Search

George Konidaris
gdk@cs.brown.edu

Fall 2021

(pictures: Wikipedia)


TELEVISION CAMERA
RANGE FINDER
DRIVE MOTOR
DRIVE WHEEL

Search

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

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

Basic to problem solving:

- ***How to take action to reach a goal?***

Search

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

Choices have consequences!

Search

Formalizing the problem statement …

- Problem can be in various *states*.
- Start in an *initial state*.
- Have some *actions* available.
- Each *action changes state*.
- Each action has a *cost*.
- Want to reach some *goal*, minimizing cost.

Happens in simulation.

Not web search.

Formal Definition

Set of states S

Start state $s \in S$

Set of actions A and action rules $a(s) \rightarrow s'$

Goal test $g(s) \rightarrow \{0, 1\}$

Cost function $C(s, a, s') \rightarrow \mathbb{R}^+$

So a search problem is specified by a tuple, (S, s, A, g, C).

Problem Statement

Find a sequence of actions $a_1, ..., a_n$
and corresponding states $s_1, ..., s_n$

… such that:

$s_0 = s$	start state
$s_i = a_i(s_{i-1}), i = 1, ..., n$	legal moves
$g(s_n) = 1$	end at the goal

while minimizing:

$$\sum_{i=1}^{n} C(s_{i-1}, a_i, s_i)$$

minimize sum of costs - *rational agent*

Example

<u>Sudoku</u>

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

States: all legal Sudoku boards.

Start state: a particular, partially filled-in, board.

Actions: inserting a *valid* number into the board.

Goal test: all cells filled and no collisions.

Cost function: 1 per move.

Example

States: airports, times.

Start state: TF Green at 5pm.

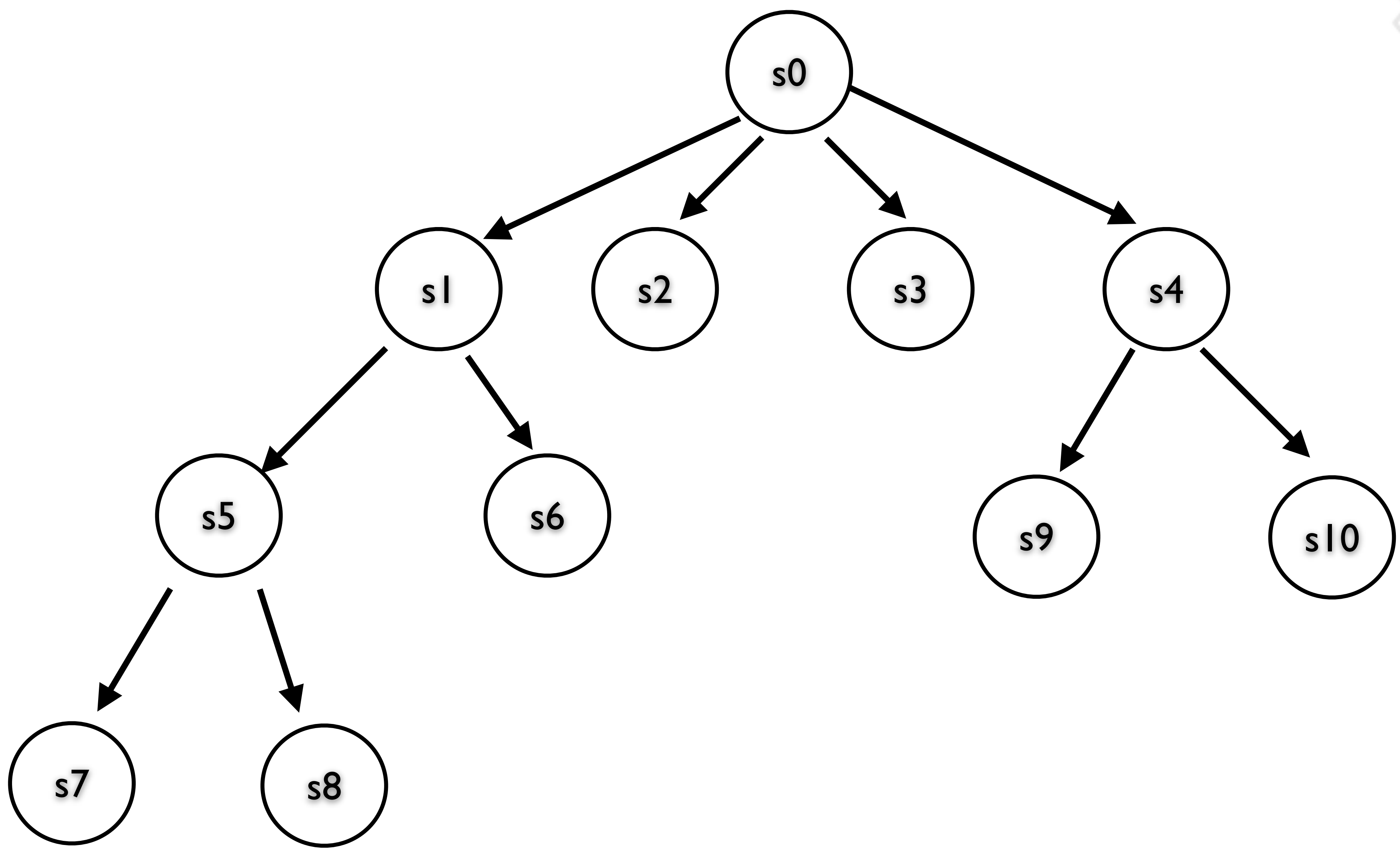
Actions: available flights from each airport after each time.

Goal test: reached Tokyo by midnight tomorrow.

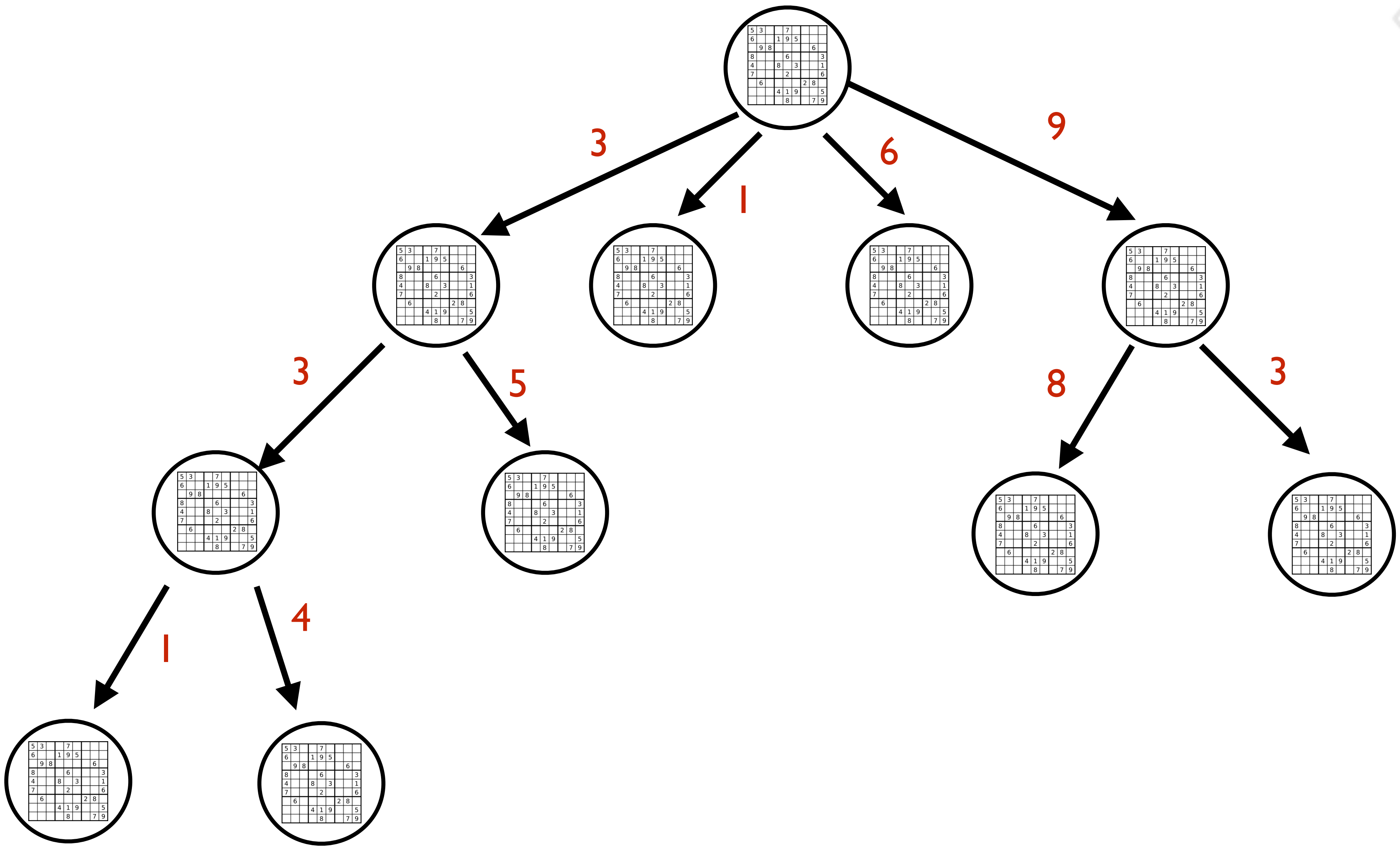
Cost function: time and/or money.

The Search Tree

Classical conceptualization of search.

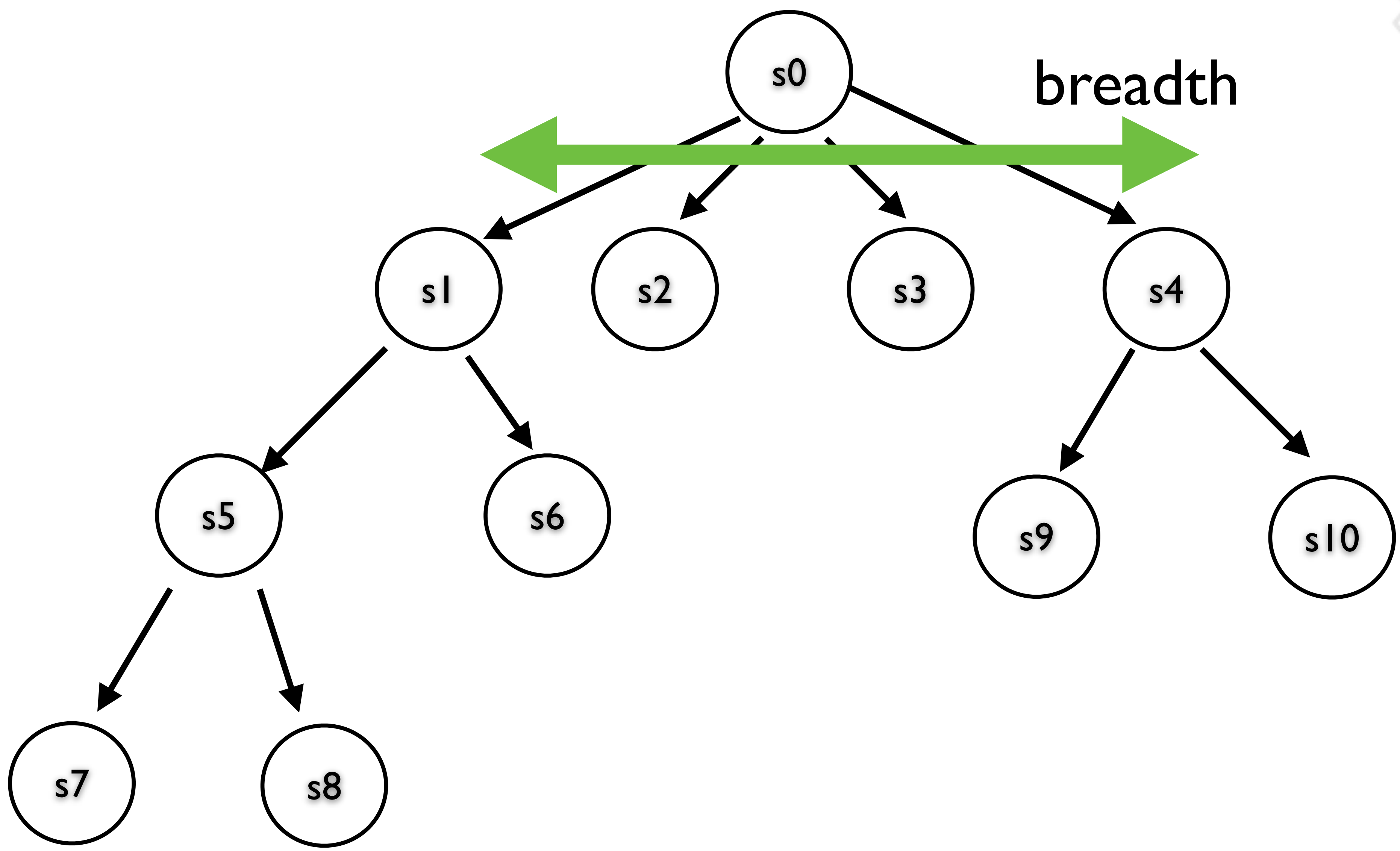


The Search Tree



Important Quantities

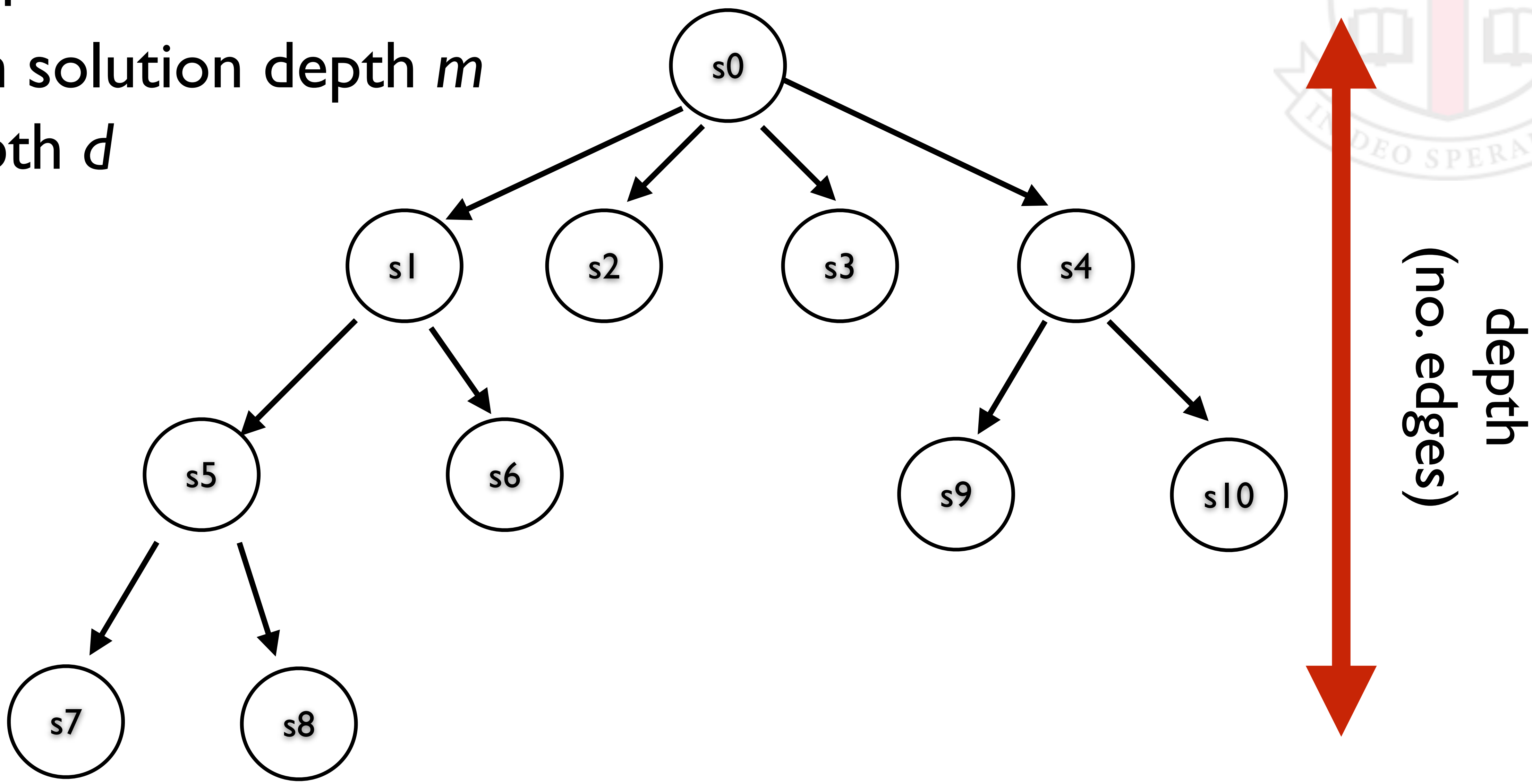
Branching factor (*breadth*)



The Search Tree

Depth

- min solution depth *m*
- depth *d*



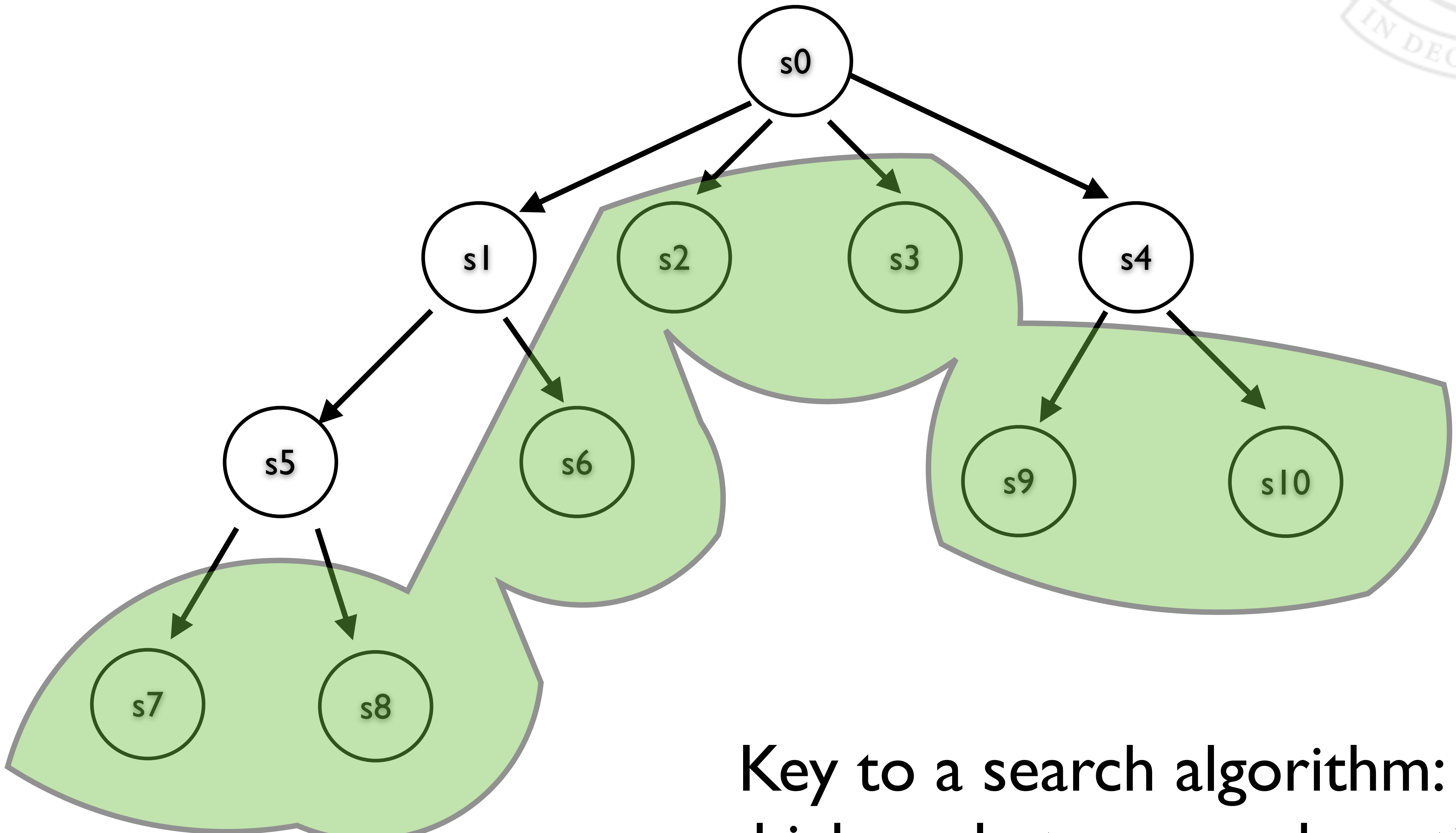
$O(b^d)$ leaves in a tree of breadth b, depth d.

$\sum_{i=0}^{d} b^i \in O(b^d)$ *total* nodes in the same tree

The Search Tree

Expand the tree one node at a time.

Frontier: set of nodes *in tree*, but not *expanded*.



Key to a search algorithm:
which node to expand next?

Searching

```
visited = {}
frontier = {s0}
goal_found = false

while not goal_found:
    node = frontier.next()
    frontier.del(node)

    if(g(node)):
        goal_found = true
    else:
        visited.add(node)
        for child in node.children:
            if(not (visited.contains(child) or frontier.contains(child)):
                frontier.add(child)
```

expand tree!

goal test

add children

How to Expand?

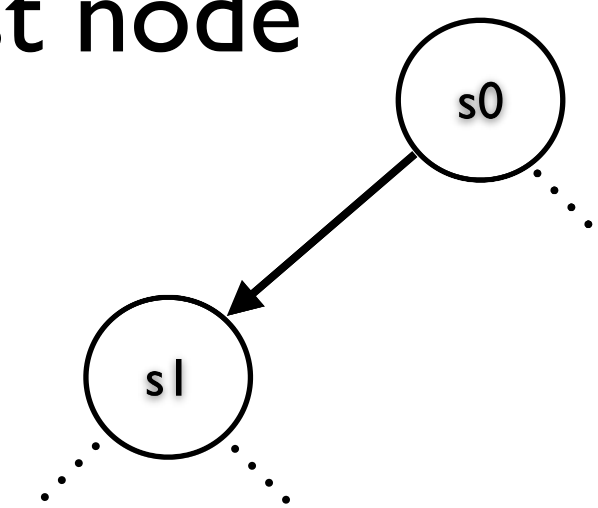
Uninformed strategy:
- nothing known about likely solutions in the tree.

What to do?
- Expand deepest node *(depth-first search)*
- Expand closest node *(breadth-first search)*

Properties
- Completeness
- Optimality
- Time Complexity *(total number of nodes visited)*
- Space Complexity *(size of frontier)*

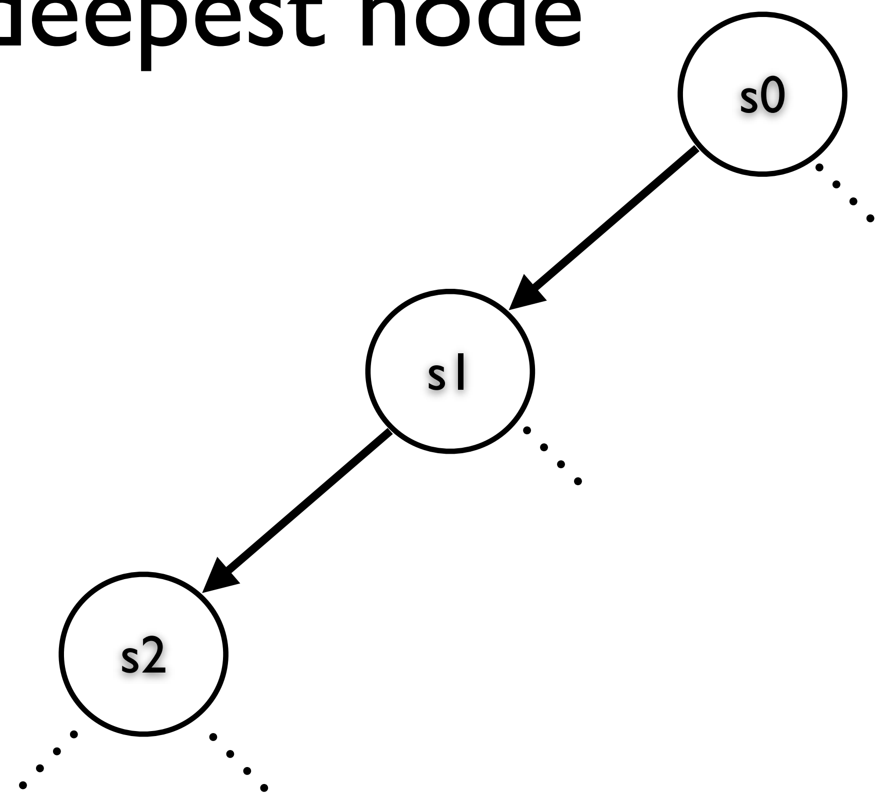
Depth-First Search

Expand deepest node



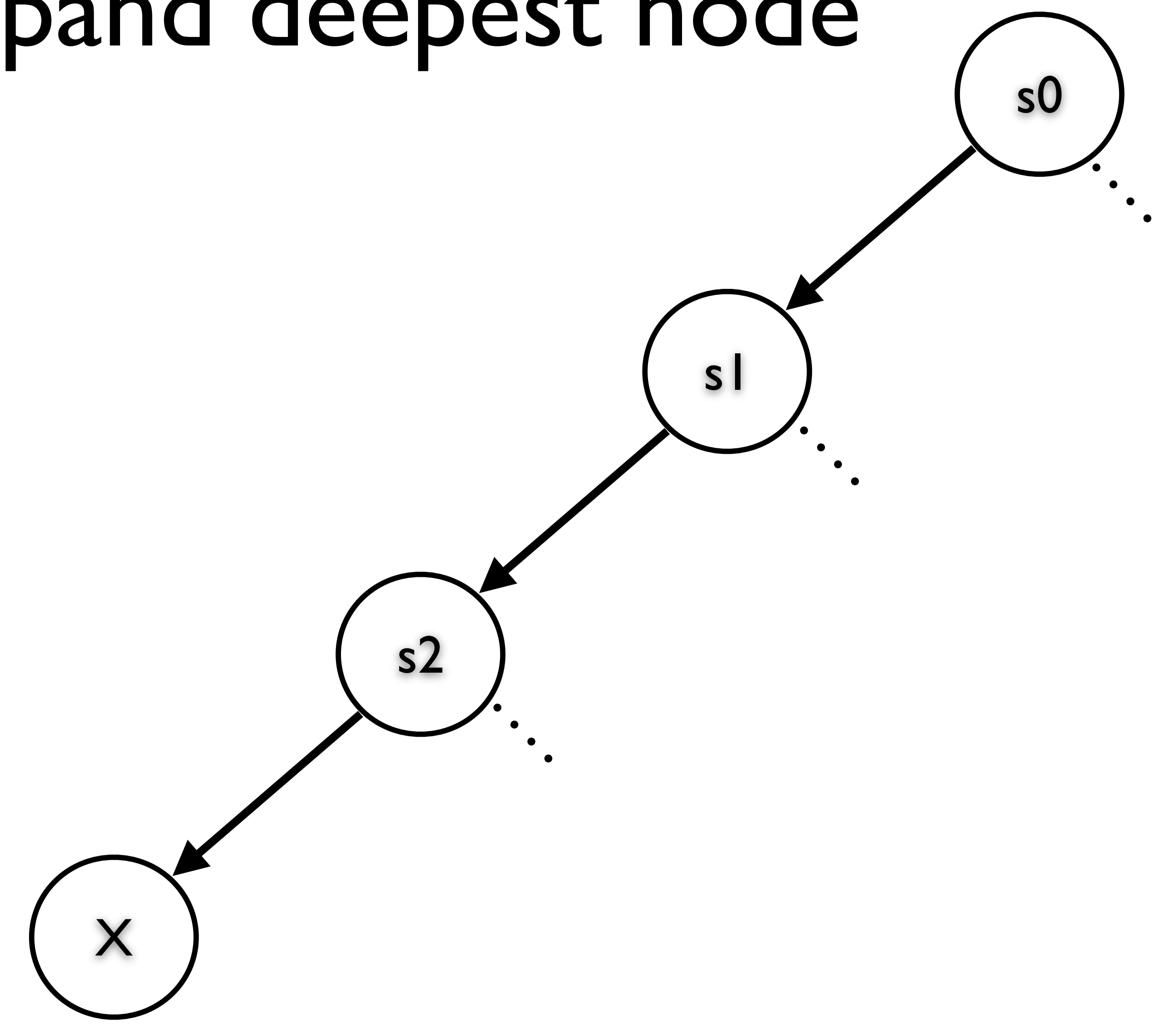
Depth-First Search

Expand deepest node



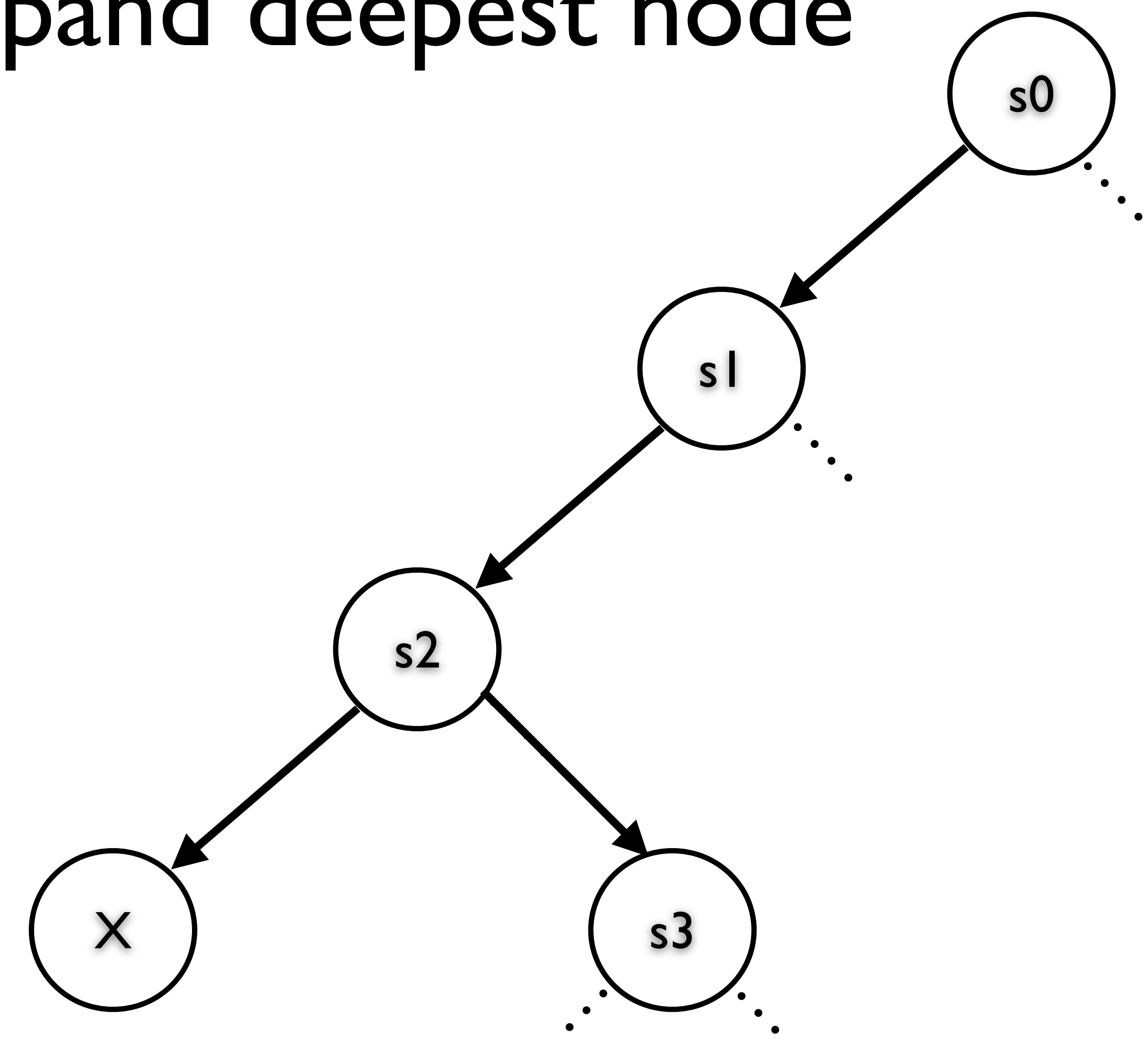
Depth-First Search

Expand deepest node



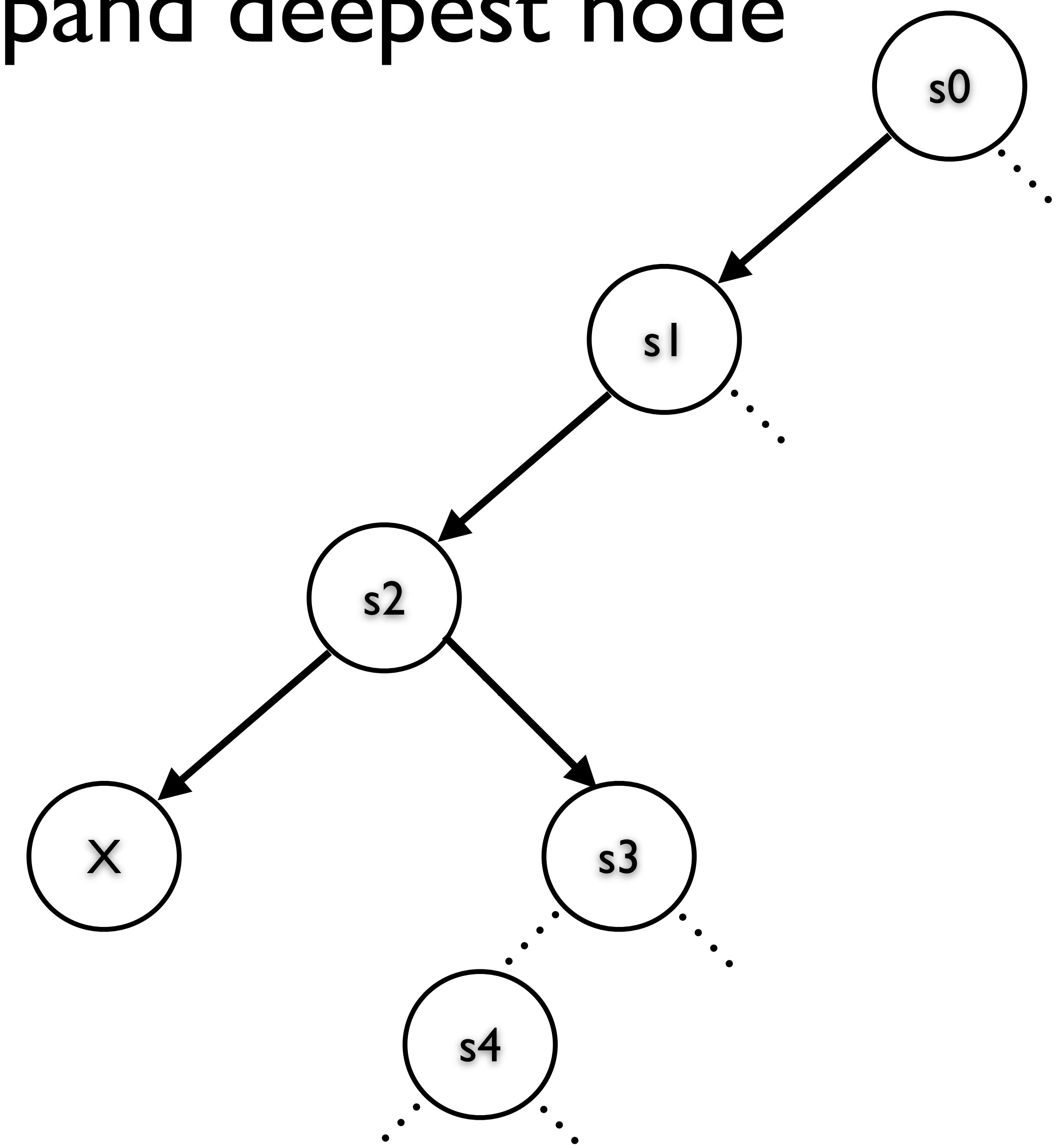
Depth-First Search

Expand deepest node

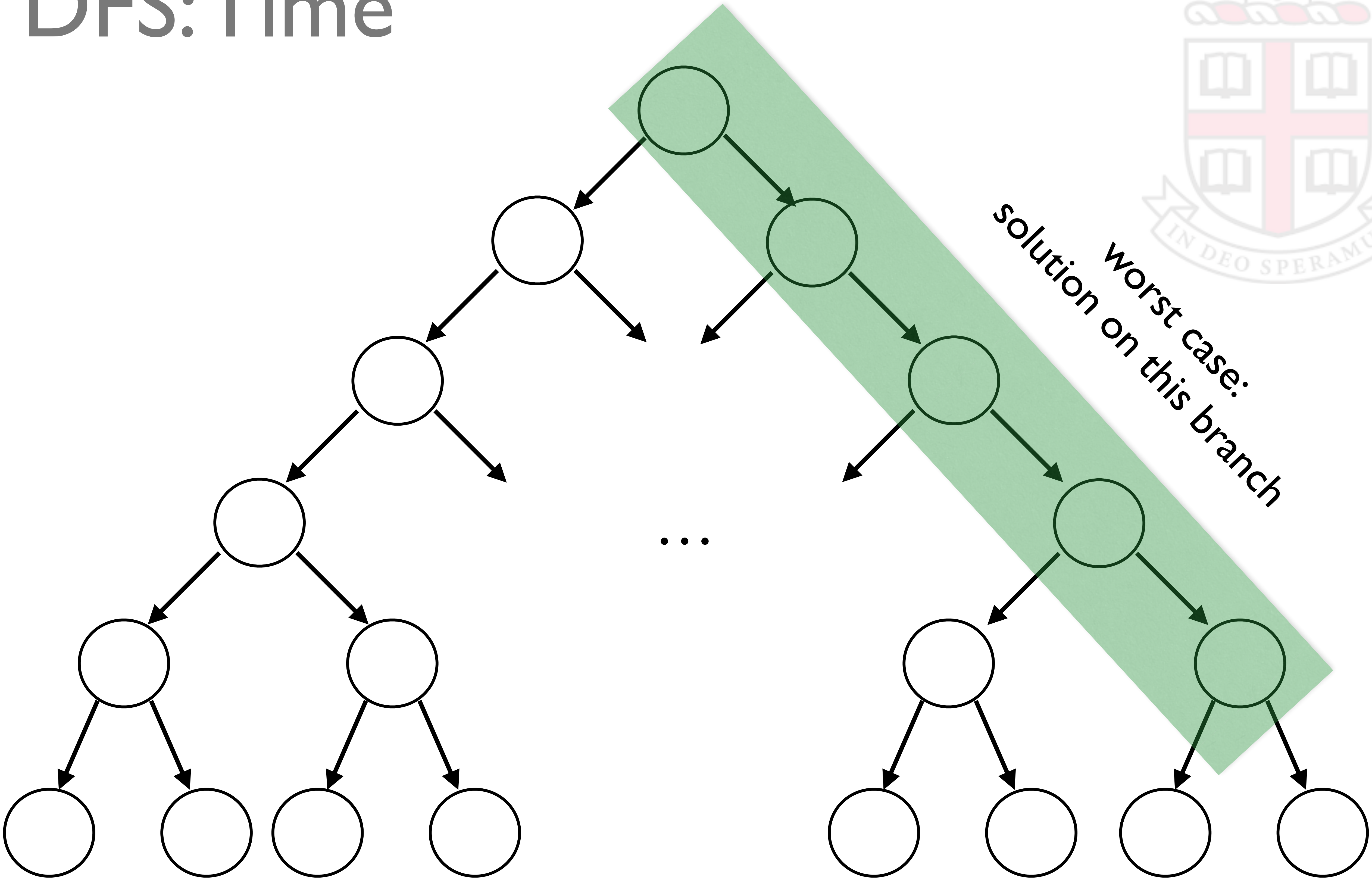


Depth-First Search

Expand deepest node

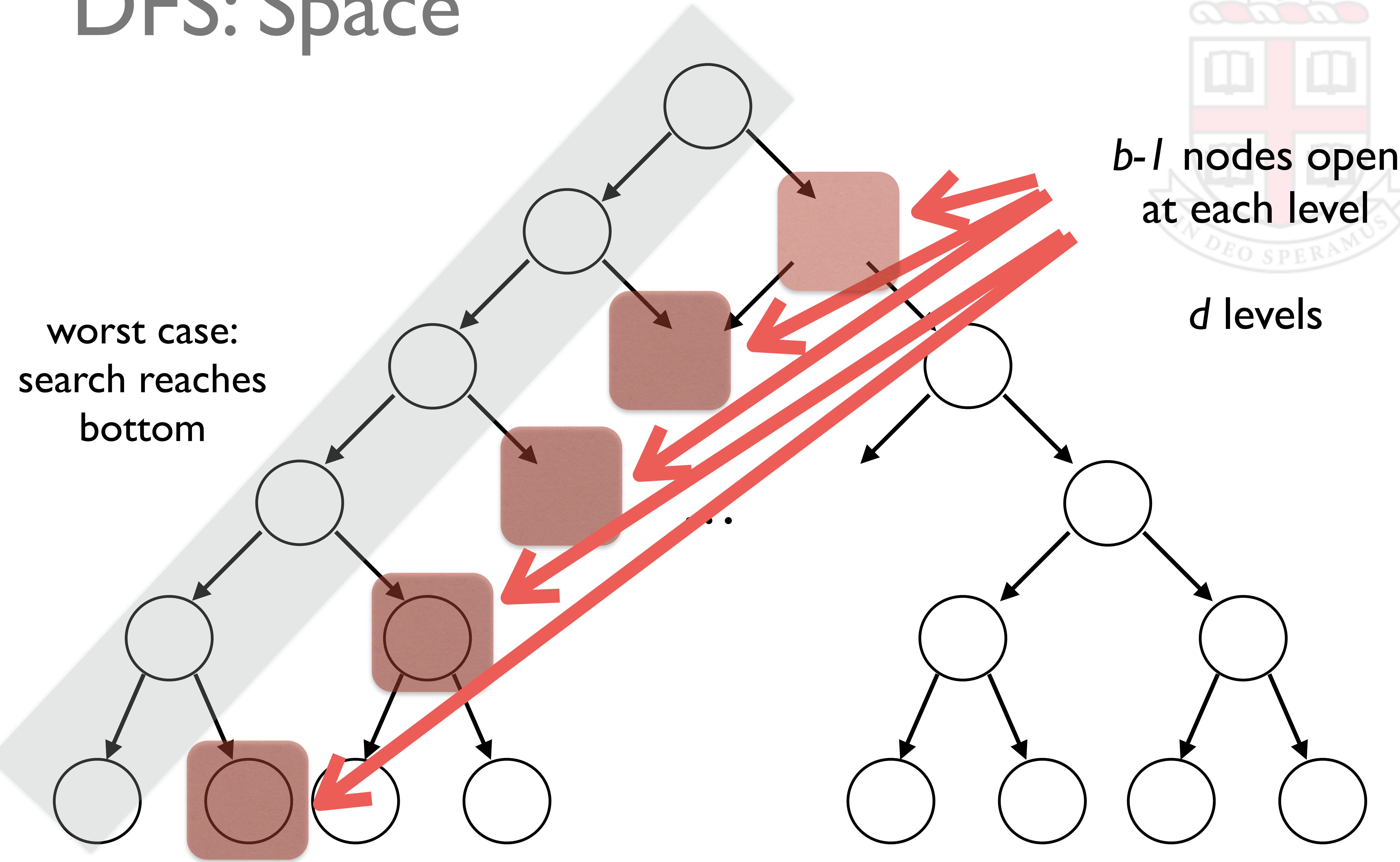


DFS: Time



$$O(b^d - b^{d-m}) = O(b^d)$$

DFS: Space



$$O((b-1)d) = O(bd)$$

Depth-First Search

Properties:

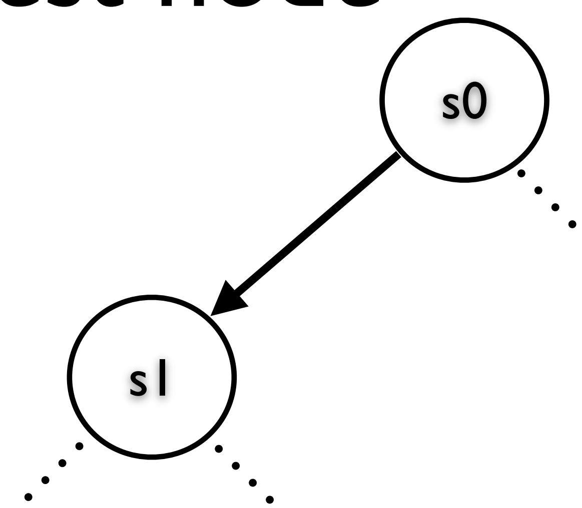
- Completeness: Only for finite trees.
- Optimality: No.
- Time Complexity: $O(b^d)$
- Space Complexity: $O(bd)$

Note that when reasoning about DFS, *m* is depth of found solution (*not necessarily min solution depth*).

The deepest node happens to be the one you most recently visited - easy to implement recursively OR manage frontier using LIFO queue.

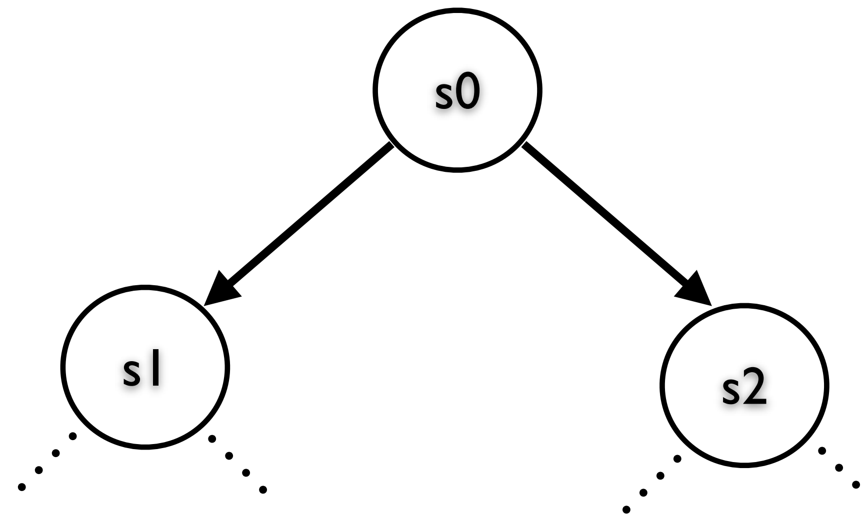
Breadth-First Search

Expand shallowest node



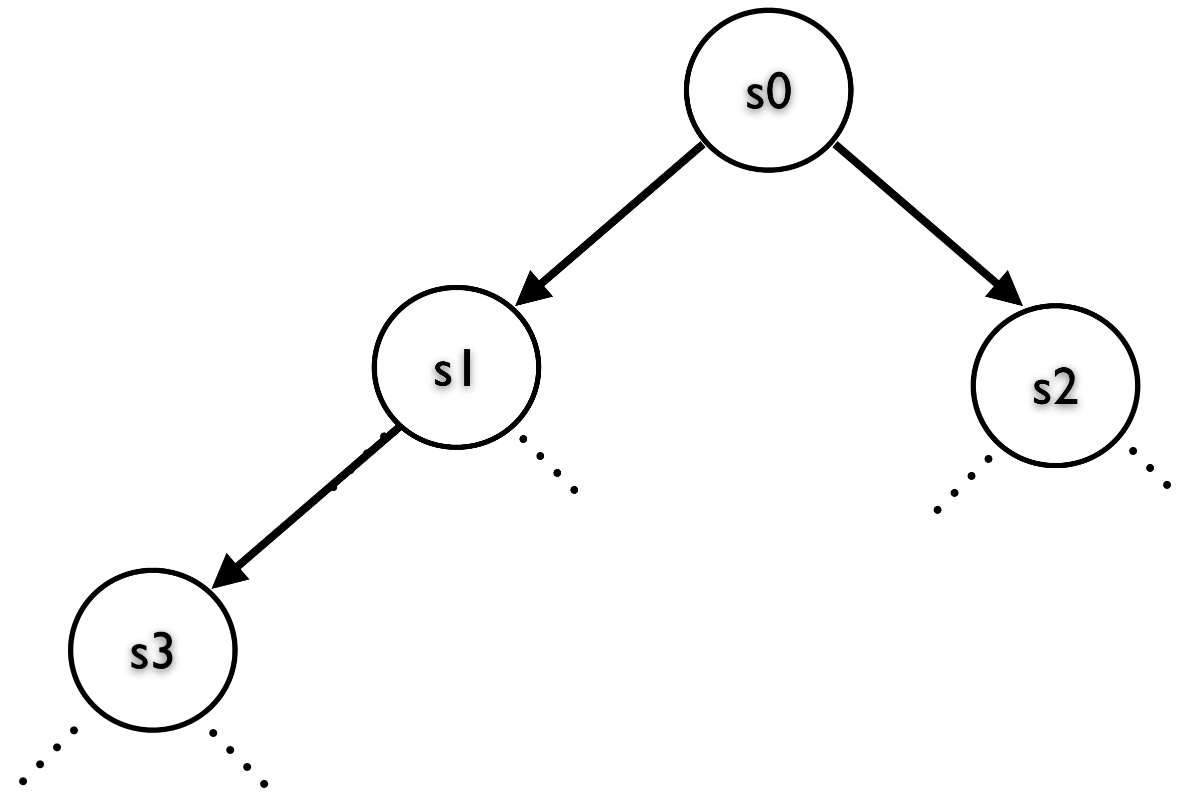
Breadth-First Search

Expand shallowest node



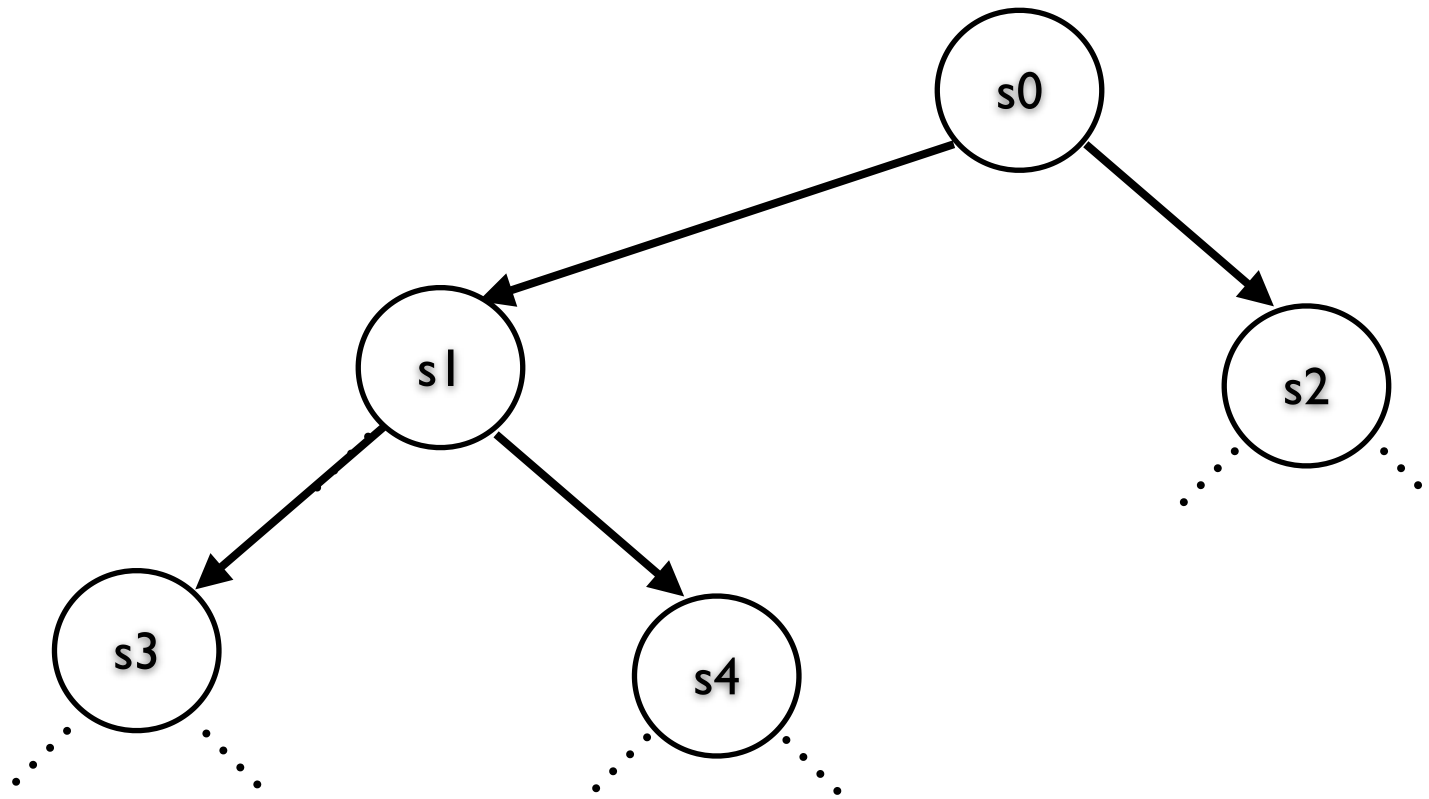
Breadth-First Search

Expand shallowest node



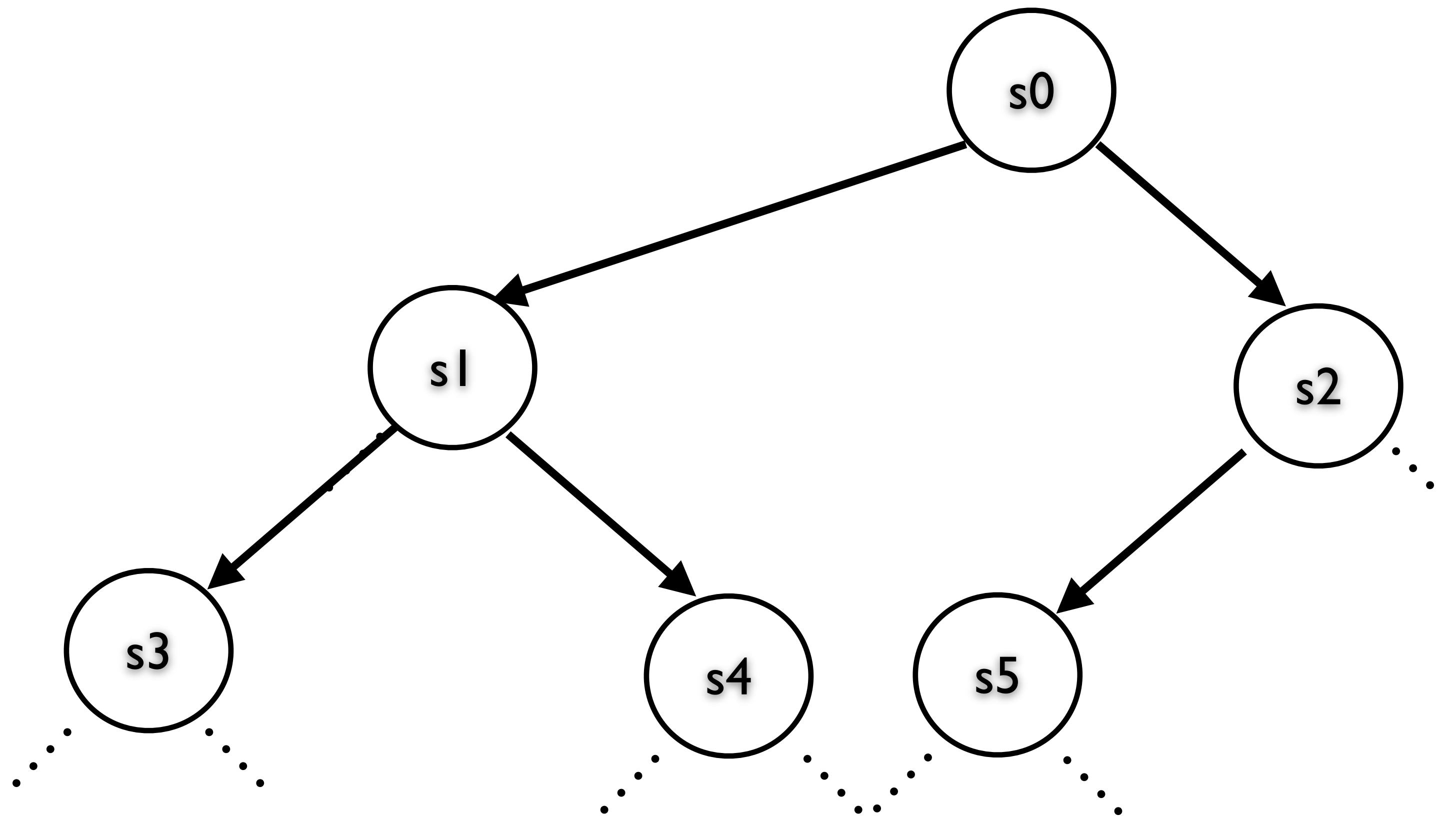
Breadth-First Search

Expand shallowest node

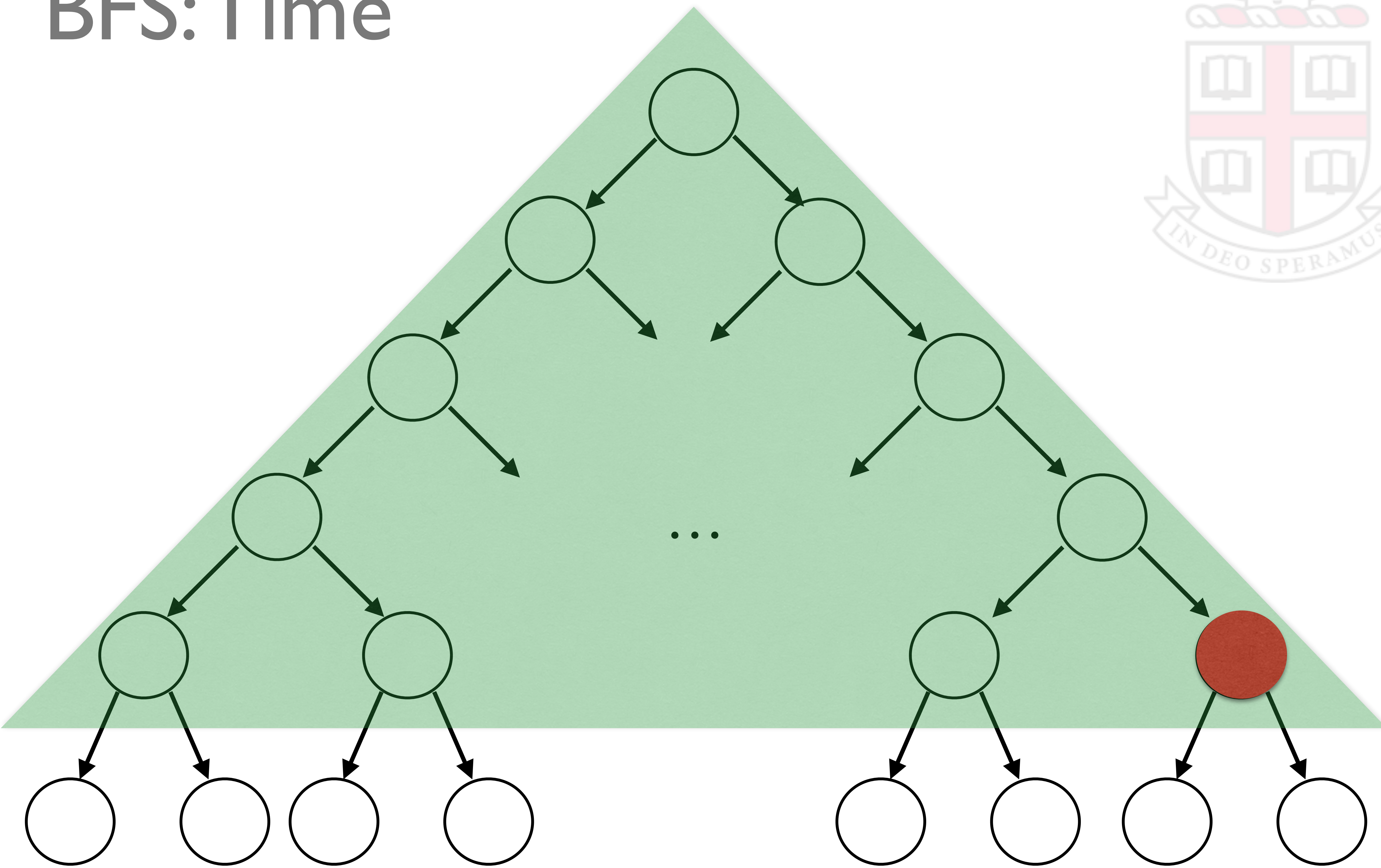


Breadth-First Search

Expand shallowest node

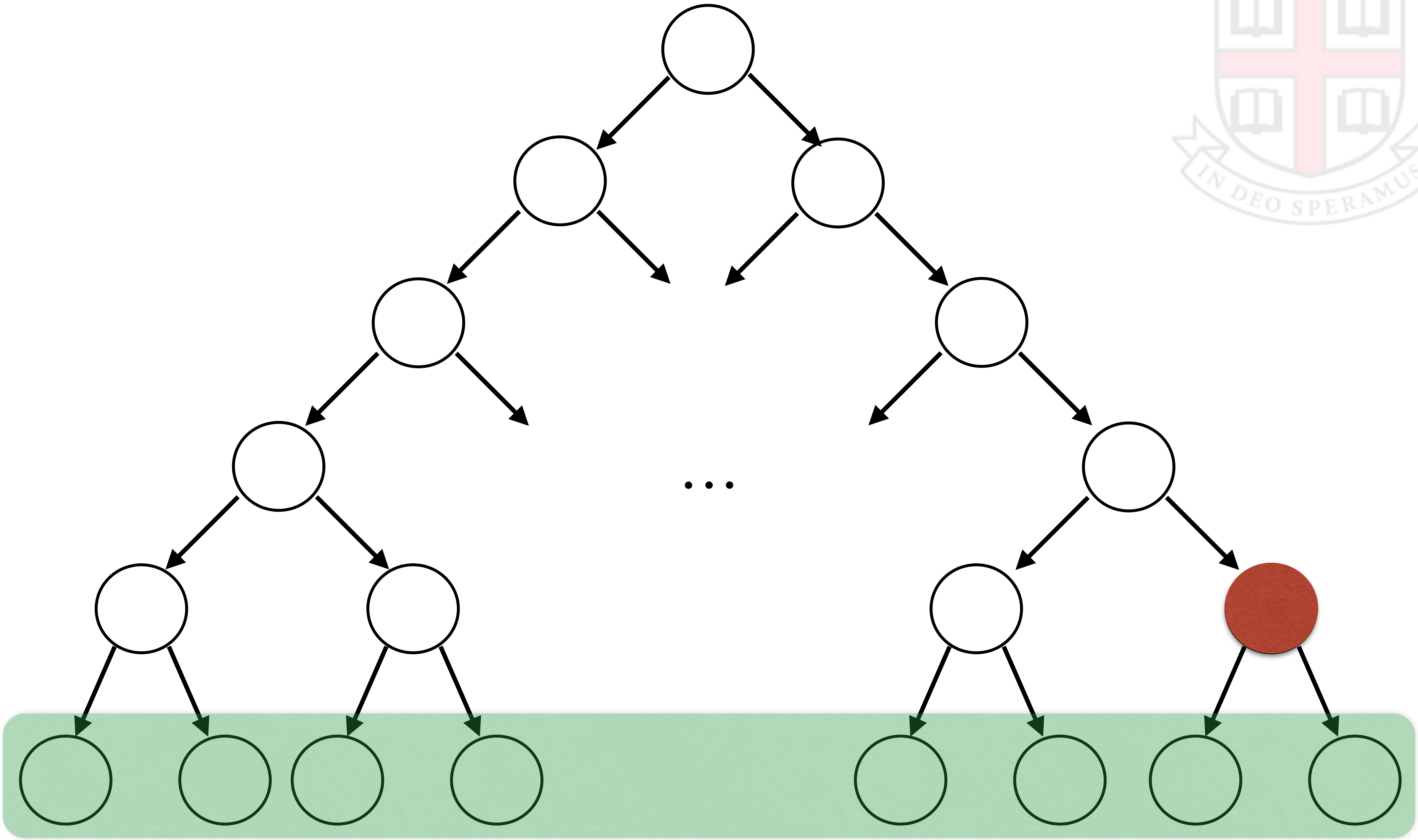


BFS: Time



$$O(b^m)$$

BFS: Space



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

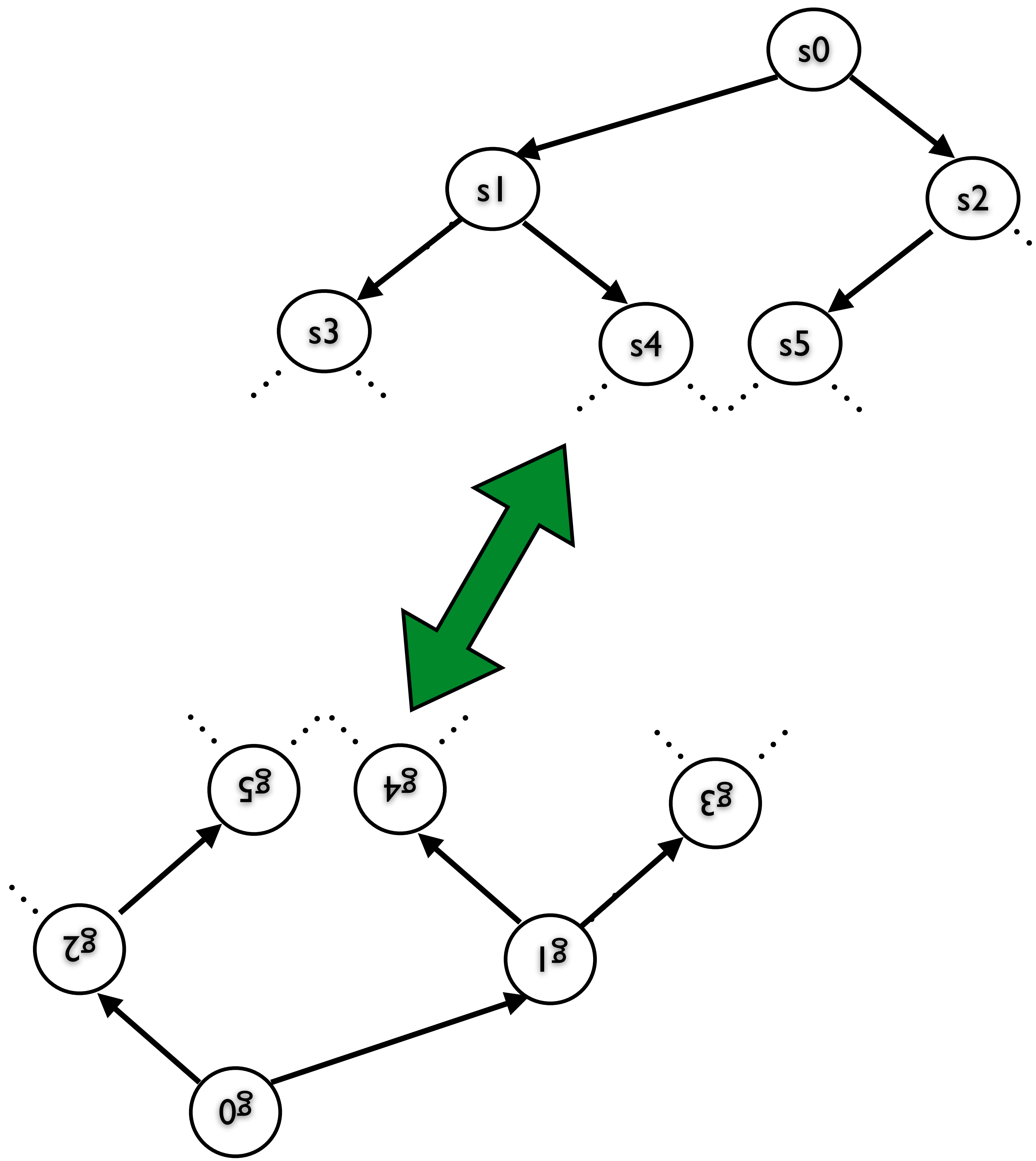
Breadth-First Search

Properties:

- Completeness: Yes.
- Optimality: Yes for *constant cost.*
- Time Complexity: $O(b^m)$
- Space Complexity: $O(b^{m+1})$

Manage frontier using FIFO queue.

Bidirectional Search



Bidirectional Search

Why?

$2 \times O(b^{\frac{d}{2}})$ is way less than $O(b^d)$

Extra requirements:

- Must be able to invert action rules.
- Sometimes easy, sometimes hard.
- Not always unique.
- Single solution.

When do you stop?

- Candidate solution when the frontiers intersect
- That solution may not be optimal - first must exhaust possible shortcuts.

Iterative Deepening Search

DFS: great memory cost - $O(bd)$ - but suboptimal solution.

BFS: optimal solution but horrible memory cost: $O(b^{m+1})$.

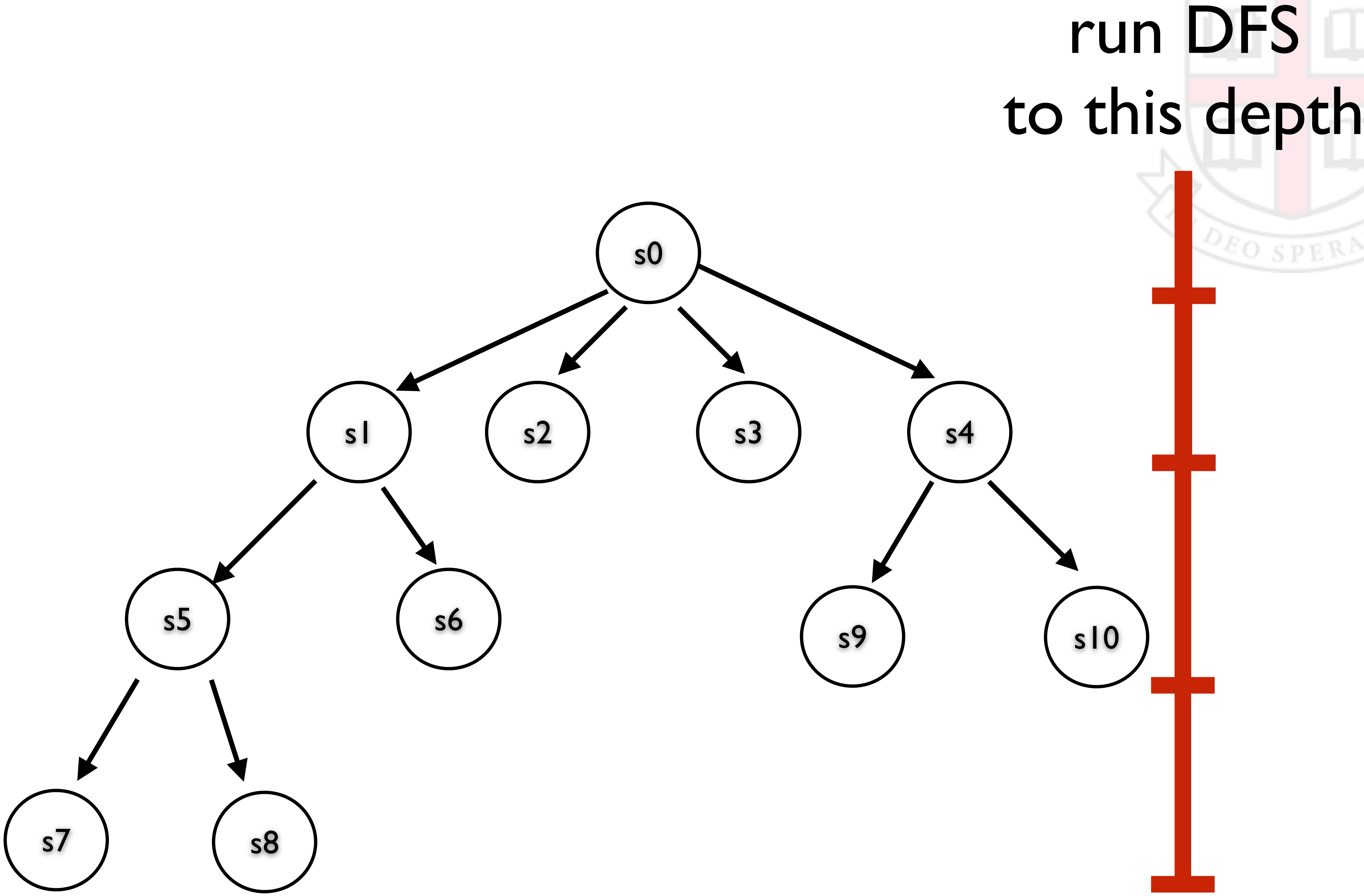
The core problems in DFS are a) *not optimal*, and b) *not complete* … because it fails to explore other branches.

Otherwise it's a very nice algorithm!

Iterative Deepening:

- Run DFS to a fixed depth *z*.
- Start at *z=0*. If no solution, increment *z* and rerun.

IDS



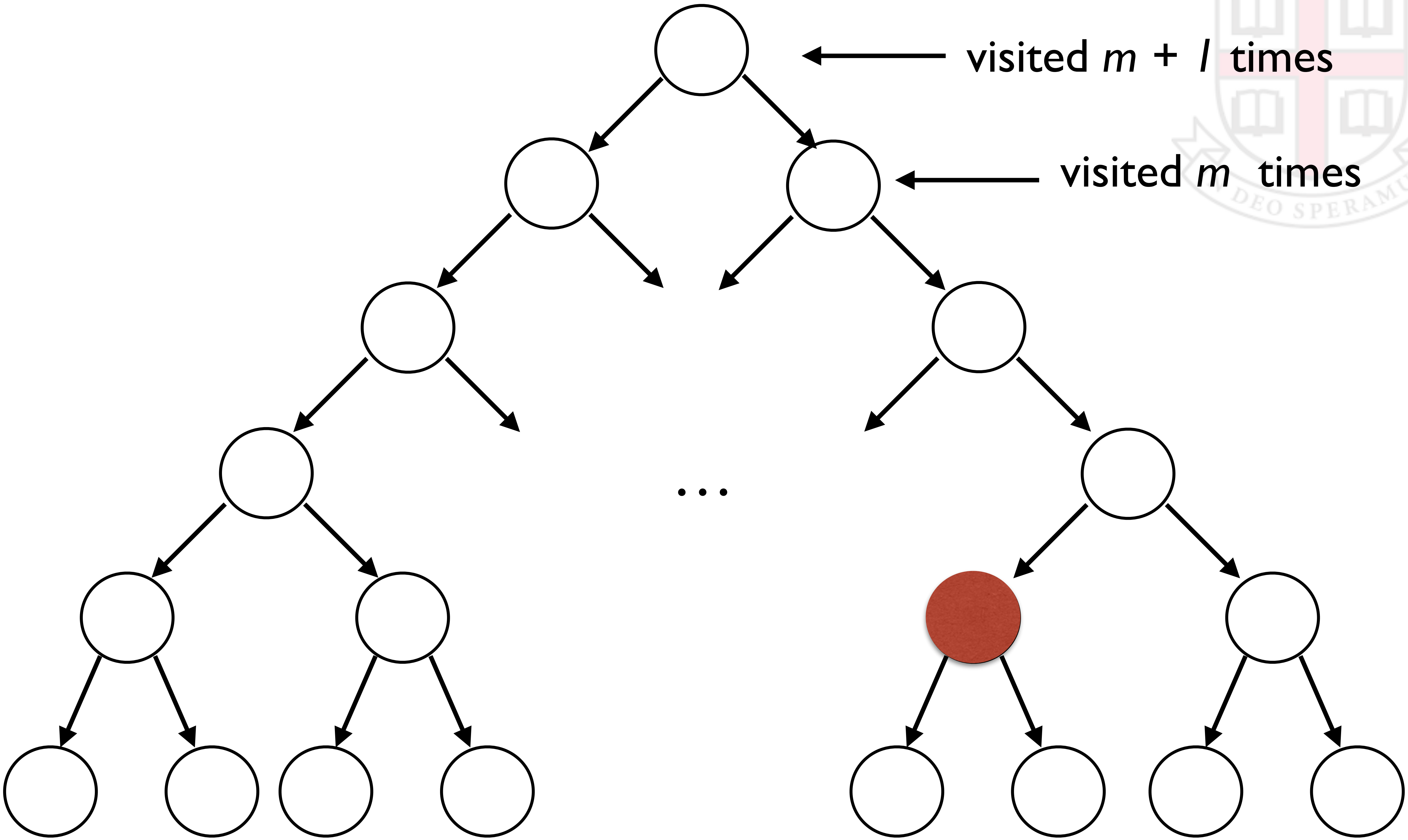
IDS

How can that be a good idea?
It duplicates work.

Optimal for constant cost! *Proof*?

Also!
- Low memory requirement (equal to DFS).
- Not many more nodes expanded than BFS. (About twice as many for binary tree.)

IDS



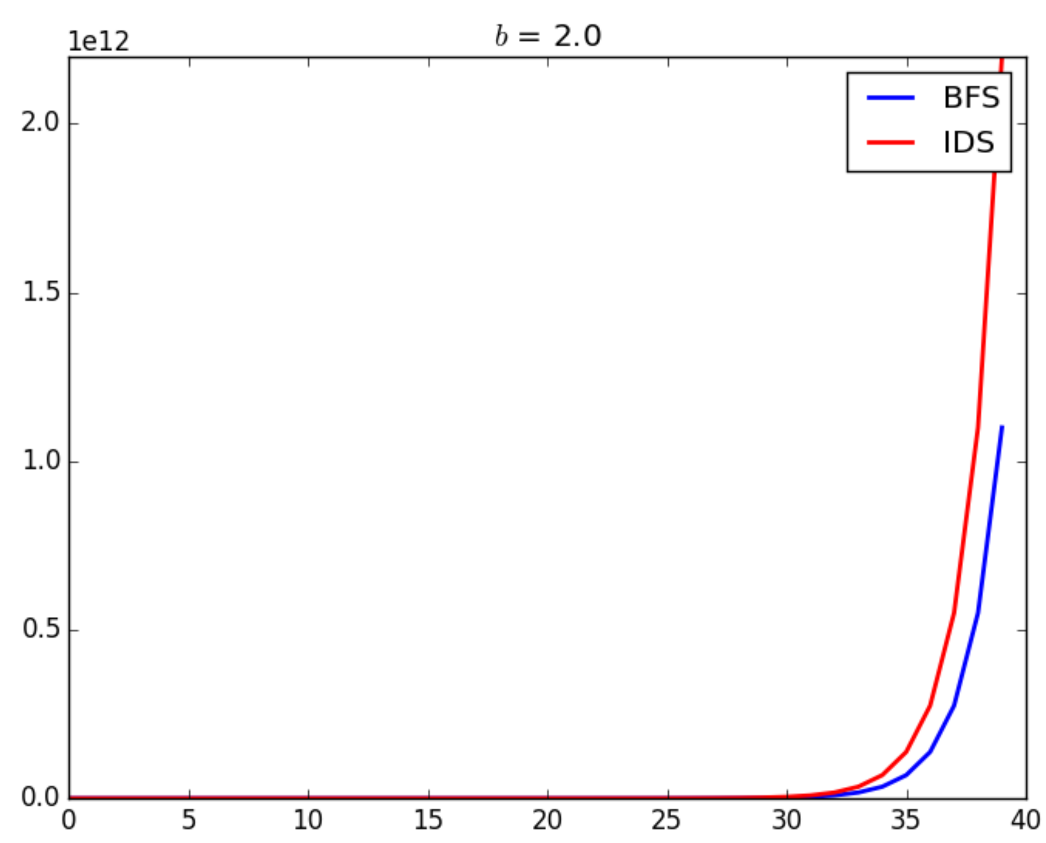
IDS

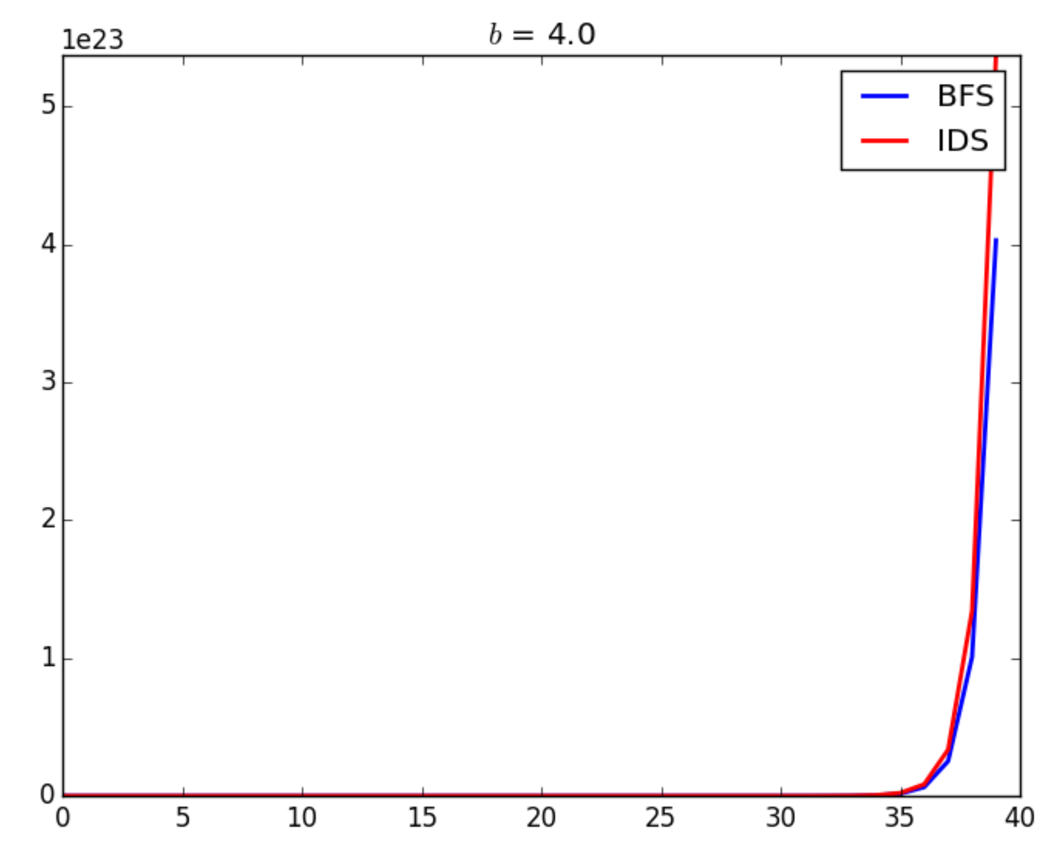
$$\sum_{i=0}^{m} b^i (m - i + 1) = \frac{b(b^{m+1} - m - 2) + m + 1}{(b-1)^2}$$

revisits

nodes at level *i*

BFS worst case: $\dfrac{b^{m+1} - 1}{b - 1}$





IDS

Key Insight:

- Many more nodes at depth $m+1$ than at depth m.

MAGIC.

"In general, iterative deepening search is the preferred uninformed search method when the state space is large and the depth of the solution is unknown." (R&N)

Uninformed Searches So Far

Simple strategy for choosing next node:

- Choose the shallowest one (**breadth-first**)
- Choose the deepest one (**depth-first**)

Neither guaranteed to find the least-cost path, in the case where action costs are not uniform.

What if we chose the one with *lowest cost?*

Uniform-Cost

Order the nodes in the frontier by ***cost-so-far***

- Cost from the start state to that node.

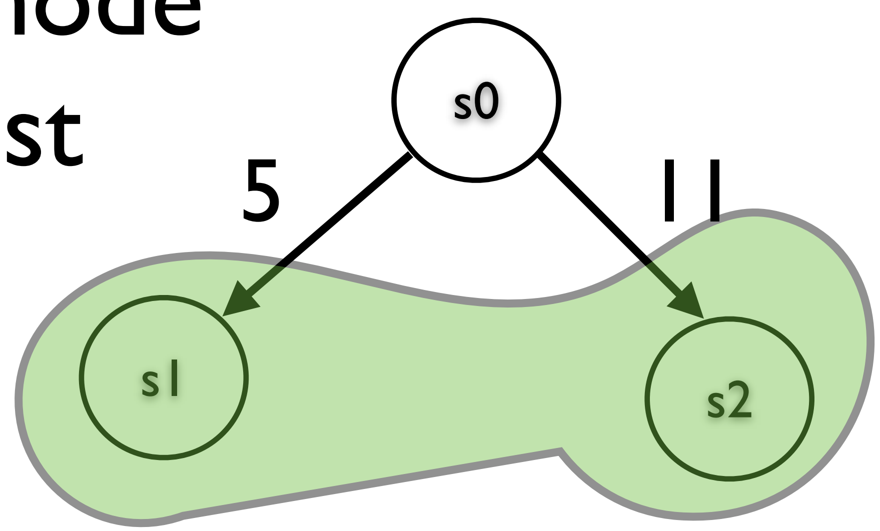
Open the next node with the smallest cost-so-far

- Optimal solution
- Complete (provided no negative costs)

Uniform-Cost

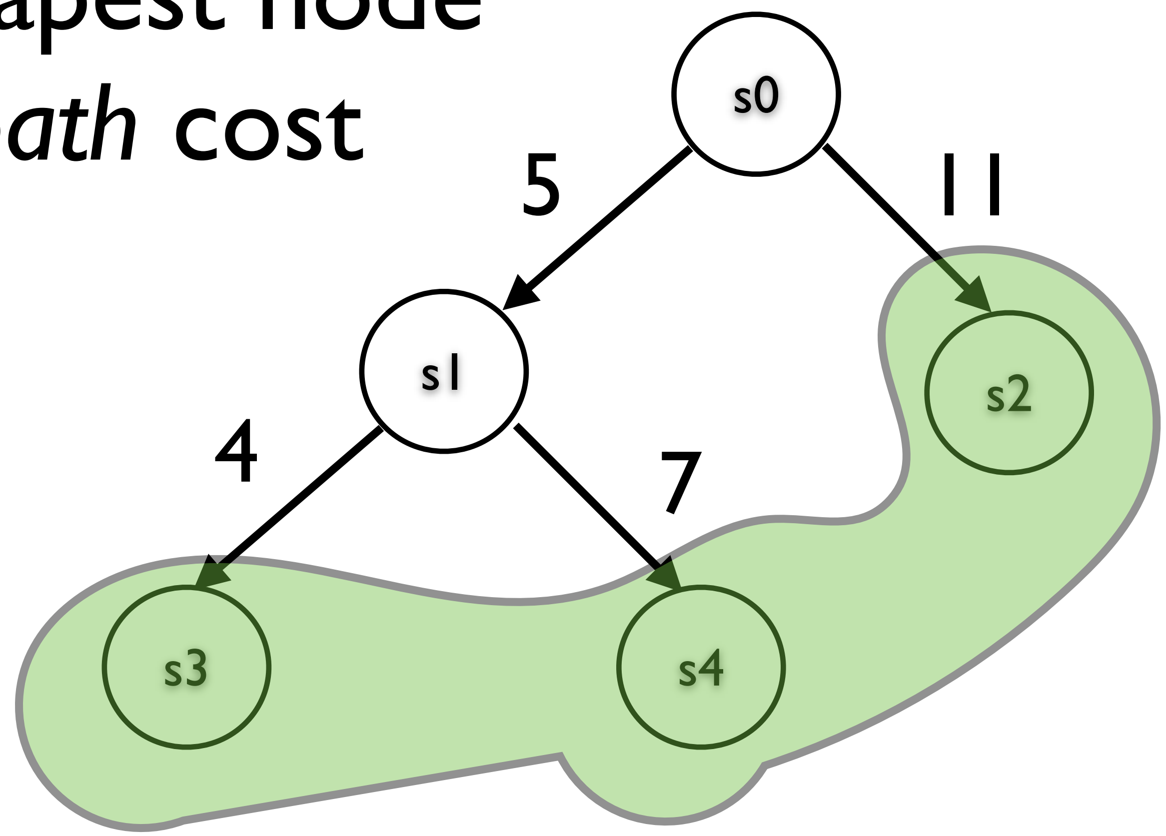
Expand cheapest node

Use *whole path* cost



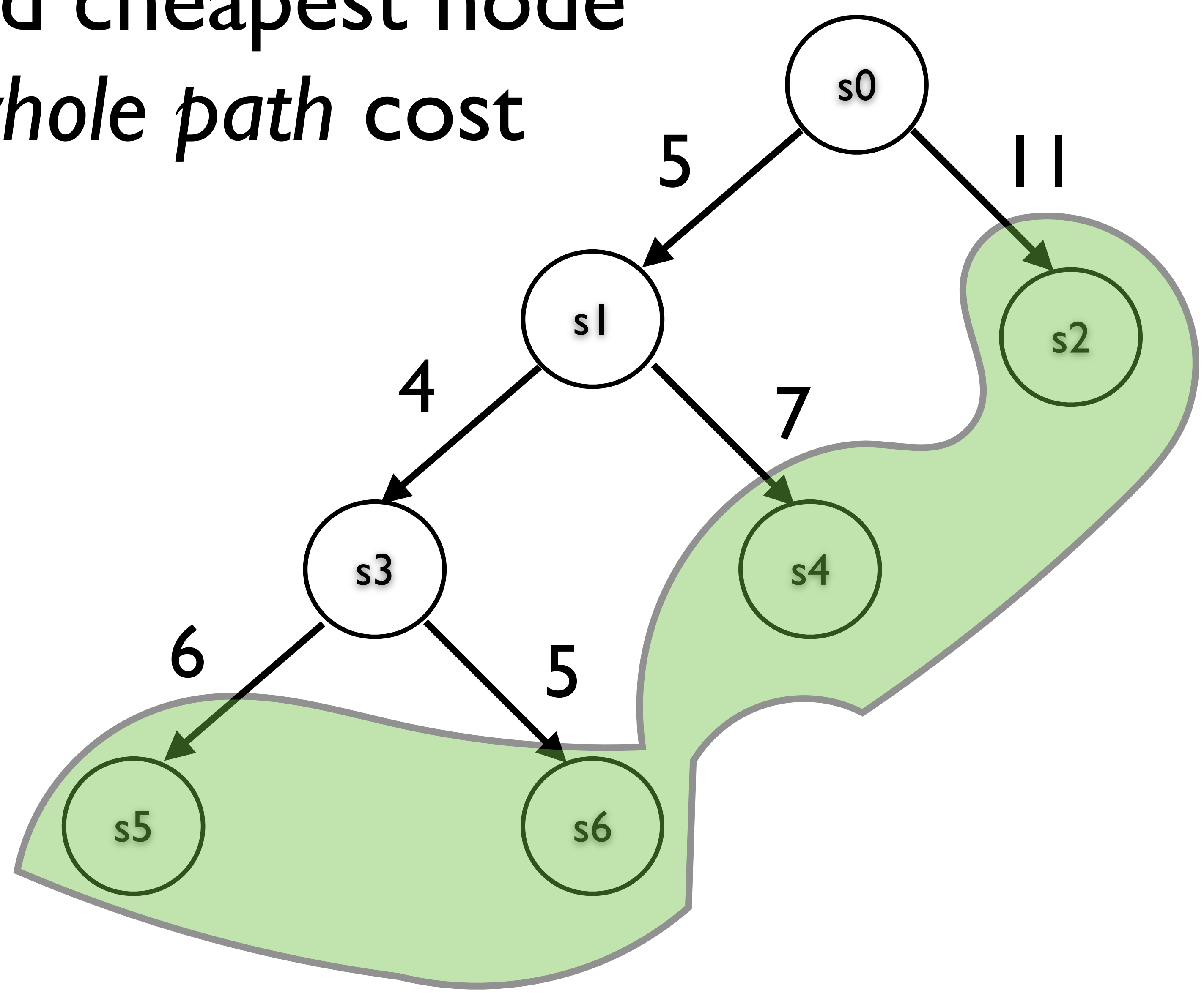
Uniform-Cost

Expand cheapest node
Use *whole path* cost



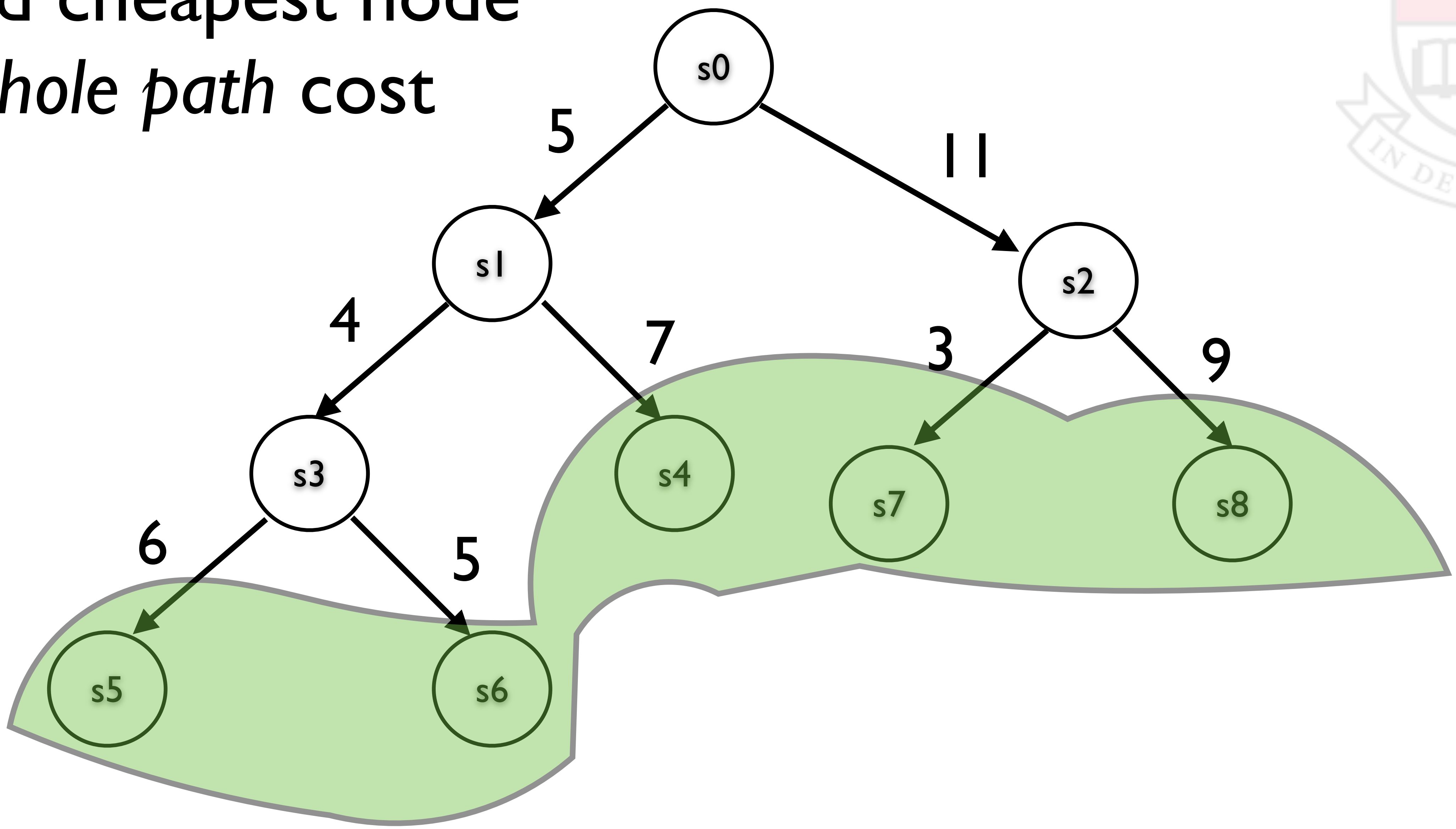
Uniform-Cost

Expand cheapest node
Use *whole path* cost



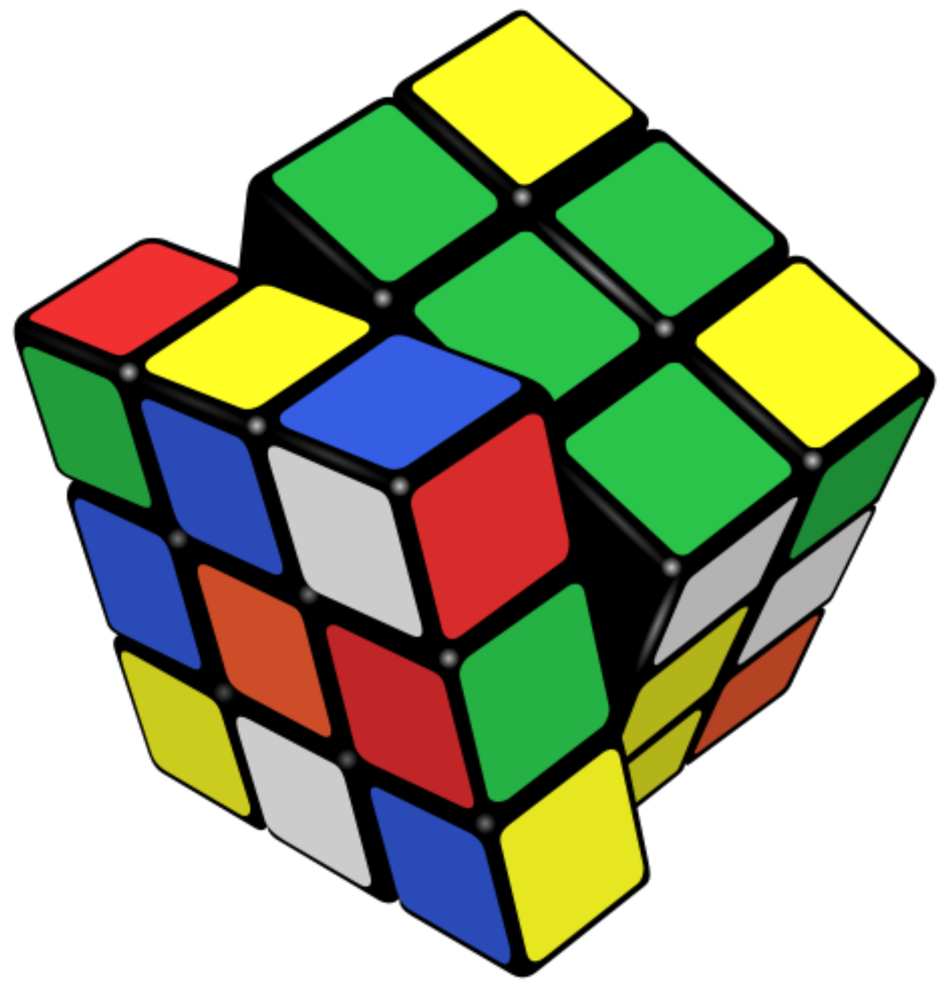
Uniform-Cost

Expand cheapest node
Use *whole path* cost



Informed Search

What if we *know something* about the search?



How should we include that knowledge?
In what form should it be expressed to be useful?

What Does Uniform Cost Suggest?

The *cost-so-far* tells us how much it cost to get to a node.

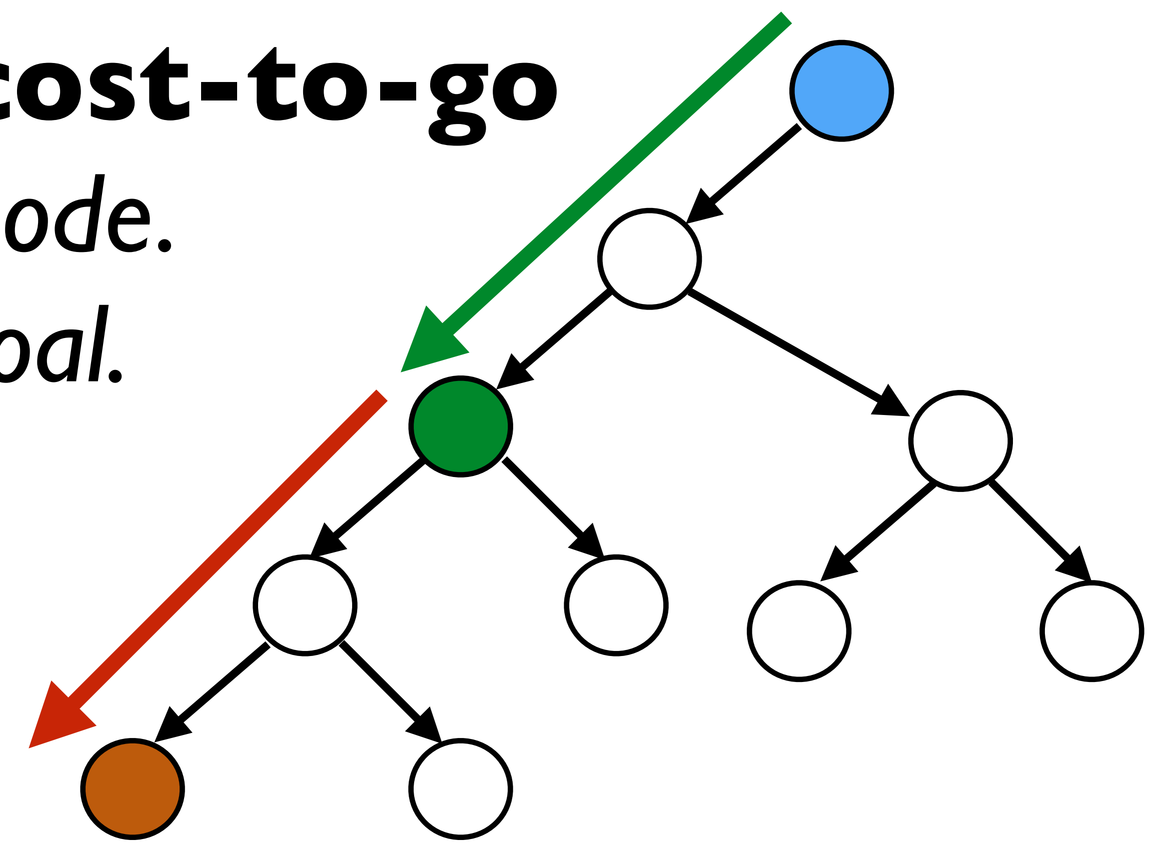
- Go to cheapest nodes first.

What remains?

Total cost = **cost-so-far** + **cost-to-go**

Cost-so-far: *cost from start to node.*

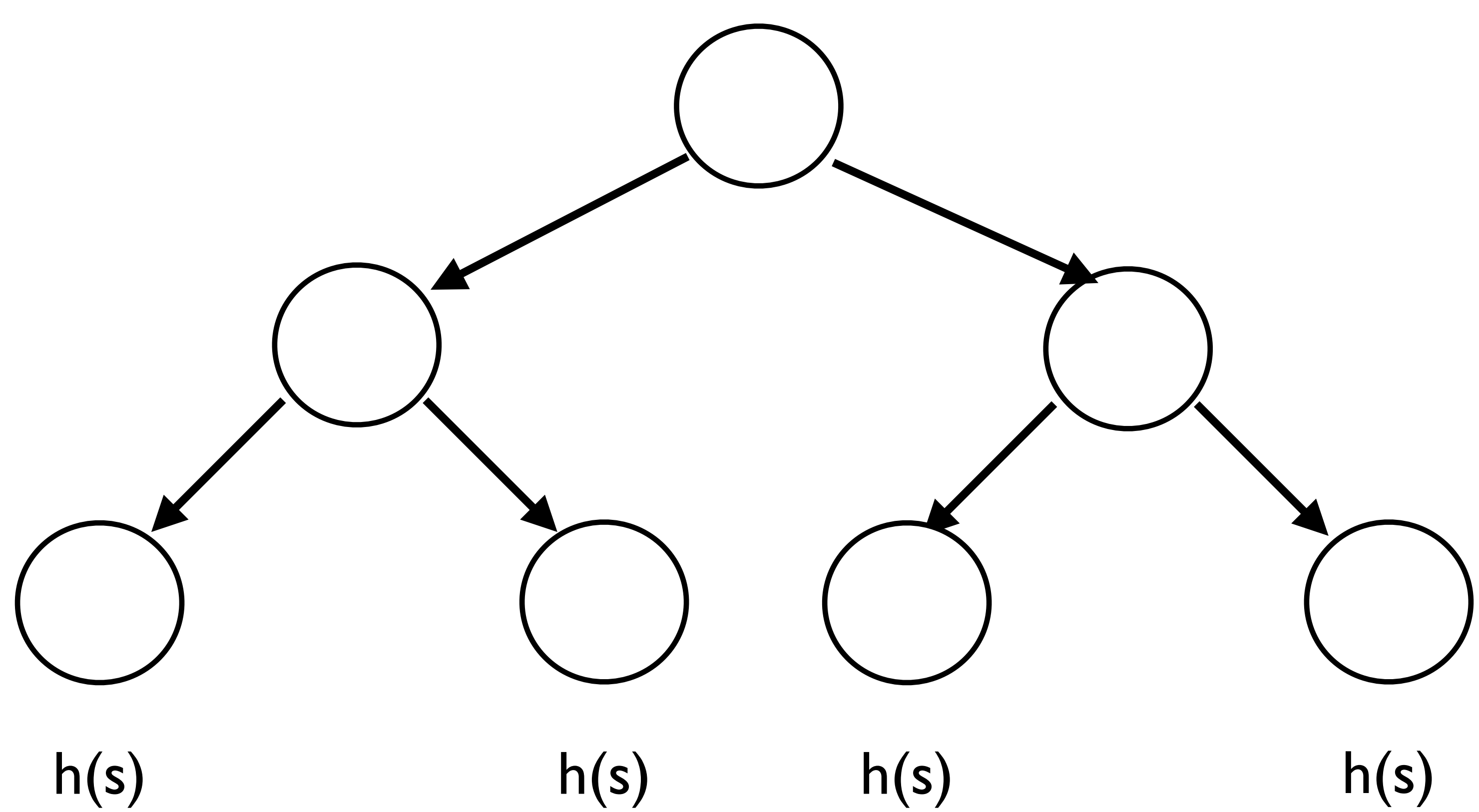
Cost-to-go: *cost from node to goal.*



Informed Search

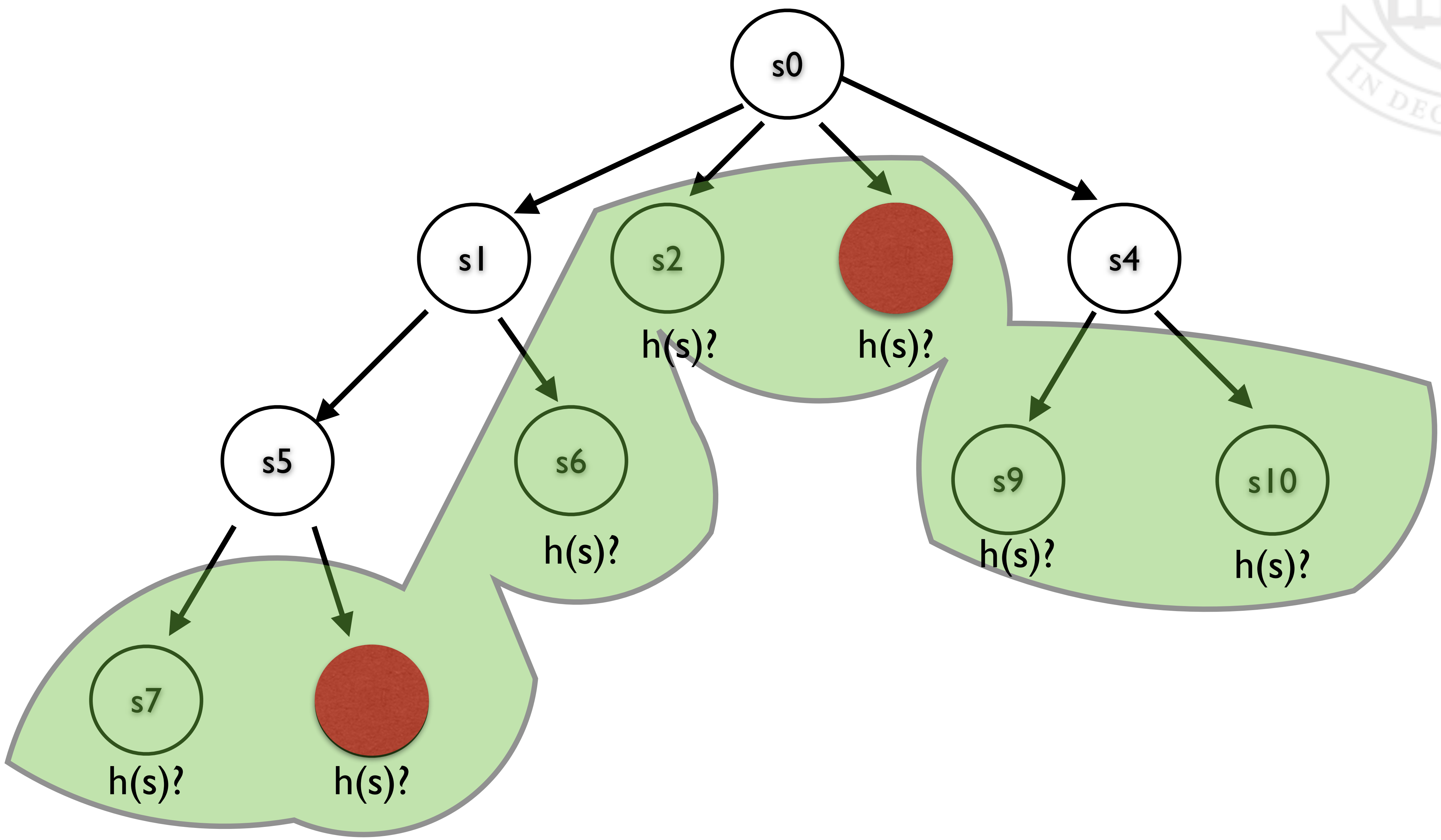
Key idea: *heuristic function*.

- *h(s)* - estimates cost-to-go
  - Cost to go *from* state *to* solution.
  - Estimates $h^*(s)$ - true cost-to-go.
  - *h(s)* = 0 if *s* is a goal.
- **Problem specific (hence *informed*)**



Greed

What if we expand the node with lowest *h(s)*?



Informed Search: A*

A* algorithm:

- *g(s)* - cost *so far* (start to *s*).
- Expand *s* that minimizes ***g(s)* + *h(s)*** both
- Manage frontier as priority queue.

- Admissible heuristic: *never overestimates* cost.

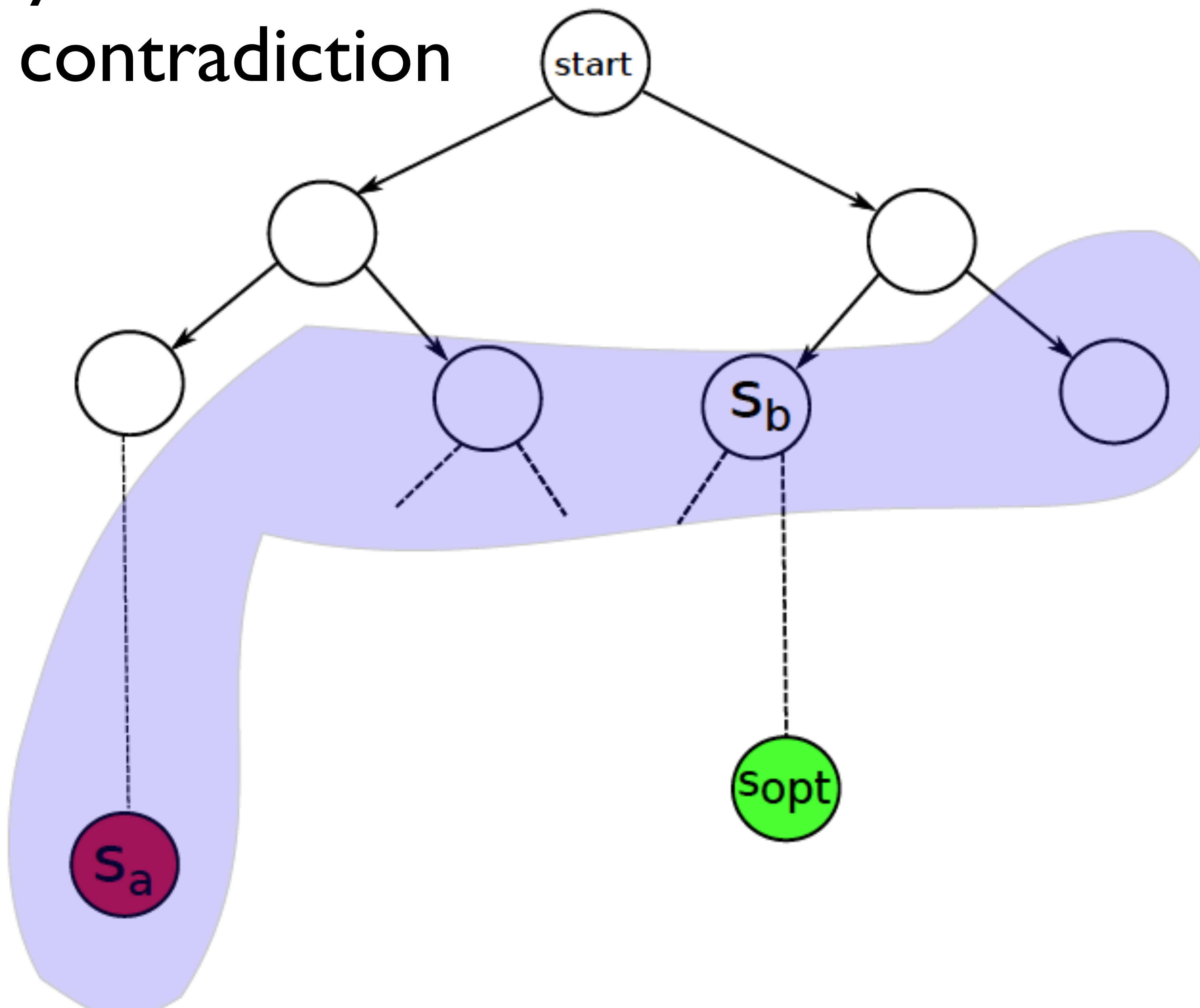
$$h(s) \leq h^*(s)$$

- *h(s) = 0* if *s* is a goal state, so *g(s)* + *h(s)* = *c(s)*

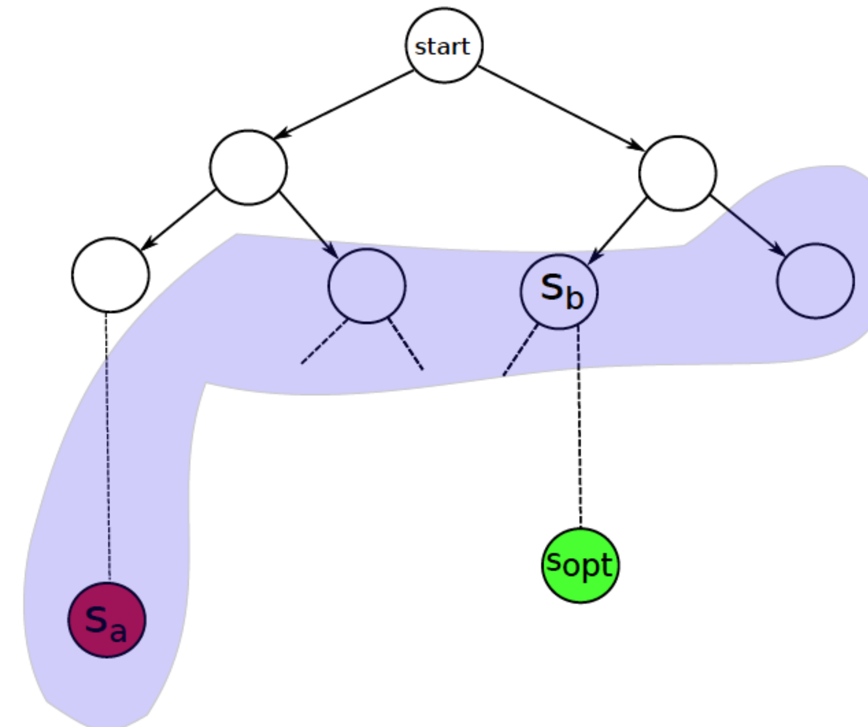
- If h is admissible, A* finds optimal solution.
- If h(s) is exact, runs in *O(bd)* time.

Admissible Heuristics

Optimality:
Proof by contradiction



Proof



Assume:

$$g(s_a) > g(s_{opt})$$

contradiction

But if s_a was opened before s_b then:

$$g(s_a) + \cancel{h(s_a)} \leq g(s_b) + h(s_b)$$

But if h is admissible then:

$$g(s_b) + h(s_b) \leq g(s_b) + h^*(s_b) = g(s_{opt})$$

… but then:

$$g(s_a) \leq g(s_b) + h(s_b) \leq g(s_{opt})$$

Example Heuristic

More on Heuristics

Ideal heuristics:

- Fast to compute.
- Close to real costs.

Some programs *automatically generate* heuristics.

ita

Software™