Analysis of Uninformed Search Methods

Brian C. Williams

16.410-13

Sep 15th, 2010

Draws from materials in:
6.034 Tomas Lozano Perez,
Russell and Norvig AIMA
6.046J Charles E. Leiserson

Assignments

- Assignment:
  - Problem Set #1 due today, Wed Sept 15th, 2010.
  - Problem Set #2: Uninformed Search out today, due Wednesday, September 22nd, 20010.
- Reading:
  - Today: Asymptotic Analysis, Lecture 2 Notes of 6.046J
    Recurrences, Lecture 12 Notes of 6.042J.
  - Monday: Proofs & Induction, Lectures 2 and 3 of 6.042J.

Outline

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

Autonomous Systems:

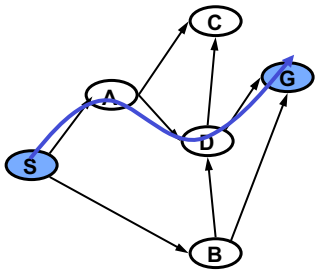
- Plan complex sequences of actions
- Schedule tight resources
- Monitor and diagnose behavior
- Repair or reconfigure hardware.

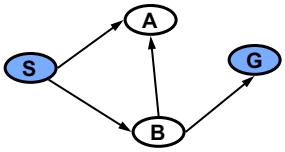
⇨formulate as state space search.

Formalizing Graph Search

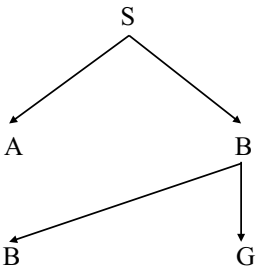
Input: A search problem SP = <g, S, G> where
- **graph g = <V, E>,**
- **start vertex S in V, and**
- **goal vertex G in V.**

Output: A simple path P = <S, v2, … G> in g from S to G.
(i.e., $\langle v_i, v_{i+1} \rangle \in E$, and $v_i \neq v_j$ if $i \neq j$).

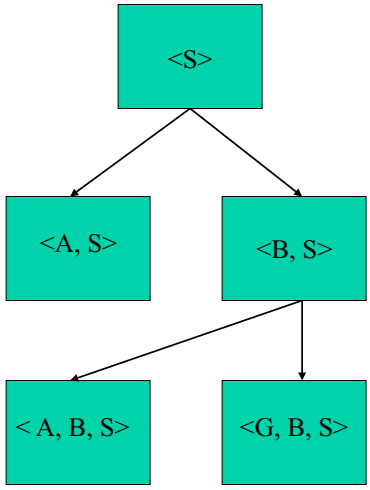




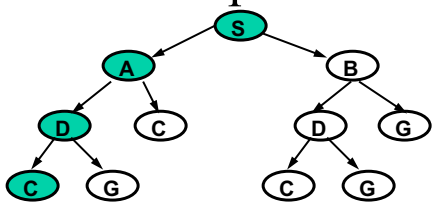
Graph Search is a Kind Of Tree Search



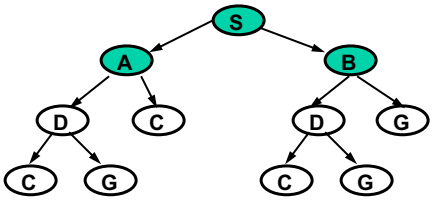
Graph Search is a Kind of State Space Search

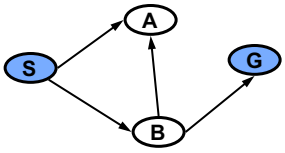


Solution: Depth First Search (DFS)



Solution: Breadth First Search (BFS)





Generate (Q)

<S>

Test

<A, S>

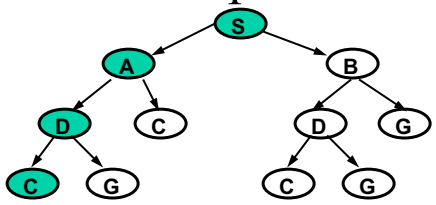
Visited

Pseudo Code For Simple Search

Let g be a Graph
S be the Start vertex of g
G be the Goal vertex of g.
Q be a list of simple partial paths in GR,

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) (its neighbors in g) not in Visited and create a one-step extension 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.**

Solution: Depth First Search (DFS)

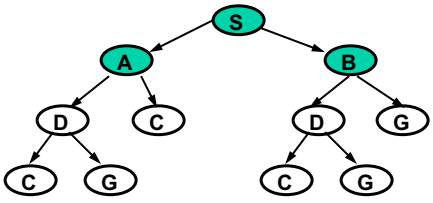


Depth-first:

Add path extensions to **front** of Q

Pick first element of Q

Solution: Breadth First Search (BFS)



Breadth-first:

Add path extensions to **back** of Q

Pick first element of Q

Outline

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

Elements of Algorithm Design

Description: (last Monday)

- Problem statement.
- Stylized pseudo code, sufficient to analyze and implement the algorithm.
- Implementation (last Wednesday).

Analysis: (today)

- Performance:
  - Time complexity:
    - how long does it take to find a solution?
  - Space complexity:
    - how much memory does it need to perform search?
- Correctness: (next Monday)
  - 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?

Performance Analysis

Analysis of run-time and resource usage:

- Helps to understand ***scalability***.
- Draws line between ***feasible*** and ***impossible***.
  - A function of program input.
  - Parameterized by input size.
  - Seeks upper bound.

Types of Analyses

Worst-case:

- T(n) = maximum time of algorithm on any input of size n.

Average-case:

- T(n) = expected time of algorithm over all inputs of size n.
  - Requires statistical distribution on inputs.

Best-case:

- T(n) = minimum time of algorithm on any input.

Analysis uses *Machine-independent* Time and Space

Performance depends on computer speed:
- Relative speed (run on same machine)
- Absolute speed (on different machines)

Big idea:
- Ignore machine-dependent constraints
- Look at growth of T(n) as $n \rightarrow \infty$

"Asymptotic Analysis"

Asymptotic notation

O-notation (upper bounds):

- $2n^2 = O(n^3)$

  means $2n^2 \leq cn^3$ *for sufficiently large* c & n

- $f(n) = O(g(n))$

  if there exists constants $c > 0$, $n_0 > 0$

  such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_o$.

Set definition of O-notation

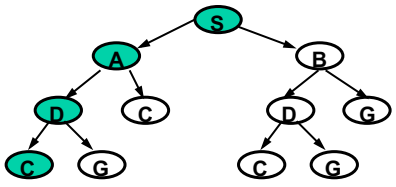
$O(n^3) = \{$ all functions bounded by $cn^3 \}$

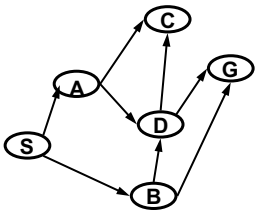
$2n^2 \in O(n^3)$

$O(g(n)) = \{f(n) \mid$ there exists constants
$c > 0, n_0 > 0$ such that
$0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_o\}$

Performance and Resource Usage

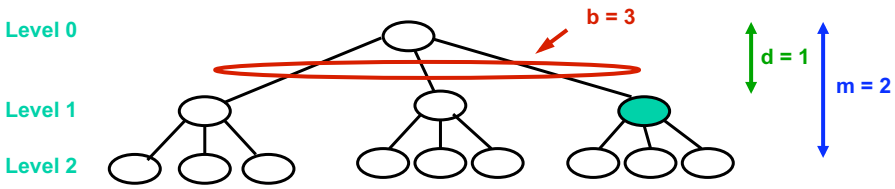
Which is better, depth-first or breadth-first?





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

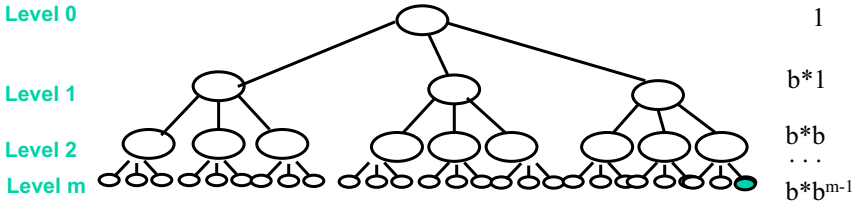
Analyzing Time and Space Complexity of Search in Terms of Trees



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

Worst Case Time for Depth-first

Worst case time T is proportional to number of nodes visited



$T_{dfs} = [b^m + \ldots b + 1]*c_{dfs}$ where c_{dfs} is time per node

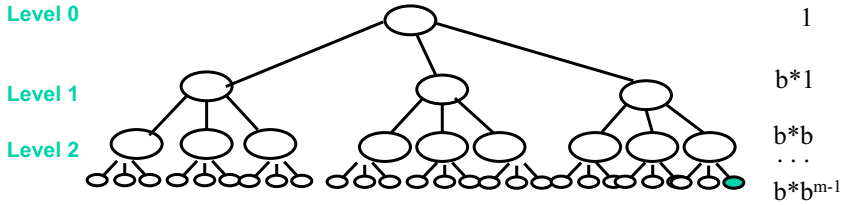
$b * T_{dfs} = [b^{m+1} + b^m + \ldots b + 0]*c_{dfs}$ Solve recurrence

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

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

Cost Using Order Notation

Worst case time T is proportional to number of nodes visited



Order Notation

- T(n) = O(e(n)) if T ≤ c * e for sufficiently large c & n

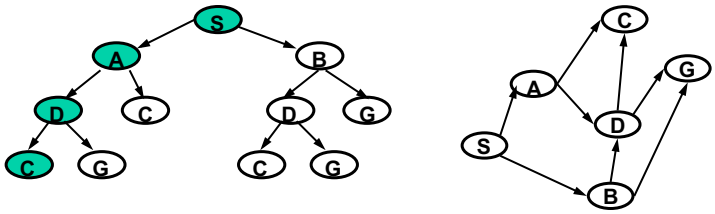
$T_{dfs} = [b^{m+1} - 1] / [b - 1] * c_{dfs}$

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

$\sim O(b^m)$ as b $\rightarrow \infty$ (used in some texts)

Performance and Resource Usage

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

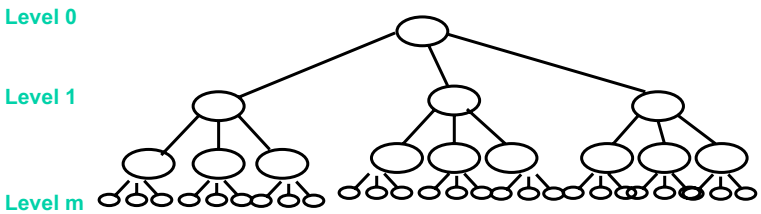


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$			
Breadth-first				

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

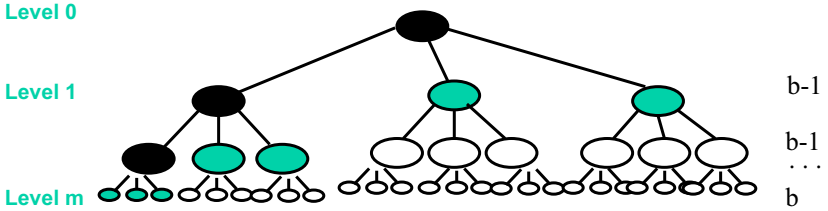
Worst case space S_{dfs} is proportional to maximum length of Q



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

Worst Case Space for Depth-first

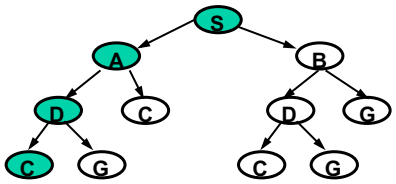
Worst case space S_{dfs} is proportional to maximuml length of Q

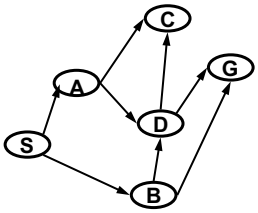


- If a node is queued its parent and siblings have been queued, and its parent dequeued.
  $S_{dfs} \geq [(b-1)*m+1] * c_{dfs}$ where c_{dfs} is space per node.
- At most one sibling of a node has its children queued.
  ➔ $S_{dfs} = [(b-1)*m+1] * c_{dfs}$
- $S_{dfs} = O(b*m)$ + add visited list

Performance and Resource Usage

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



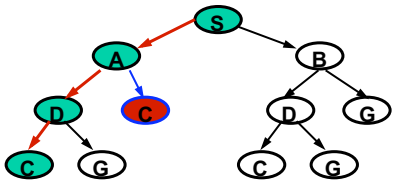


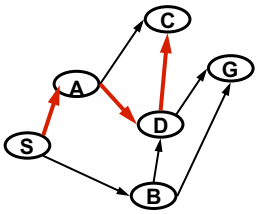
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	b*m		
Breadth-first				

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

Performance and Resource Usage

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



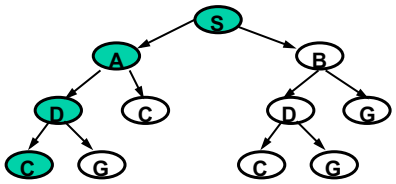


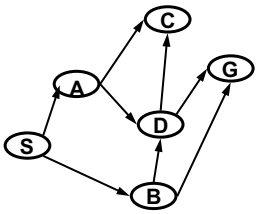
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	b*m	No	
Breadth-first				

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

Performance and Resource Usage

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



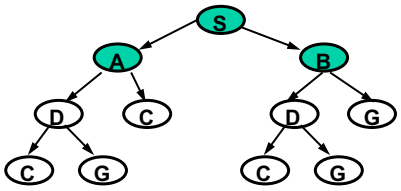


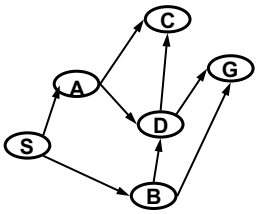
Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	b*m	No	Yes for finite graph
Breadth-first				

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

Performance and Resource Usage

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



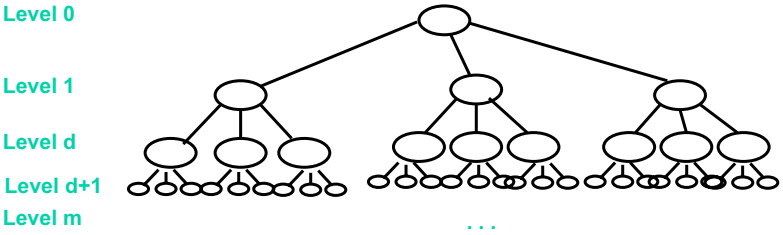


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	b*m	No	Yes for finite graph
Breadth-first				

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

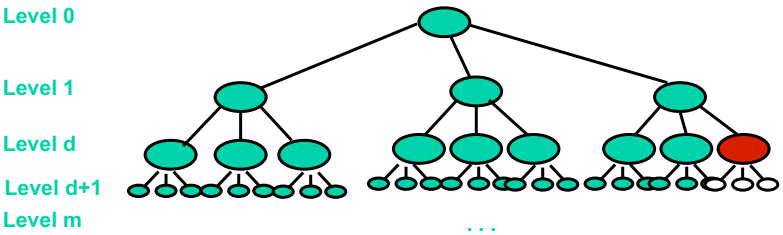
Worst case time T is proportional to number of nodes visited



Consider case where solution is at level d (absolute worst is m):

Worst Case Time for Breadth-first

Worst case time T is proportional to number of nodes visited



1
b
. . .
b^d
b^{d+1} - b

Consider case where solution is at level d (absolute worst is m):

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

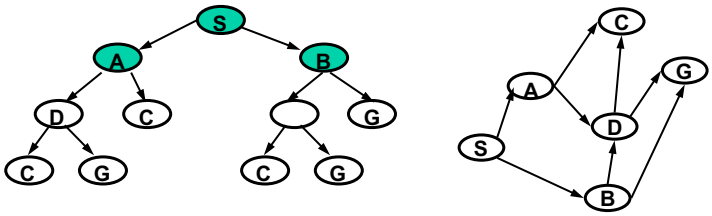
$= [b^{d+2} - b^2 + b - 1] / [b - 1] * c_{bfs}$

$= O[b^{d+2}]$

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

Performance and Resource Usage

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

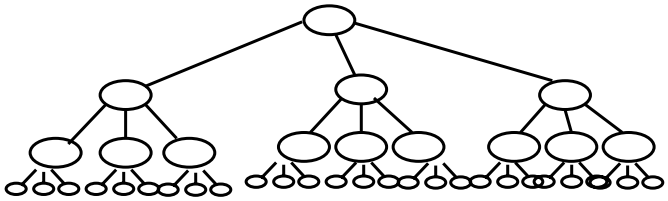
Worst case space S_{dfs} is proportional to maximum length of Q

Level 0

Level 1

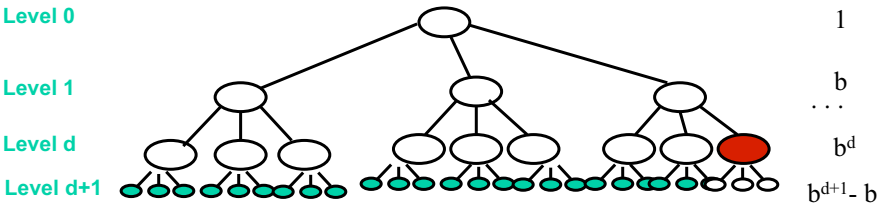
Level d

Level d+1



Worst Case Space for Breadth-first

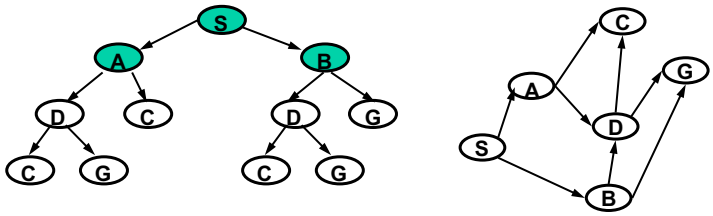
Worst case space S_{dfs} is proportional to maximum length of Q



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

Performance and Resource Usage

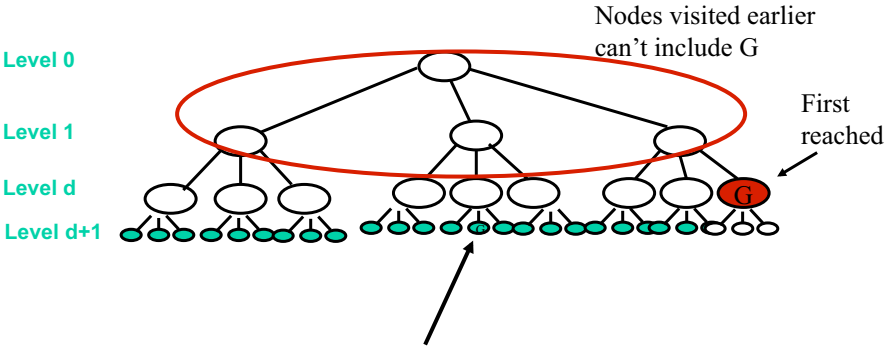
Which is better, depth-first or breadth-first?



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

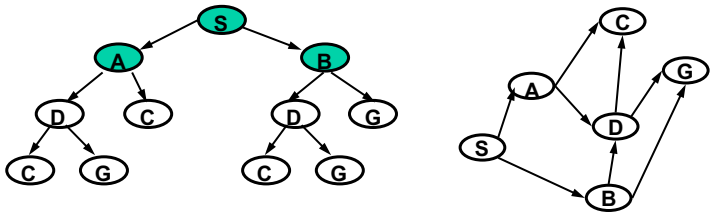
Breadth-first Finds Shortest Path



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

Performance and Resource Usage

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

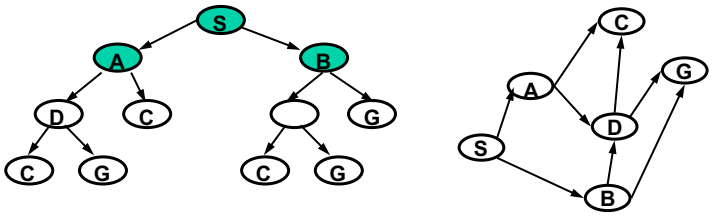


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	$b*m$	No	Yes for finite graph
Breadth-first	$\sim b^{d+1}$	b^{d+1}	Yes unit lngth	

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

Performance and Resource Usage

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

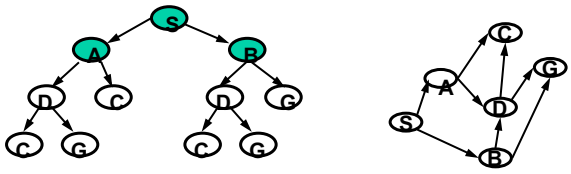


Search Method	Worst Time	Worst Space	Shortest Path?	Guaranteed to find path?
Depth-first	$\sim b^m$	b*m	No	Yes for finite graph
Breadth-first	$\sim b^{d+1}$	b^{d+1}	Yes unit lngth	Yes

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.
- Best-first can't expand to level m+1, just m.

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	$\sim b^m$	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

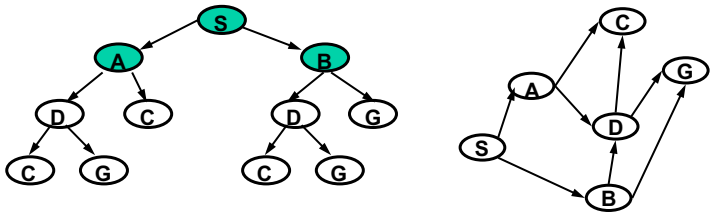
For best first search, which runs out first – time or memory?

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	~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

Outline

- Analysis
- Iterative deepening

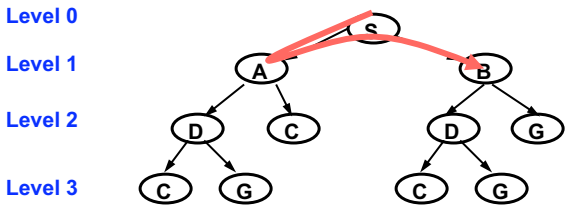
Iterative Deepening (IDS)

Idea:

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

➔ Search tree to depth 1, ….

called depth-limited search



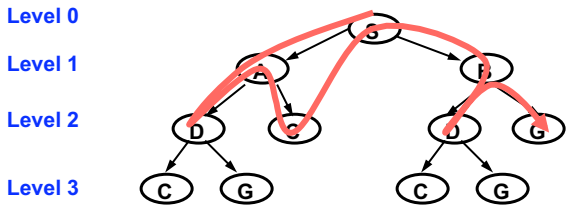
Iterative Deepening (IDS)

Idea:

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

➔ Search tree to depth 1, then 2, ….

called depth-limited search



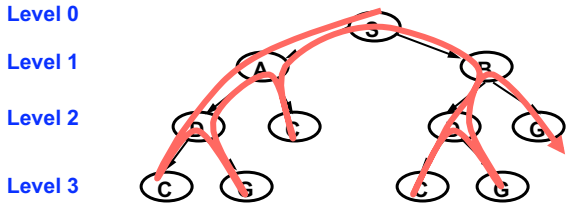
Iterative Deepening (IDS)

Idea:

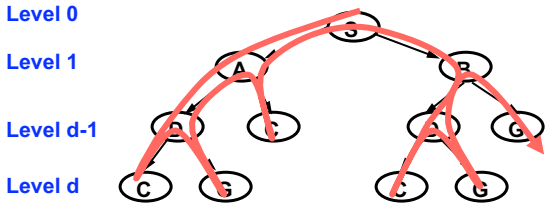
• Explore tree in breadth-first order, using depth-first search.

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

called depth-limited search



Speed of Iterative Deepening



Compare speed of BFS vs IDS:

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

- $T_{ids} = (d + 1)1 + (d)b + (d - 1)b^2 + \ldots 2b^{d-1} + b^d$
  $= [b^{d+2} + d(b-1) + 1)] / [b - 1]^2$

$\sim O(b^d)$ for lrg b

➔ Iterative deepening performs better than breadth-first!

Speed of Iterative Deepening

$T_{ids} = (d + 1)1 + (d)b + (d - 1)b^2 + \ldots 2b^{d-1} + b^d$
$bT_{ids} = (d + 1)b + (d)b^2 + (d - 1)b^3 + \ldots 2b^d + b^{d+1}$
$(b-1)T_{ids} = (d + 1) + b + b^2 + b^3 + \ldots b^d + b^{d+1}$
$(b-1)T_{ids} = d + \{[b^{d+2} + 1)] / [b - 1]\}$

$= [b^{d+2} + d(b-1) + 1)] / [b - 1]^2$

$\sim O(b^d)$ for lrg b

➔ Iterative deepening performs better than breadth-first!

Soundness and Completeness

(next Monday)

Soundness:

- All returned solutions are correct.
- Returns only simple paths from S to G.

Completeness:

- Always returns a solution if one exists.
- Returns a simple path from S to G whenever S is connected to G.

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, 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.

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010