CSPB3202 Artificial Intelligence

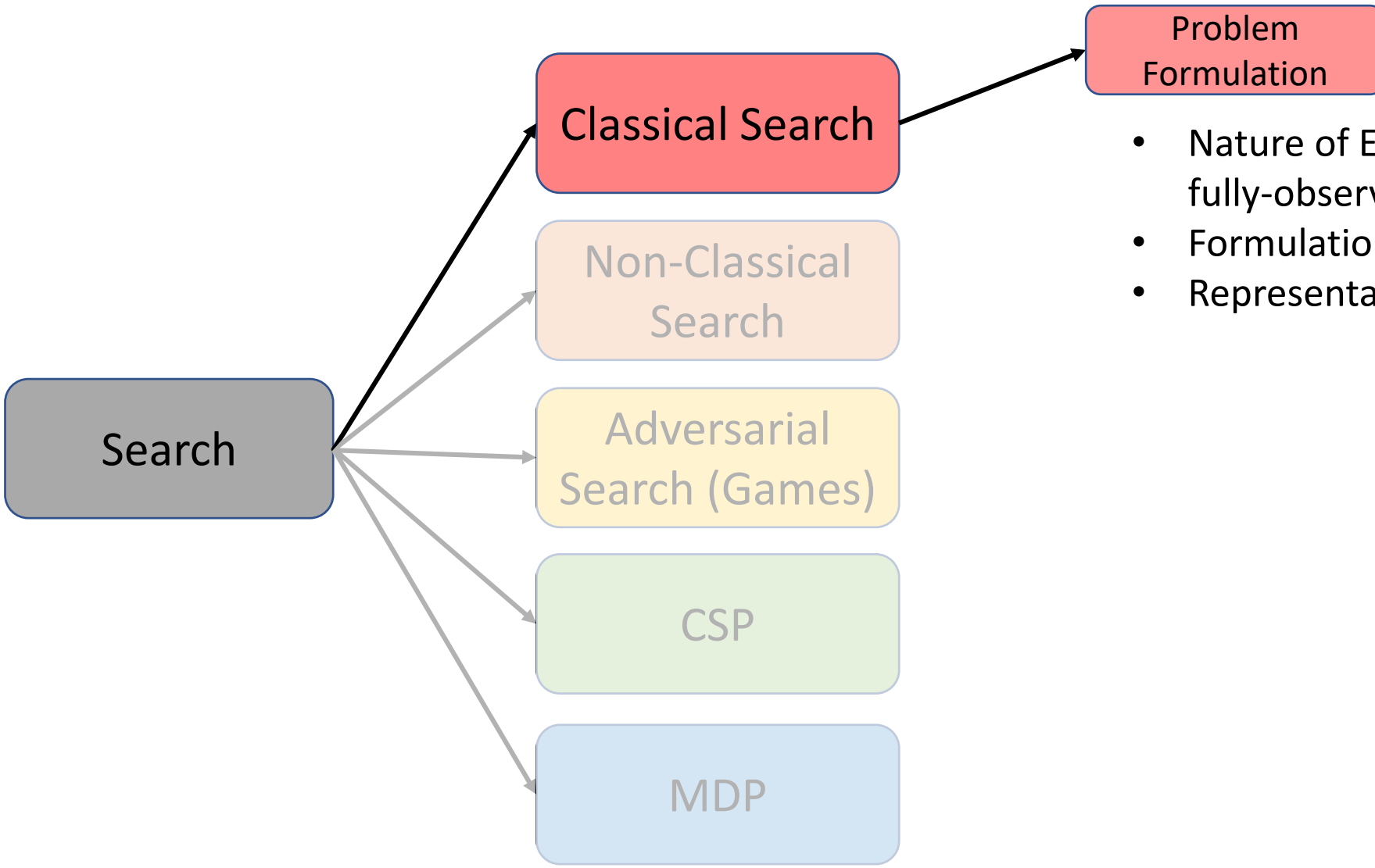
Search

University of Colorado **Boulder**

Note: Some slides were adapted from the materials from Prof. Pieter Abbeel and Prof. Dan Klein at UC Berkeley, and Russell & Norvig's AI book.

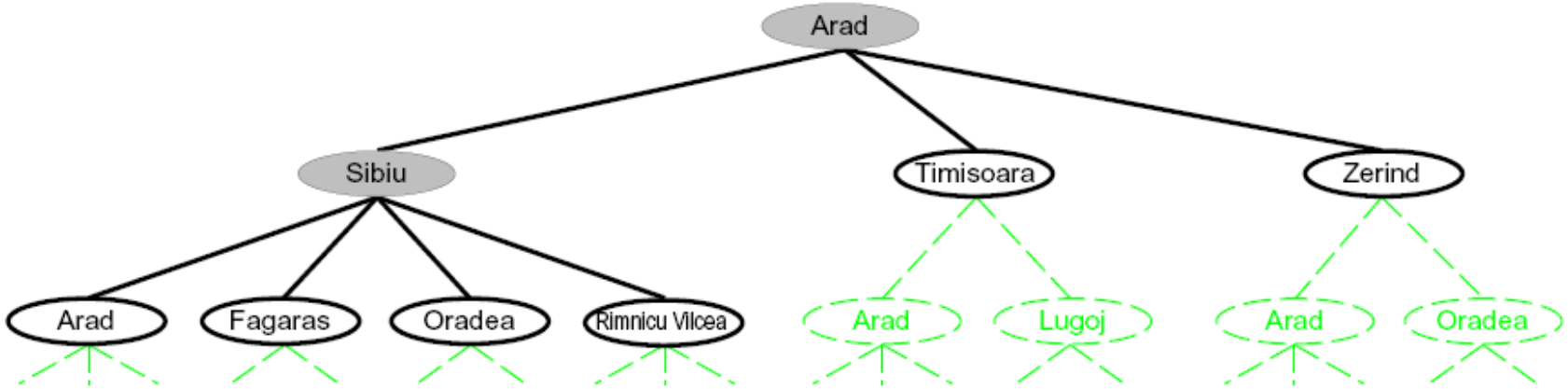
Search Strategy

Where we are



- Nature of Environment: deterministic, fully-observable, discrete, known
- Formulation: state-space
- Representation: Tree/Graph

Review: Search Algorithm




```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

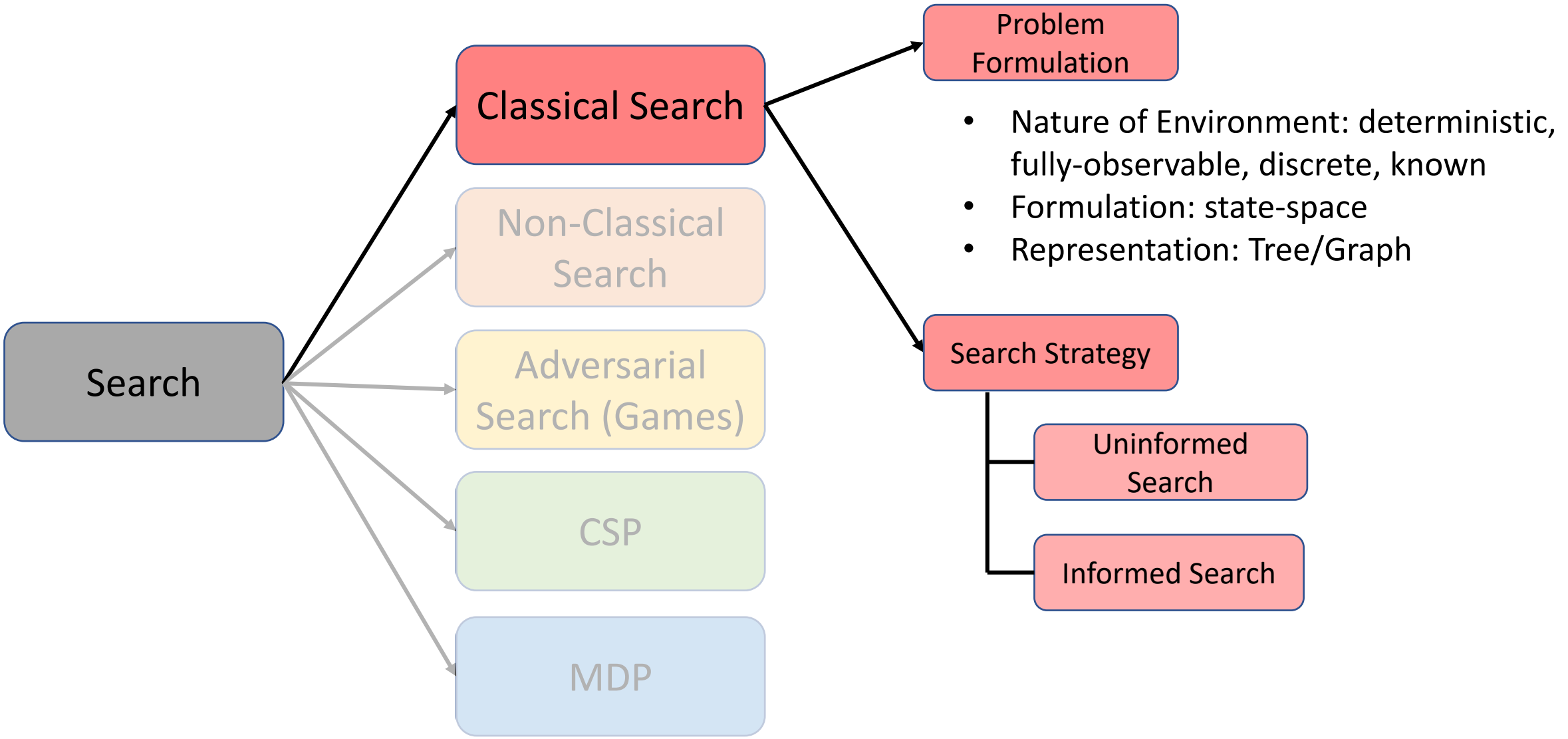
Review: Tree Search Strategy?

- Can we get to the goal with the fewest effort (without having to expand all the nodes)?

- Which Fringe Node to Explore First?

  -> Search Strategy

Where we are



- Nature of Environment: deterministic, fully-observable, discrete, known
- Formulation: state-space
- Representation: Tree/Graph

Uninformed Search Overview

Properties

- Blind search- No information other than problem definition
- Is Goal == True or False
- Does not know if one state is more promising than another

Kinds

- Depth-first search (DFS)
- Breadth-first search (BFS)
- Uniform-cost search (UCS)

Tools for Algorithm Analysis

Algorithm Properties

- Completeness- guaranteed to find a solution if one exists?
- Optimality- guaranteed to find an optimal solution?
- Time Complexity- how long the computation take to find a solution?
- Space Complexity- how much memory to do the computation?

Tools for Algorithm Analysis

Big-O notation

- BigO notation estimates the complexity given input size N
- It is machine-independent
- They can be used for both time and space complexity
- There are worst-case, best-case, and average-case, but we typically use worst-case
- When analyze, we ignore the constant in front of O, and we only care about the most dominant term.
- Dominance order

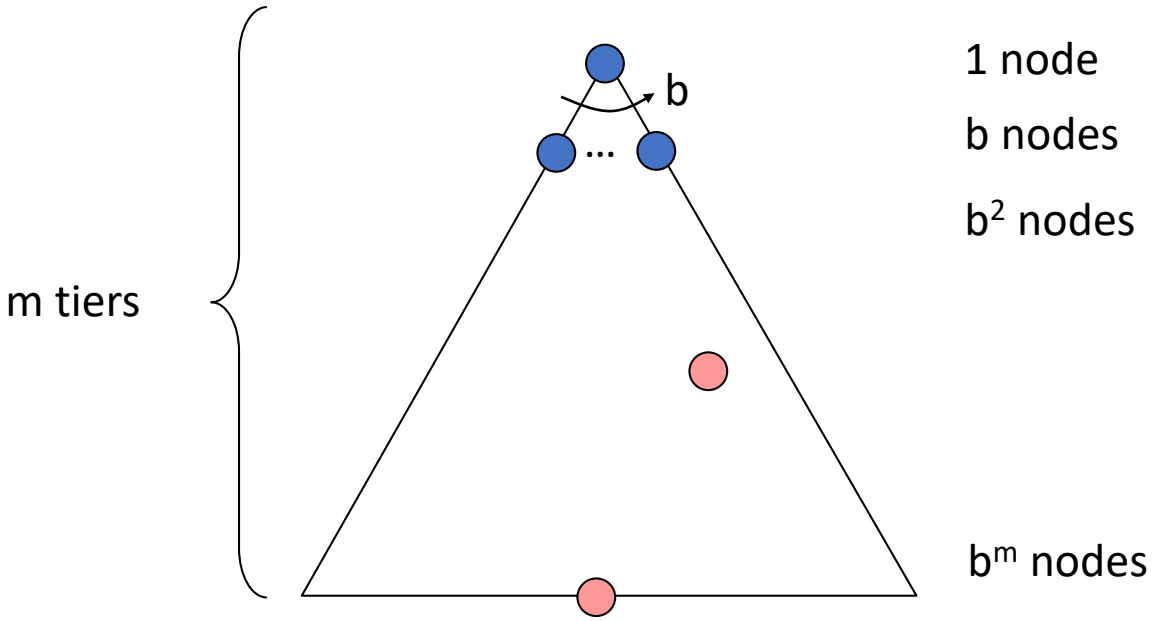
$$O(1) < O(\log(n)) < O(\sqrt{n}) < O(n) < O(n\log(n)) < O(n^2) < O(2^n) < O(n!)$$

Search Algorithm Properties

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

- Quantifying search tree parameters
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

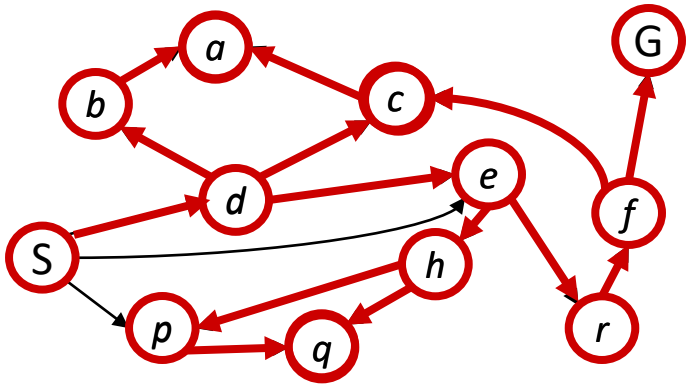
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$

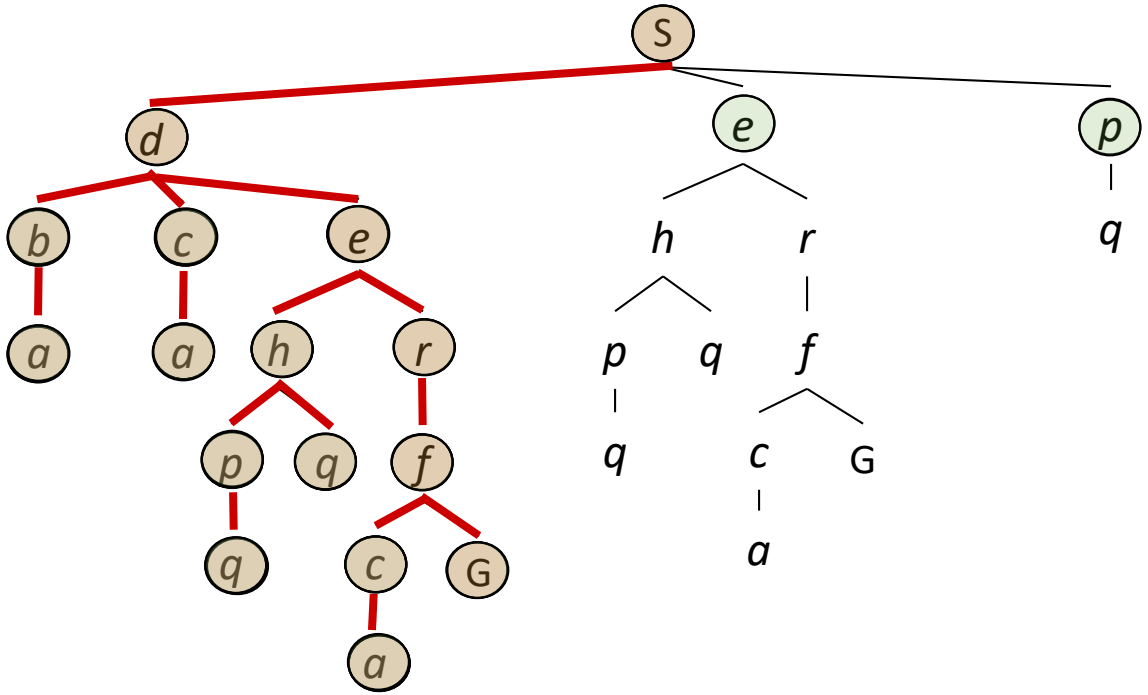


Depth-First Search

Strategy: expand a deepest node first

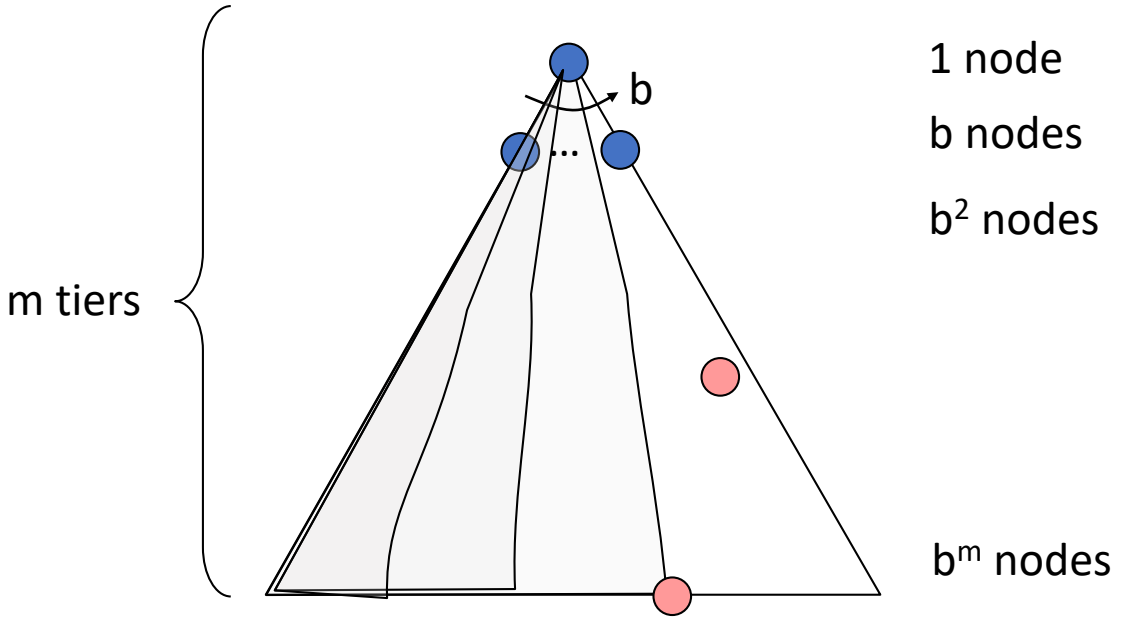
Implementation: Fringe is a LIFO stack





Depth-First Search (DFS) Properties

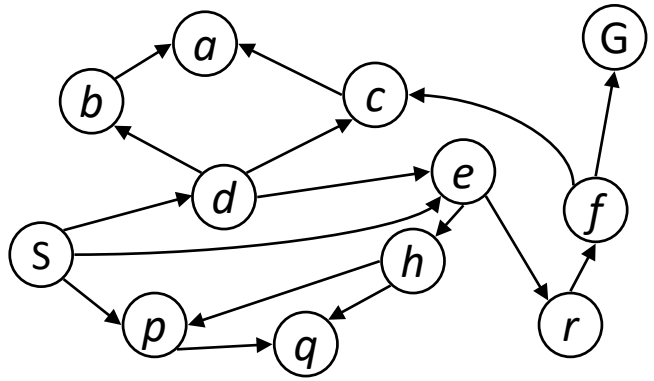
- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so O(bm)
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost



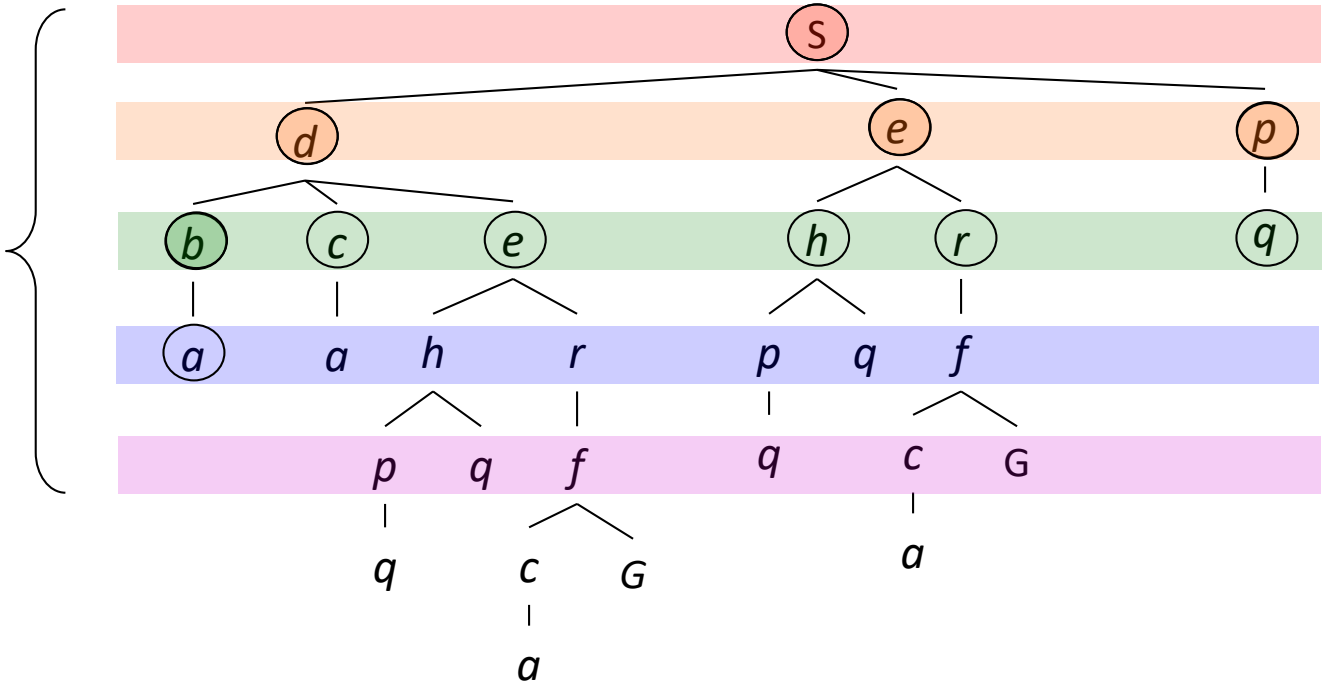
Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

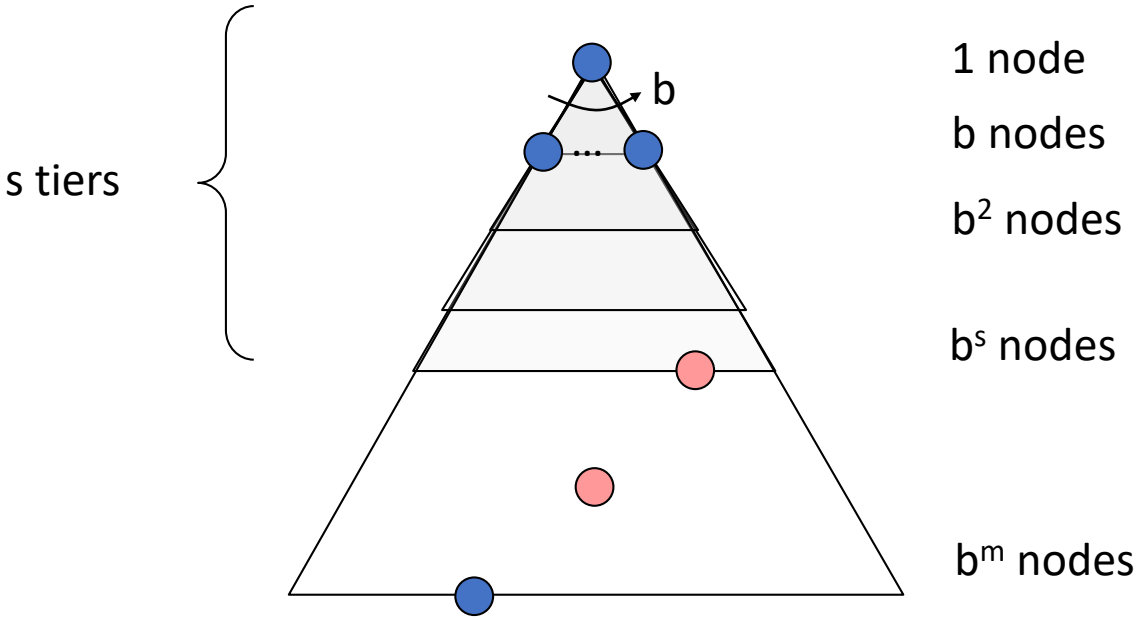


Search Tiers



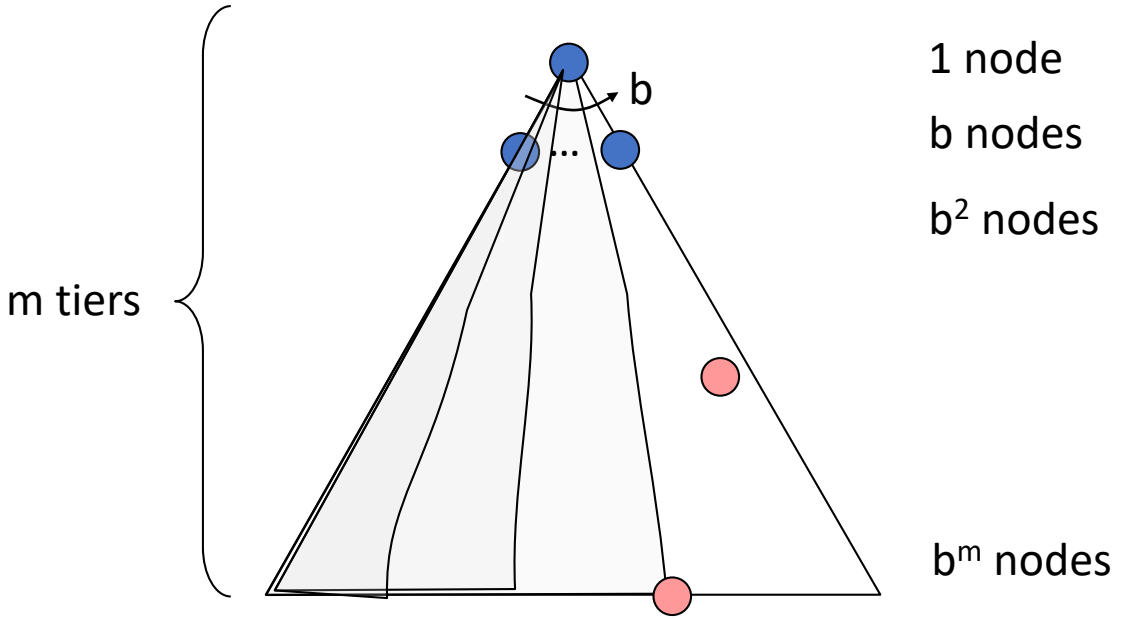
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



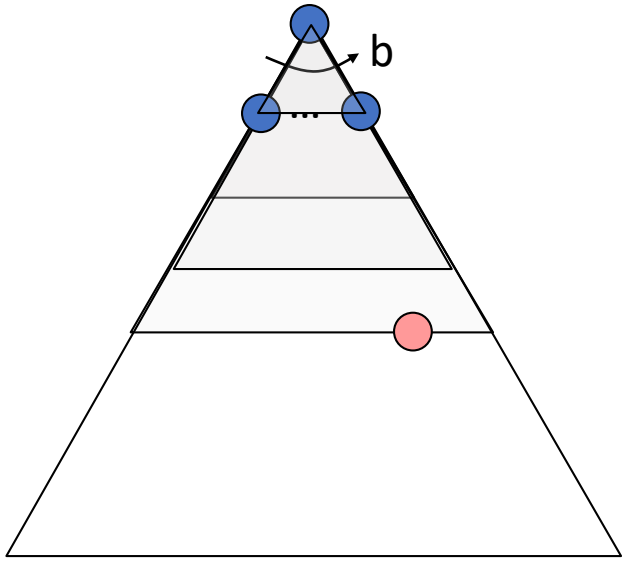
DFS vs. BFS

Depth-Limited Search

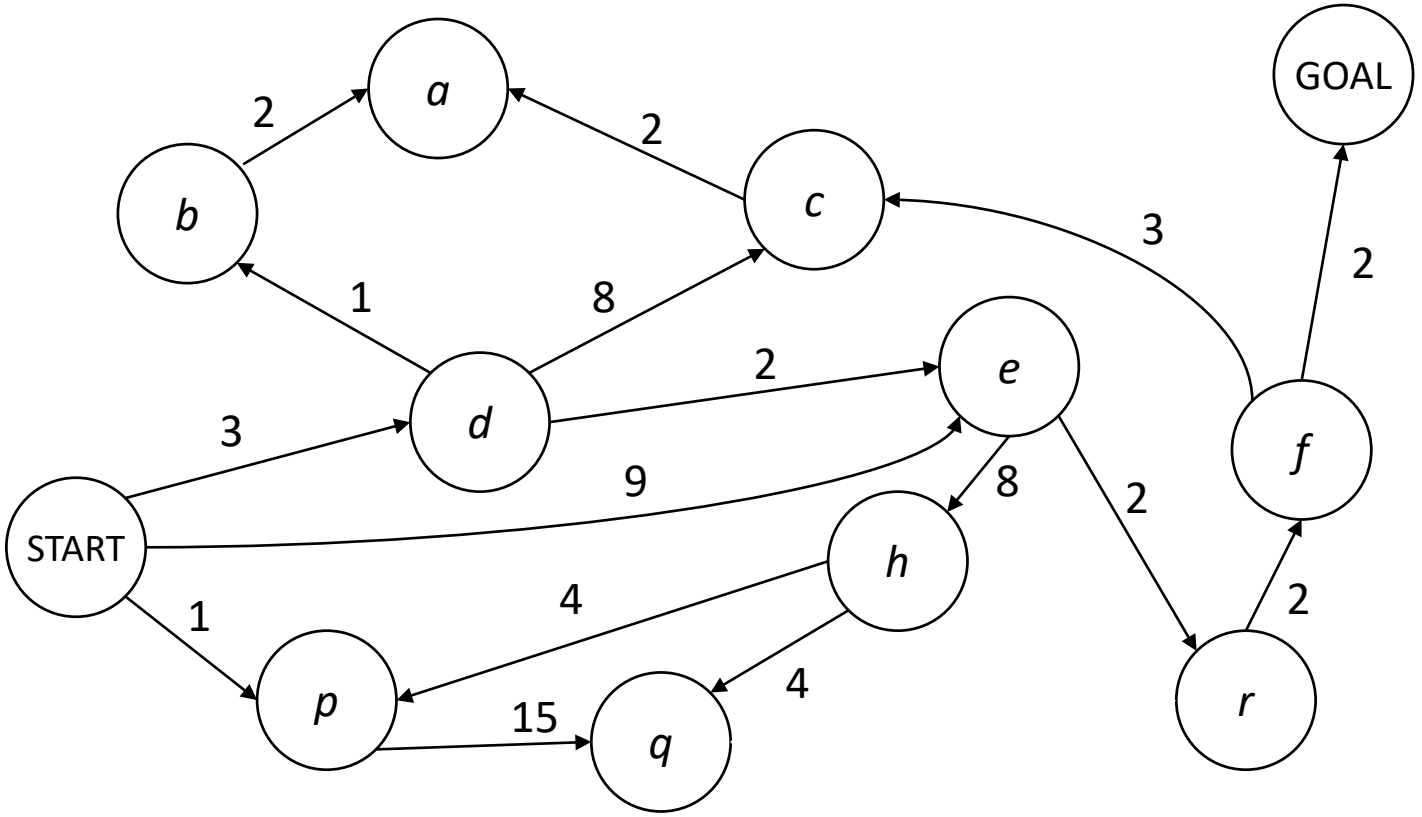


Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..
- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!



Searching the least cost

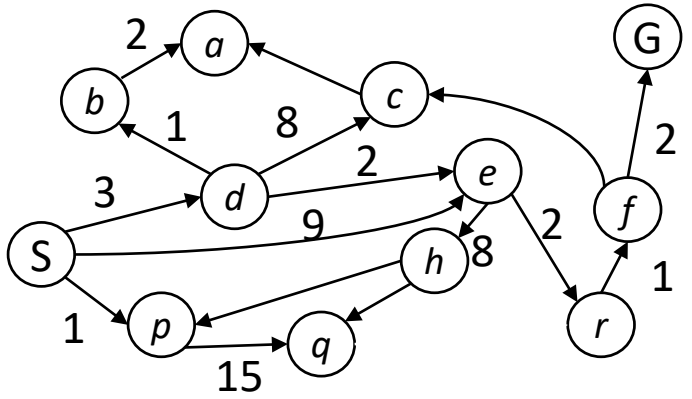


BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

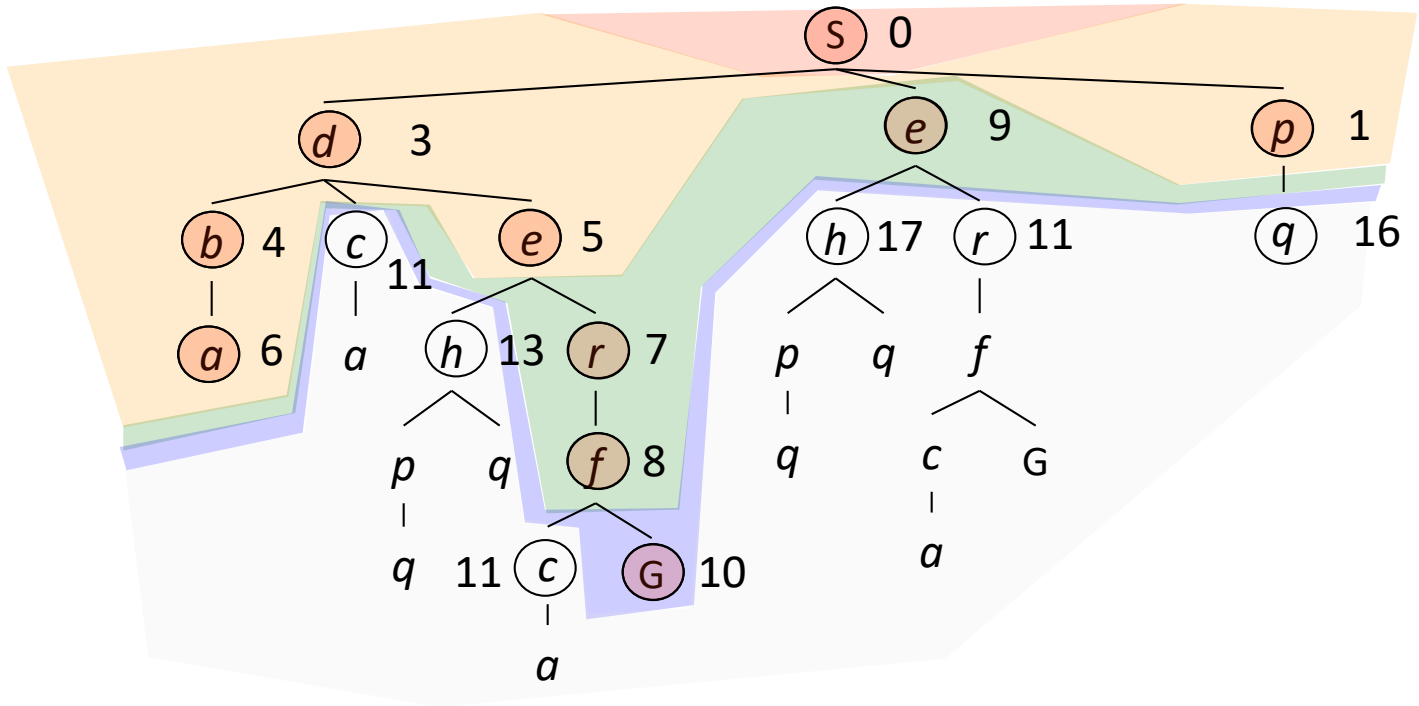
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

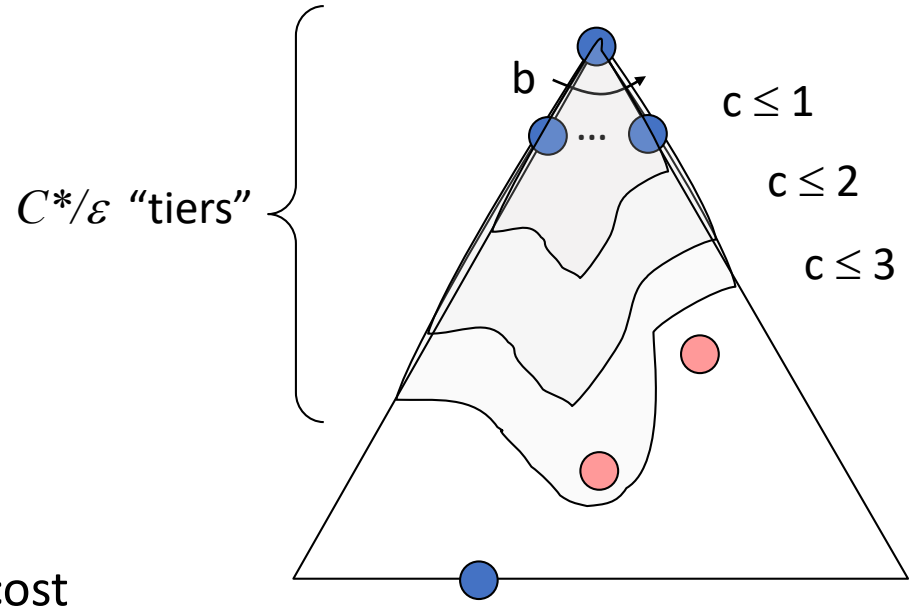


Cost contours



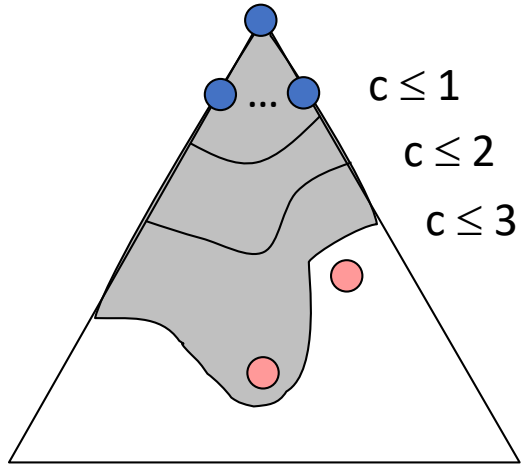
Uniform Cost Search (UCS) Properties

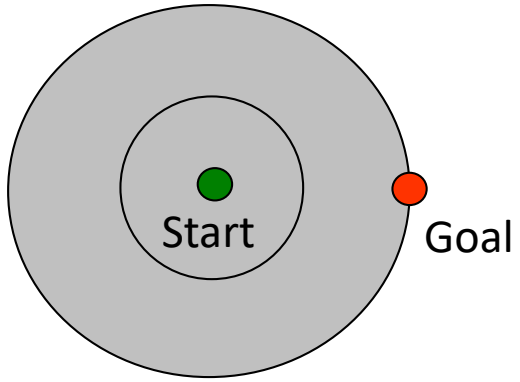
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)



Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location
- We’ll fix that soon!





Uninformed Search Summary

	DFS	BFS	Iterative Deepening	UCS
Completeness	Yes only if no cycle	Yes	Yes	Yes (assuming no cost loops)
Optimality	No	Yes if all costs are equal	Yes if all costs are equal	Yes
Time Complexity	$O(b^m)$	$O(b^s)$	$O(b^s)$	$O(b^{C*/\varepsilon})$
Space Complexity	$O(bm)$	$O(b^s)$	$O(bs)$	$O(b^{C*/\varepsilon})$
Advantage	Efficient in space	Relatively efficient in time when $s < m$	Combines advantages of DFS and BFS	Complete and optimal
Disadvantage / Limitation	Not optimal, can go into a rabbit hole	Not memory efficient	Does not consider cost	Cheap cost so far doesn't mean it's a right direction