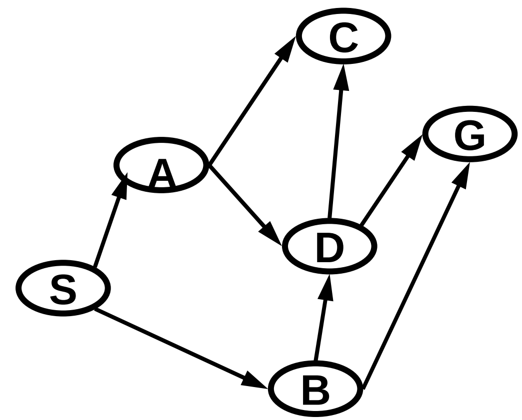
Growth for Best First Search

b = 10; 10,000 nodes/sec; 1000 bytes/node

Depth	Nodes	Time	Memory
2	1,100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

How Do We Get The Best of Both Worlds?





Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first	$\sim b^{d+1}$	b^{d+1}	Yes unit lngth	Yes

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Outline

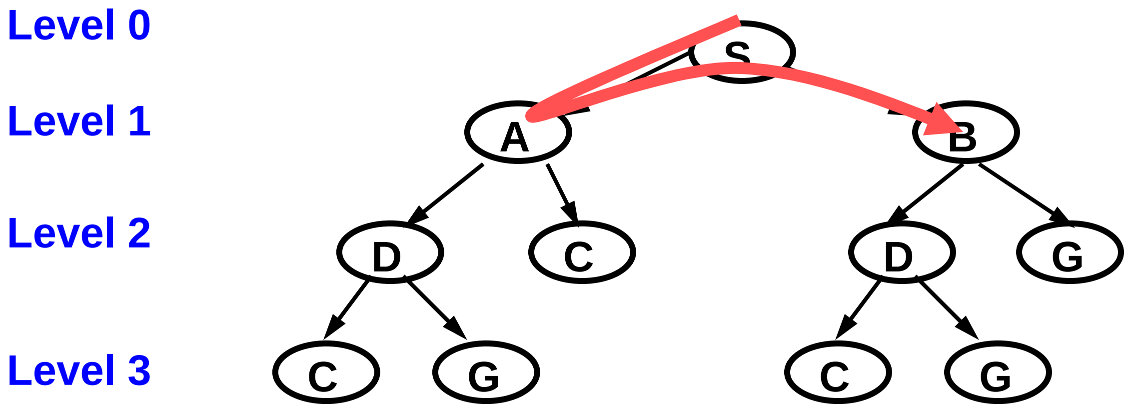
- Analysis
- Iterative deepening

Iterative Deepening (IDS)

Idea:

• Explore tree in breadth-first order, using depth-first search.

➔ Search tree to depth 1, ….

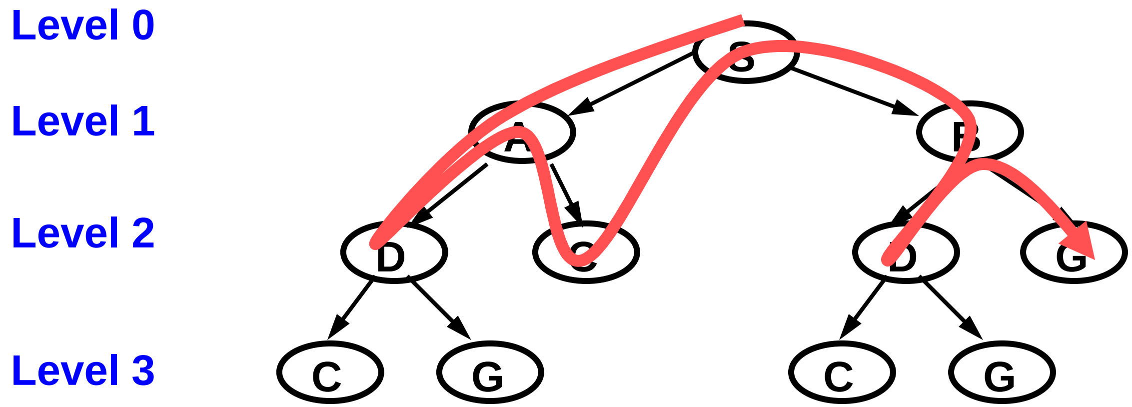


Iterative Deepening (IDS)

Idea:

- Explore tree in breadth-first order, using depth-first search.

➔ Search tree to depth 1, then 2, ….

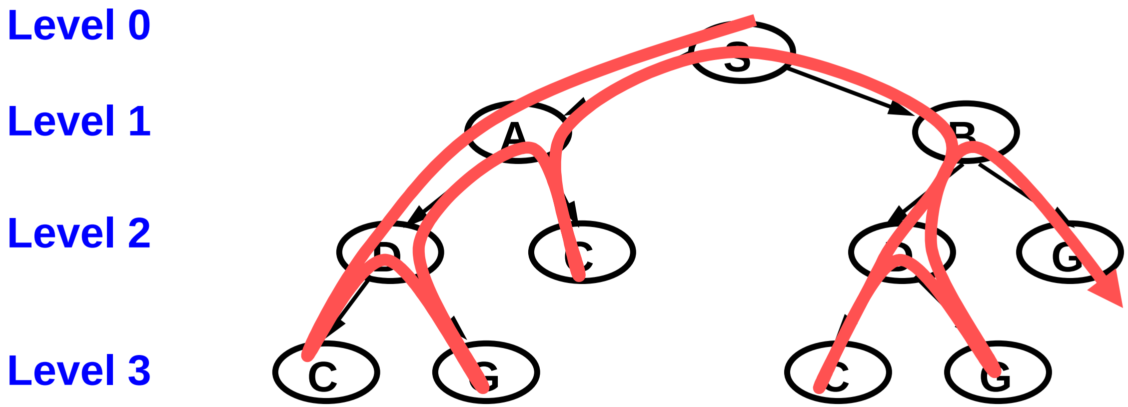


Iterative Deepening (IDS)

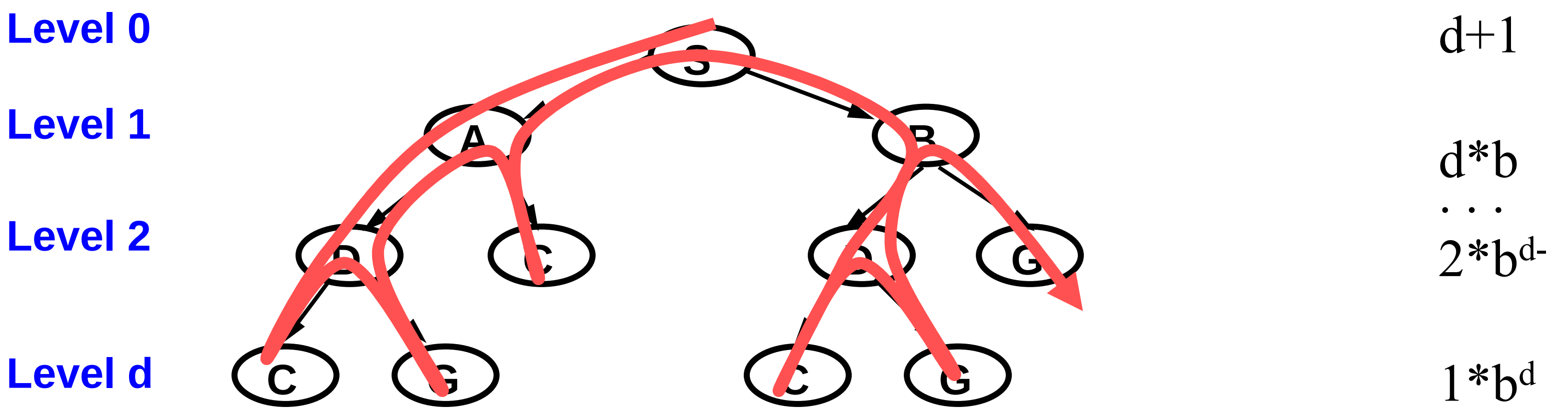
Idea:

• Explore tree in breadth-first order, using depth-first search.

➔ Search tree to depth 1, then 2, then 3….



Speed of Iterative Deepening



Compare speed of BFS vs IDS:

- $T_{bfs} = 1 + b + b^2 + \ldots b^d + (b^{d+1} - b) \qquad = O(b^{d+2})$
- $T_{ids} = (d + 1)1 + (d)b + (d - 1)b^2 + \ldots b^d \qquad = O(b^{d+1})$

➔ Iterative deepening performs better than breadth-first!

Summary

- Most problem solving tasks may be encoded as state space search.
- Basic data structures for search are graphs and search trees.
- Depth-first and breadth-first search may be framed, among others, as instances of a generic search strategy.
- Cycle detection is required to achieve efficiency and completeness.
- Complexity analysis shows that breadth-first is preferred in terms of optimality and time, while depth-first is preferred in terms of space.
- Iterative deepening draws the best from depth-first and breadth-first search.