Analysis of Uninformed Search Methods

Brian C. Williams

16.410

Feb 18th, 2003

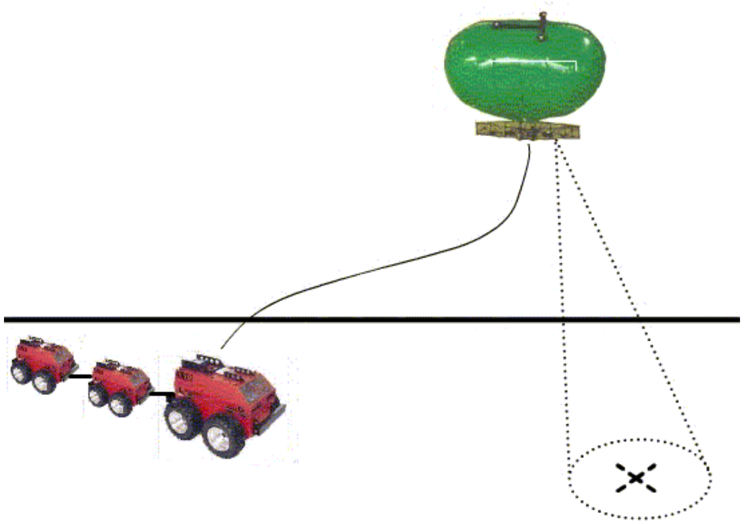
Slides adapted from:
6.034 Tomas Lozano Perez,
Russell and Norvig AIMA

Assignment

- Reading:
  - Solving problems by searching: AIMA Ch. 3
  - Informed search and exploration: AIMA Ch. 4.1-2
- Homework:
  - Online problem set 2 due next Monday Feb 24th

Outline

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



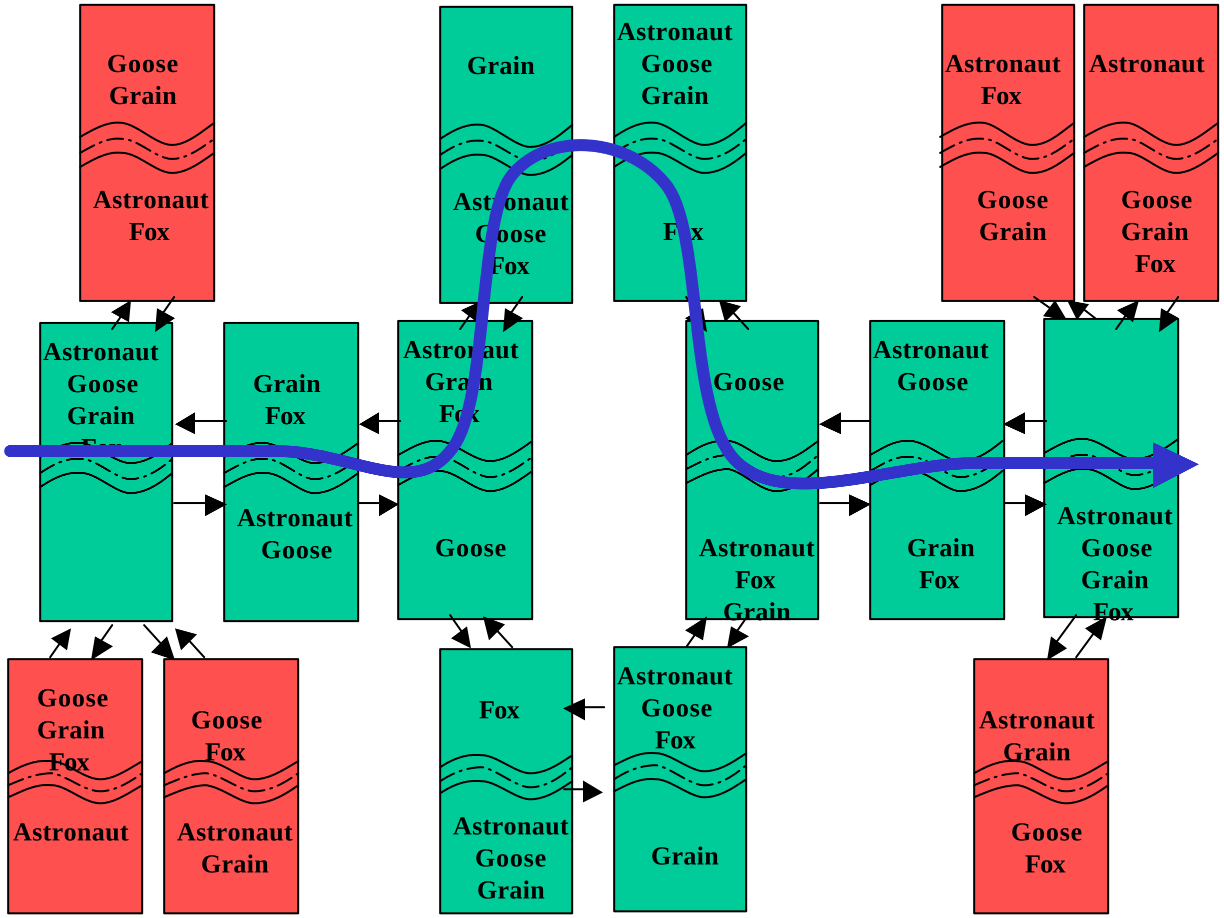
Complex missions must carefully:

- 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 Searches
Paths in a Graph

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



Goose Grain / Astronaut Fox

Grain / Astronaut Goose Fox

Astronaut Goose Grain / Fox

Astronaut Fox / Goose Grain

Astronaut / Goose Grain Fox

Astronaut Goose Grain Fox

Grain Fox / Astronaut Goose

Astronaut Grain Fox / Goose

Goose / Astronaut Fox Grain

Astronaut Goose / Grain Fox

Astronaut Goose Grain Fox

Goose Grain Fox / Astronaut

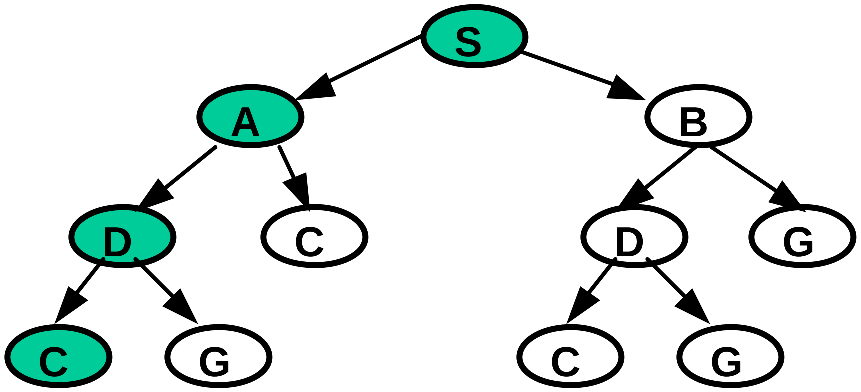
Goose Fox / Astronaut Grain

Fox / Astronaut Goose Grain

Astronaut Goose Fox / Grain

Astronaut Grain / Goose Fox

Depth First Search (DFS)

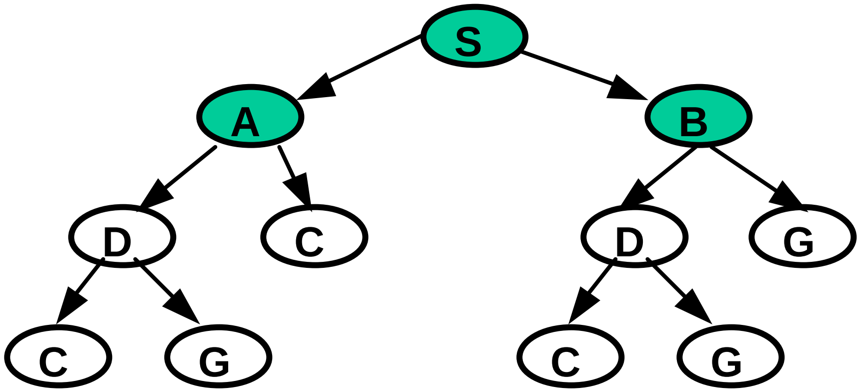


Depth-first:

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

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.**

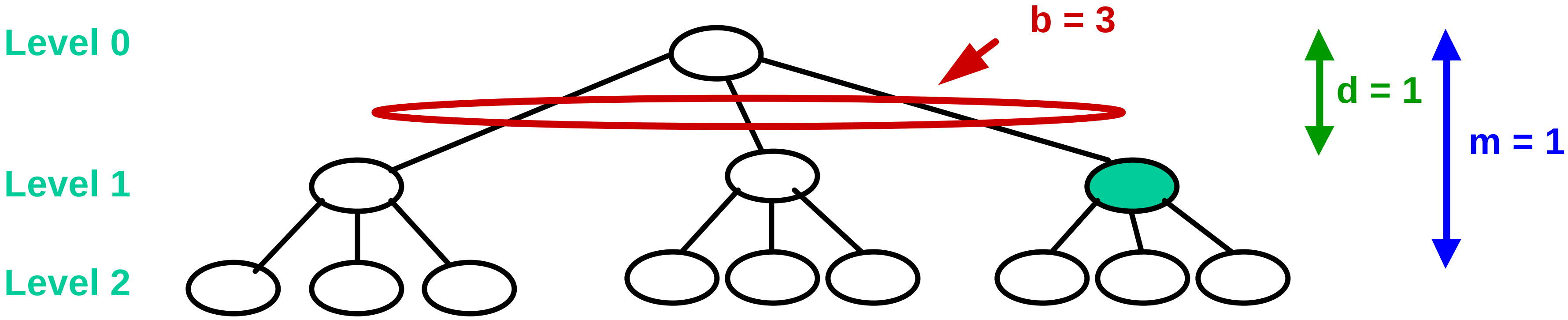
Outline

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

Elements of Algorithmic Analysis

- Soundness:
  - is a solution returned by the algorithm guaranteed to be correct?
- Completeness:
  - is the algorithm guaranteed to find a solution when there is one?
- Optimality:
  - is the algorithm guaranteed to find a best 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?

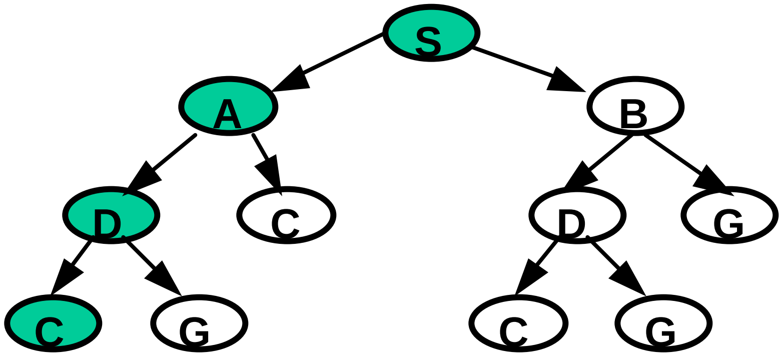
Characterizing Search Algorithms

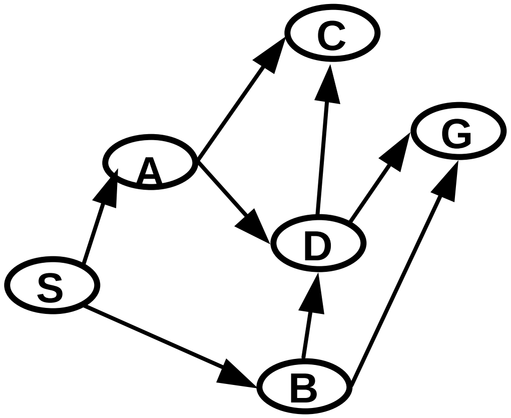


b = maximum branching factor, number of children
d = depth of the shallowest goal node
m = maximum length of any path in the state space

Cost and Performance

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



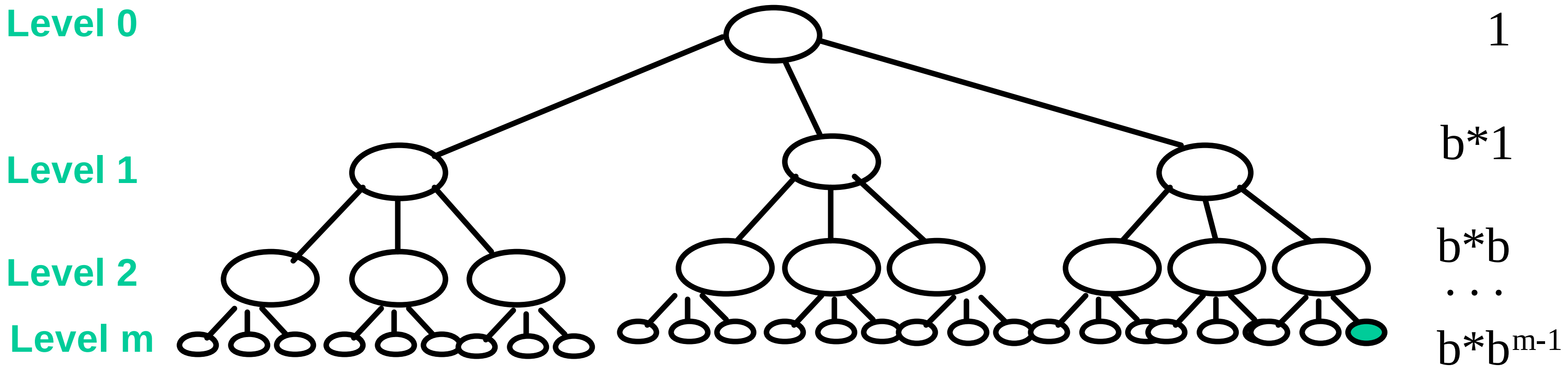


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first				
Breadth-first				

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



$T_{dfs} = b^m + \ldots b + 1$

$b * T_{dfs} = b^{m+1} + b^m + \ldots b$

Solve recurrence

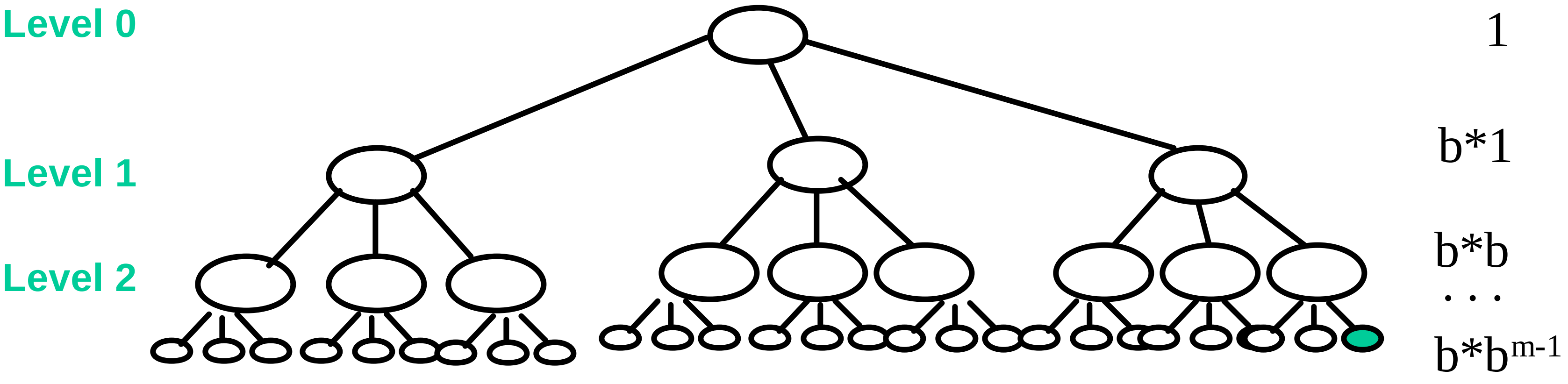
$[b - 1] * T_{dfs} = b^{m+1} - 1$

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

where c_{dfs} is time per node

Cost Using Order Notation

Worst case time T is proportional to number of nodes visited



Level 0

Level 1

Level 2

1

$b*1$

$b*b$

$\cdots$

$b*b^{m-1}$

Order Notation

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

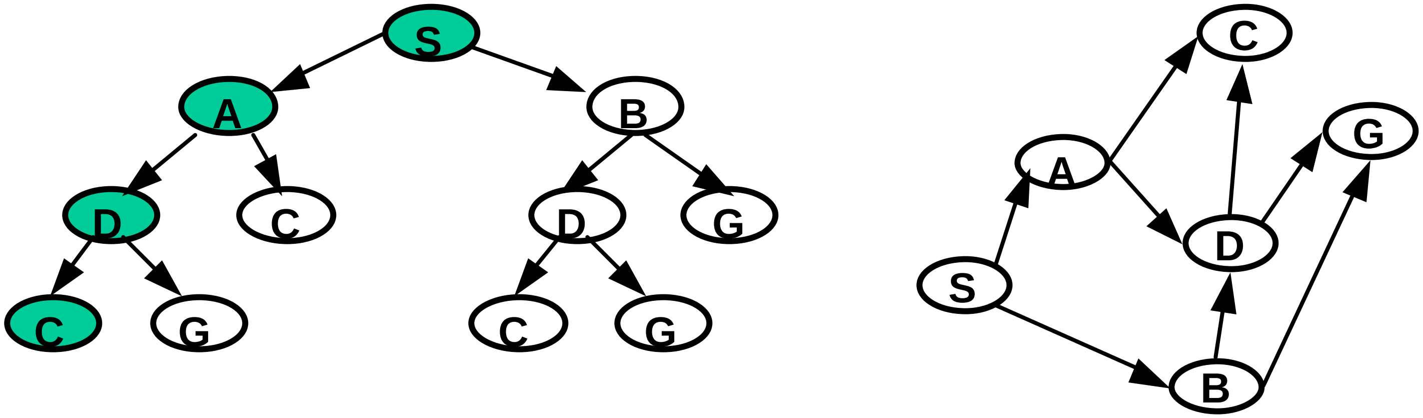
$$T_{dfs} = [b^m + \ldots b + 1]*c_{dfs}$$

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

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

Cost and Performance

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



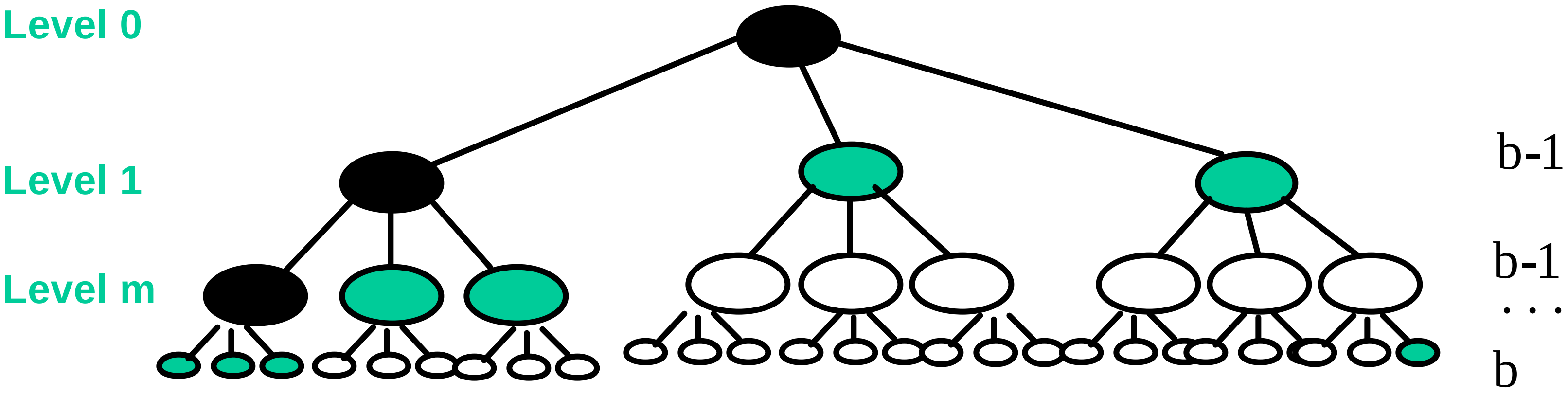
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m			
Breadth-first				

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



- If a node is queued its parent and parent’s siblings have been queued.

➔ $S_{dfs} = (b-1)*m+1$

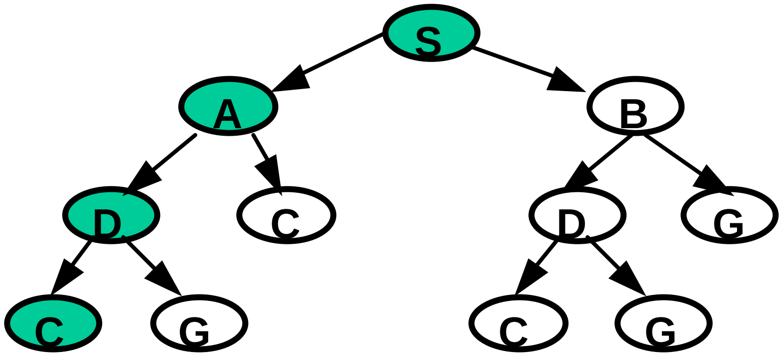
The children of at most one sibling is expanded at each level.

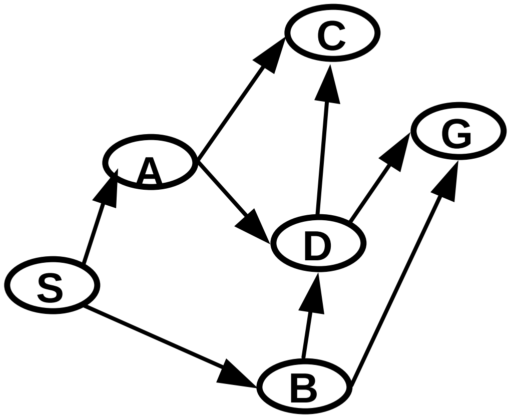
➔ $S_{dfs} = (b-1)*m+1$

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

Cost and Performance

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



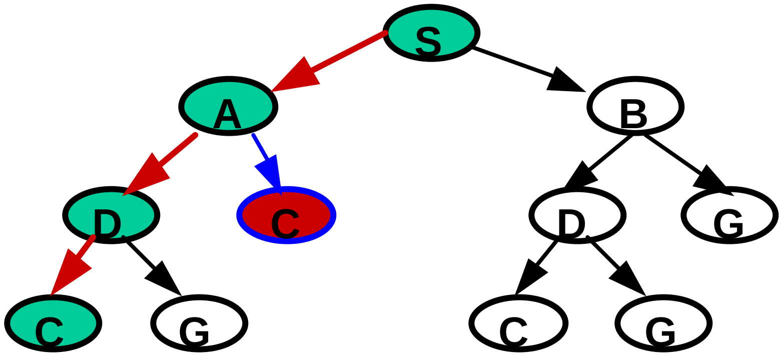


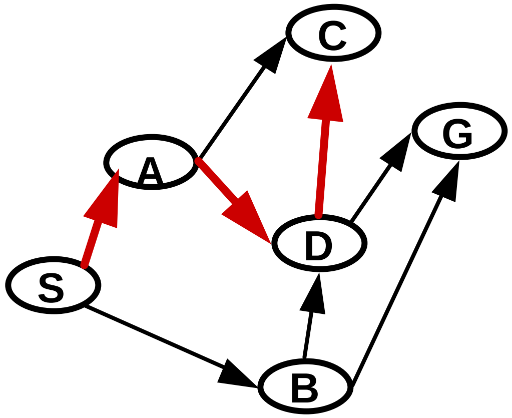
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m	b*m		
Breadth-first				

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?



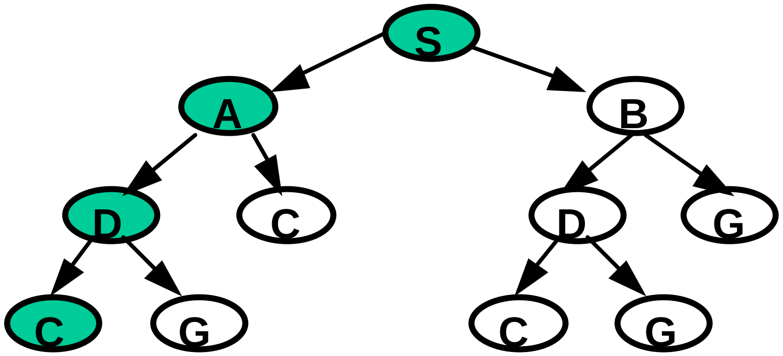


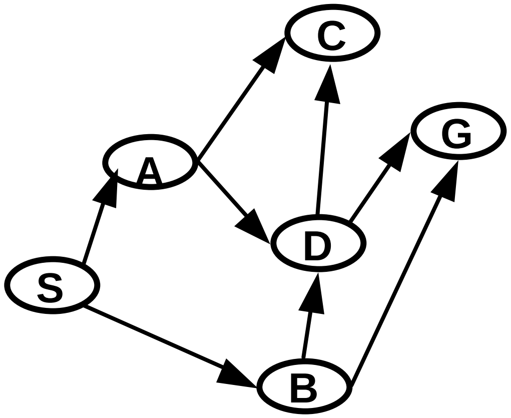
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m	$b*m$	No	
Breadth-first				

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?



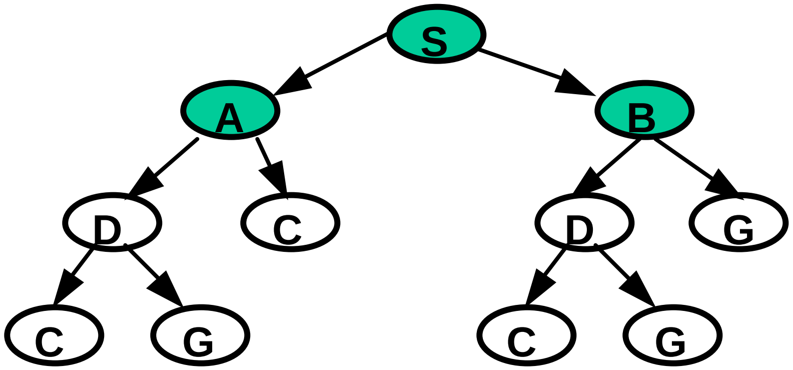


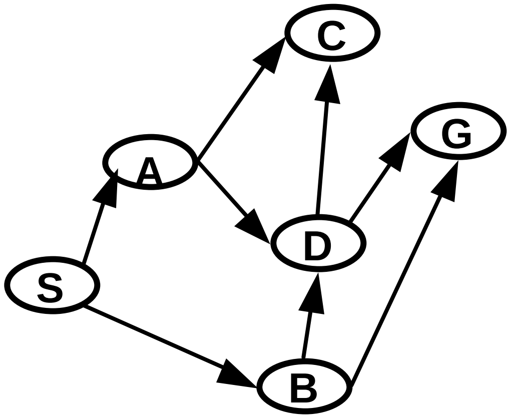
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m	$b*m$	No	Yes for finite graph
Breadth-first				

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?



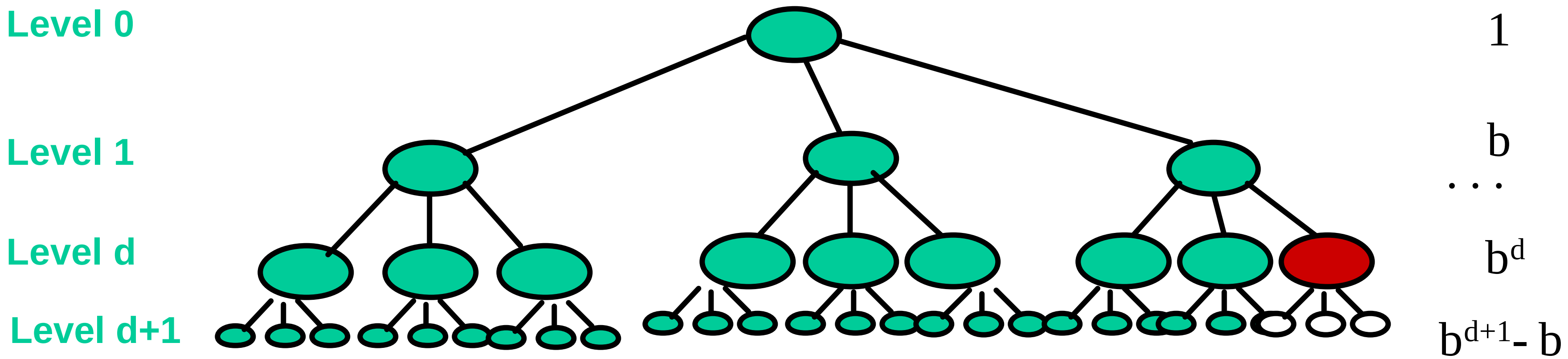


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m	$b*m$	No	Yes for finite graph
Breadth-first				

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

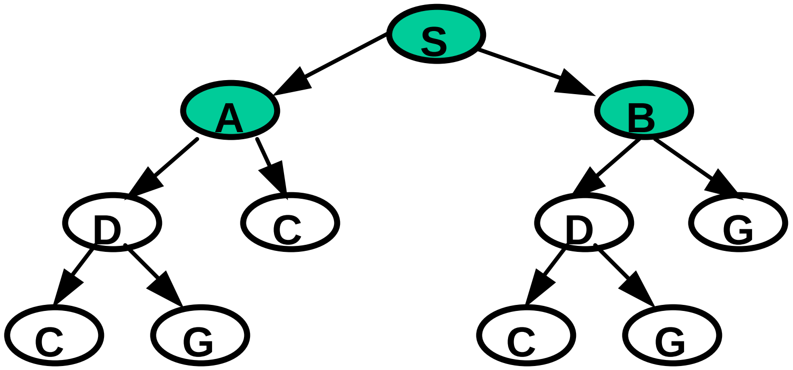
Worst case time T is proportional to number of nodes visited

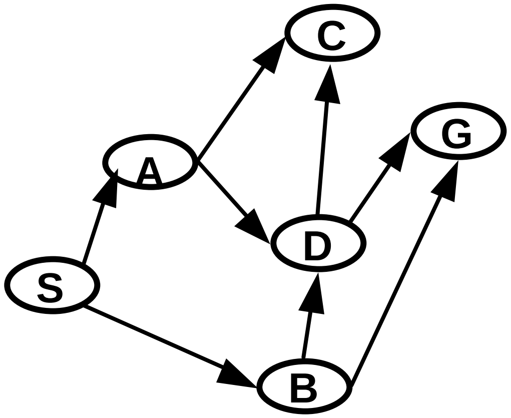


$$T_{bfs} = [b^{d+1} + b^d + \ldots\ b + 1 - b]*c_{bfs}$$
$$= O(b^{d+1})$$

Cost and Performance

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



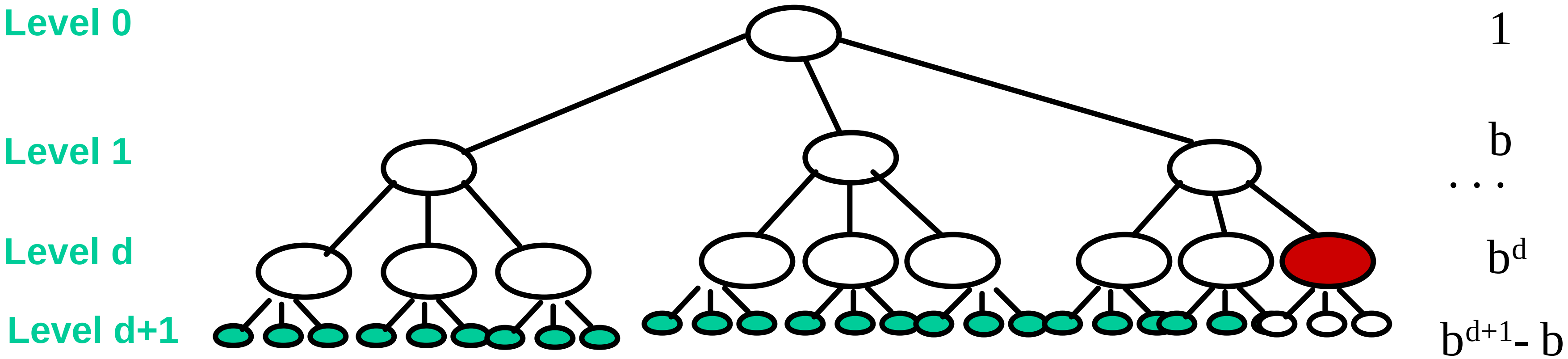


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m	$b*m$	No	Yes for finite graph
Breadth-first	b^{d+1}			

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

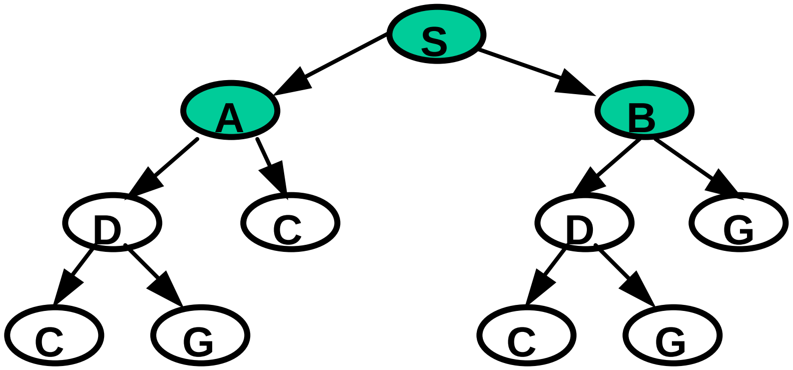
Worst case space S_{dfs} is proportional to maximal length of Q

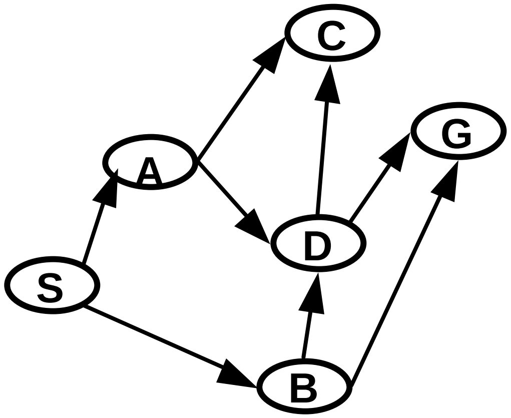


$$
\begin{aligned}
S_{bfs} &= [b^{d+1}- b + 1]*c_{bfs} \\
&= O(b^{d+1})
\end{aligned}
$$

Cost and Performance

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



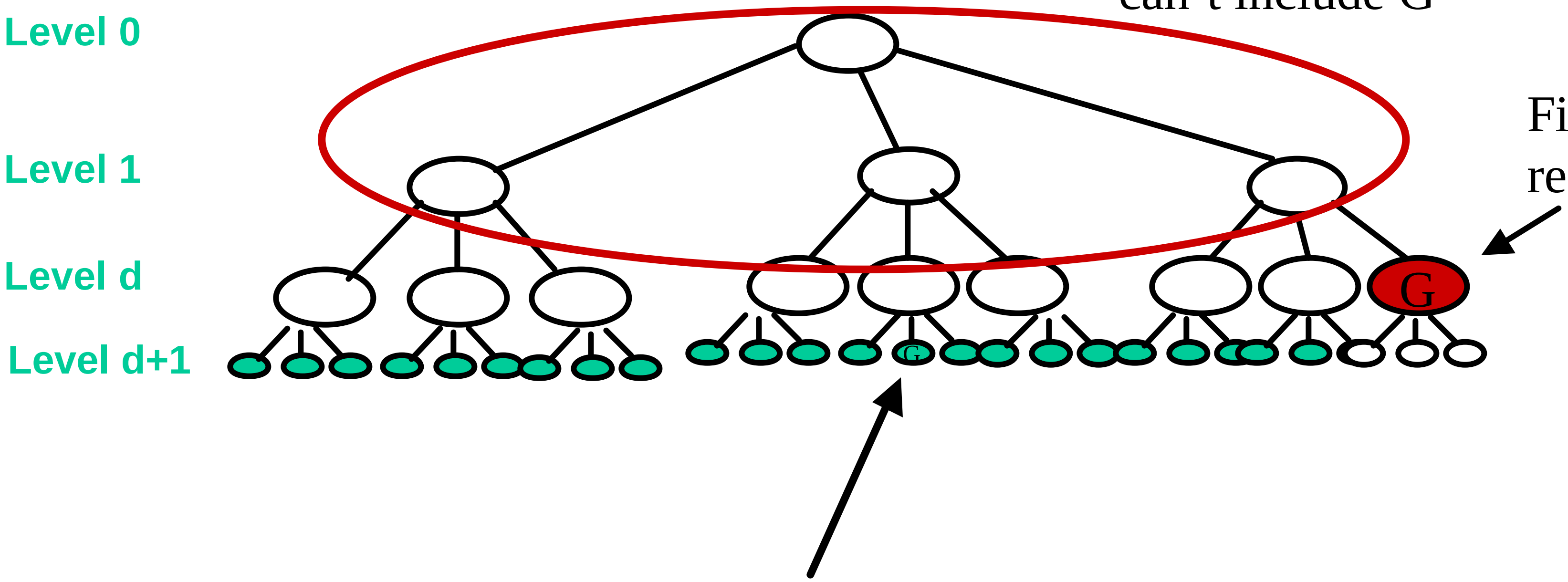


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m	$b*m$	No	Yes for finite graph
Breadth-first	b^{d+1}	b^{d+1}		

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

Nodes visited earlier can't include G

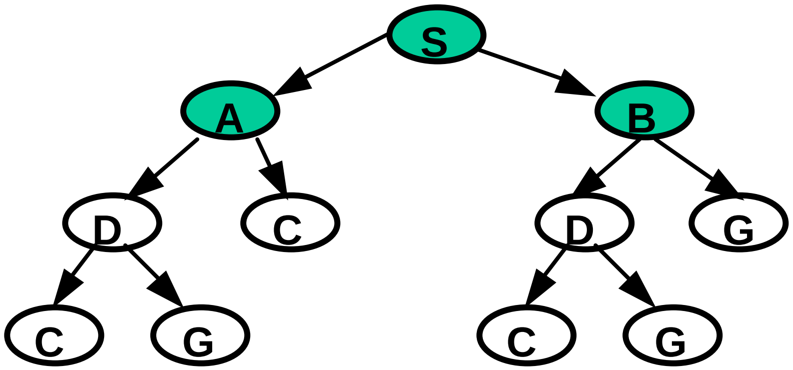


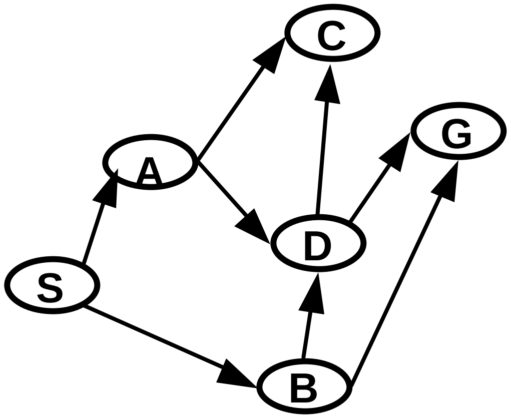
First reached

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?



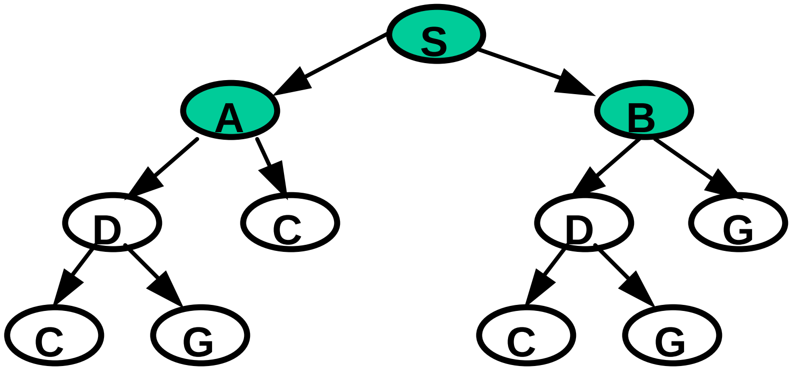


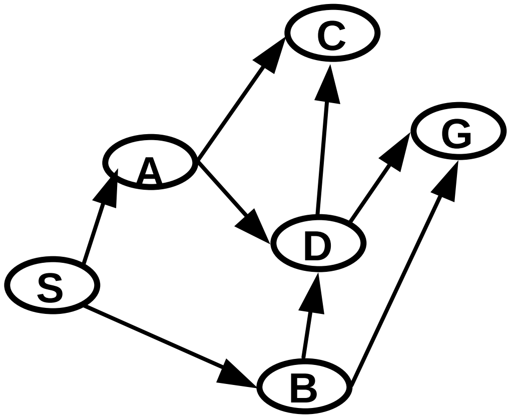
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	b^m	$b*m$	No	Yes for finite graph
Breadth-first	b^{d+1}	b^{d+1}	Yes unit lngth	

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	B^m	b*m	No	Yes for finite graph
Breadth-first	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

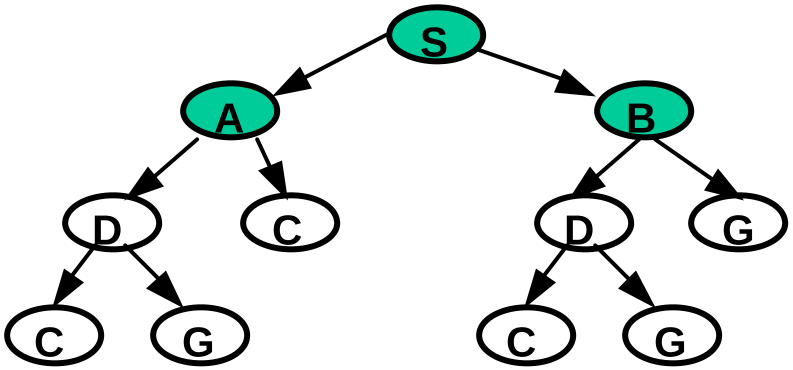
Growth for Best First Search

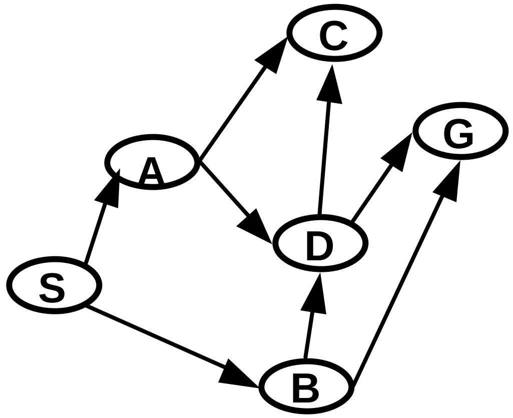
b = 10; 10,000 nodes/sec; 1000 bytes/node

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

Cost and Performance

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



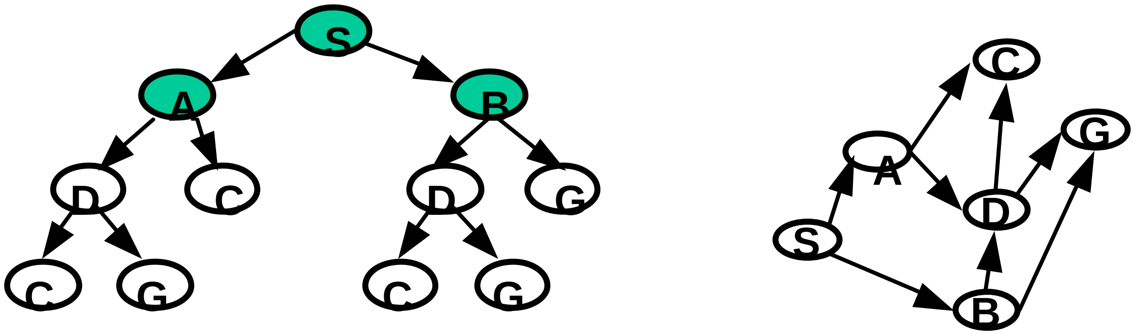


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	~~b^m~~	$b*m$	~~No~~	~~Yes for finite graph~~
Breadth-first	b^{d+1}	~~b^{d+1}~~	Yes unit lngth	Yes

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

Cost and Performance 6.034 Style: What the Electronic Tutor Thinks

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



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

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

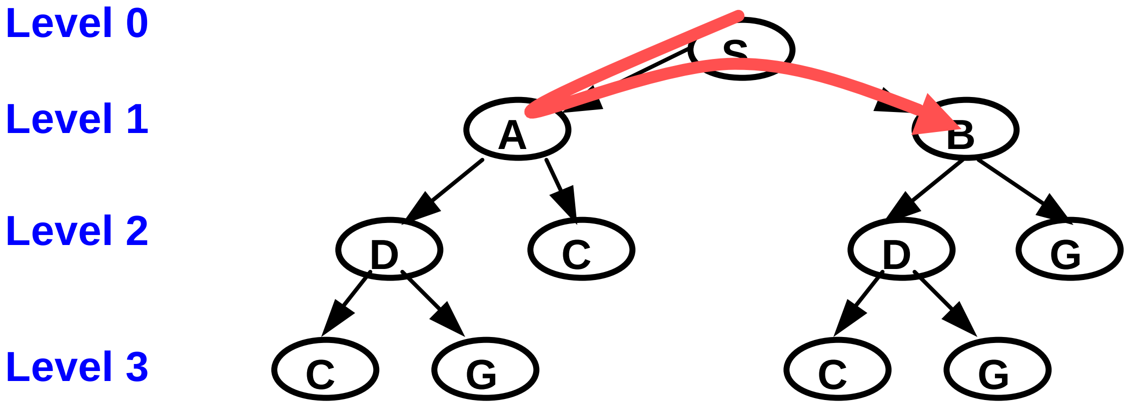
- Recap
- Analysis
- Iterative deepening

Iterative Deepening (IDS)

Idea:

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

➔ Search tree to depth 1, ….

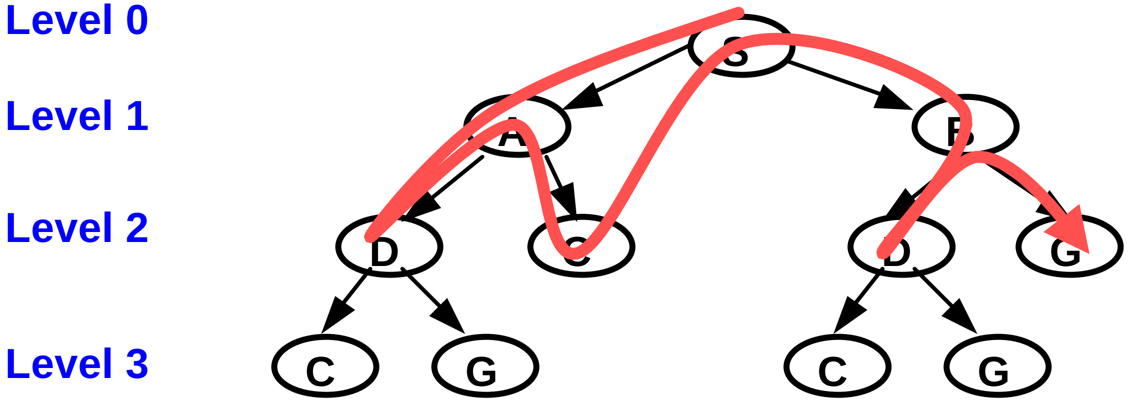


Iterative Deepening (IDS)

Idea:

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

➔ Search tree to depth 1, then 2, ….

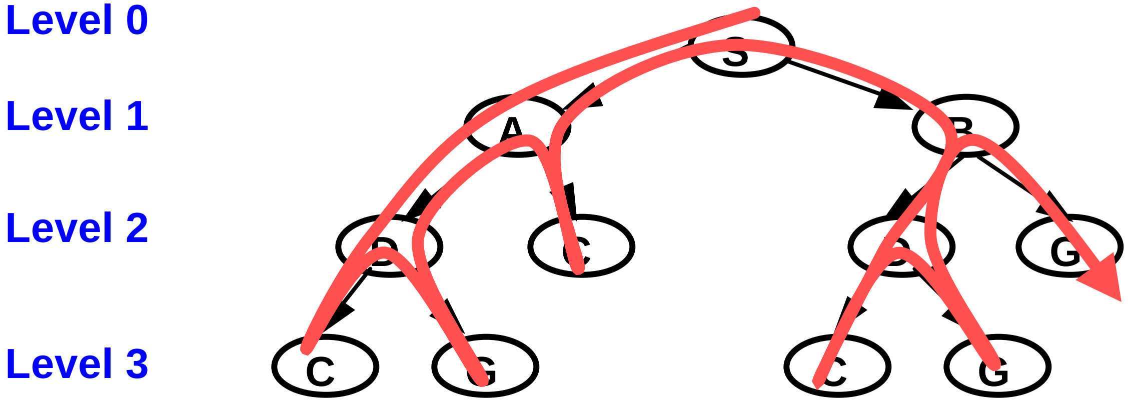


Iterative Deepening (IDS)

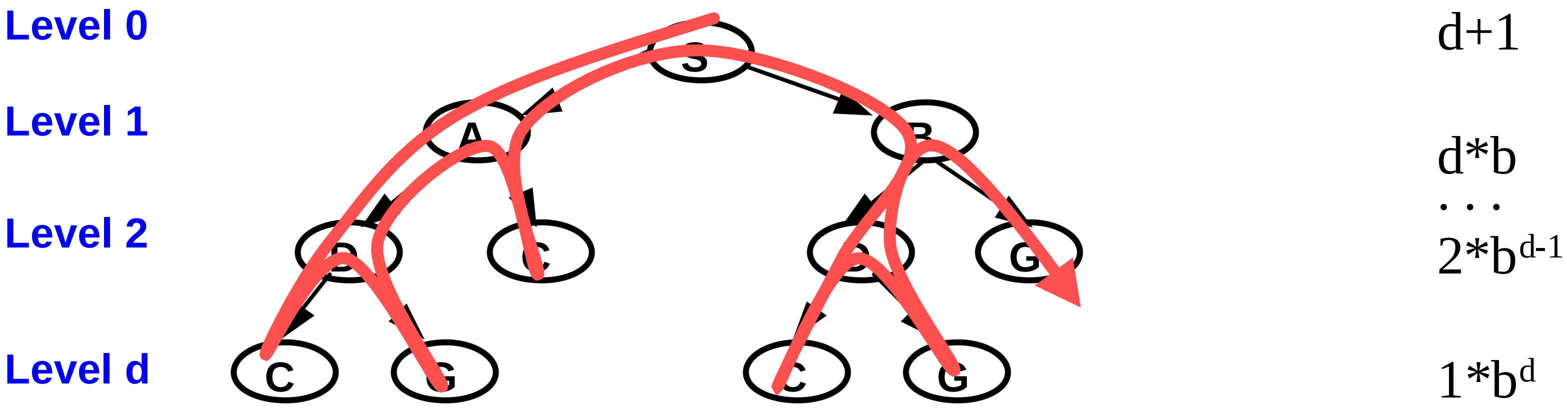
Idea:

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

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



Cost of Iterative Deepening



Level 0 — d+1

Level 1 — d*b

. . .

Level 2 — $2*b^{d-1}$

Level d — $1*b^d$

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

➔ 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.