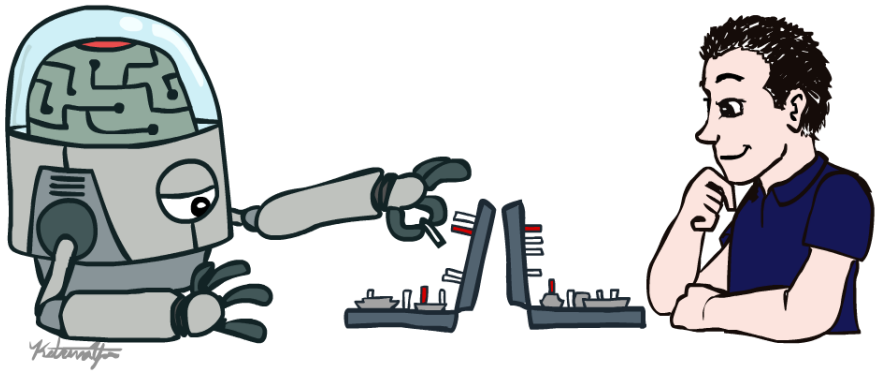
CSE 3521:
Introduction to Artificial Intelligence



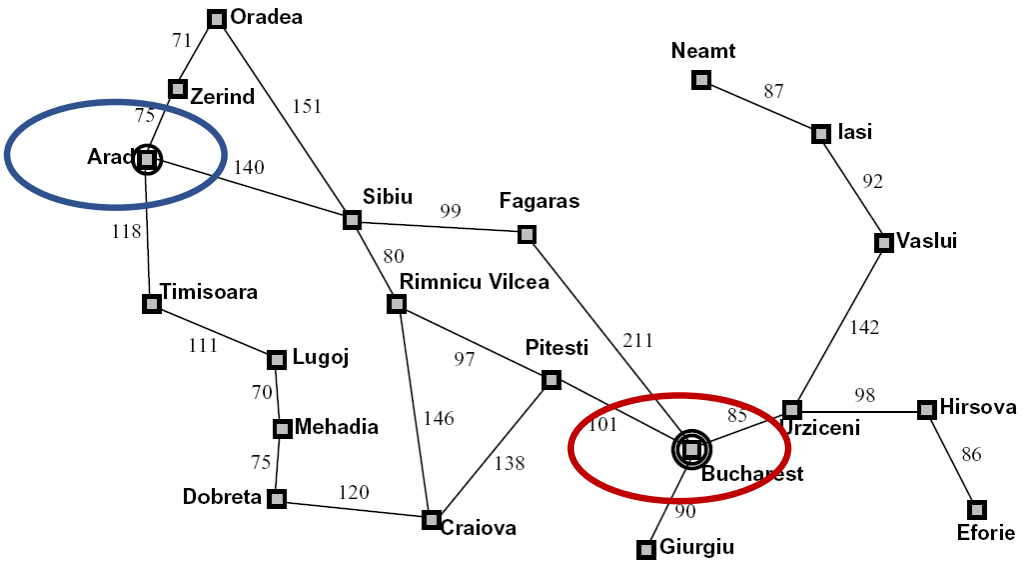
[These slides are partially adapted from the UC Berkeley. CS188 Intro to AI at UC Berkeley]

THE OHIO STATE UNIVERSITY

Search Problems

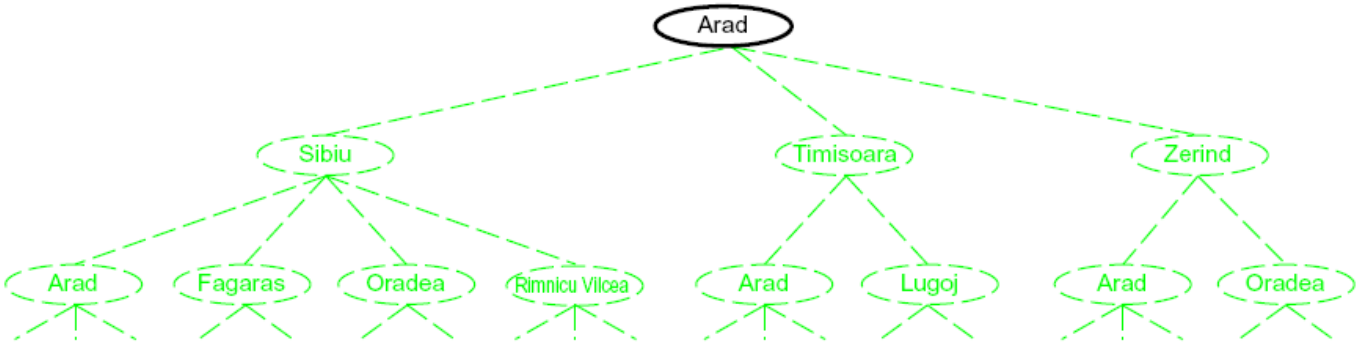
- A search problem consists of:
  - A state space
  - A successor function (with actions, costs)
  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Example: Traveling in Romania



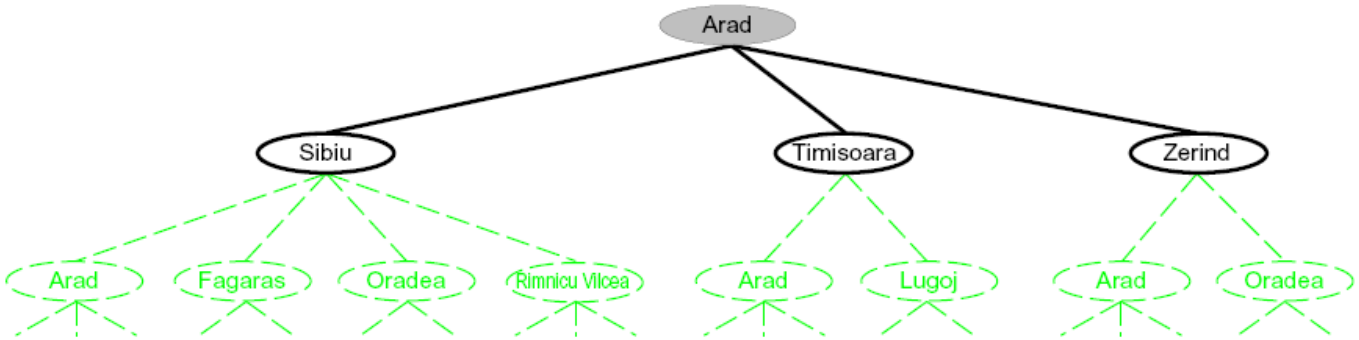
- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?
  - If there exists one, it is a sequence of cities from Arad to Bucharest

Searching with a Search Tree



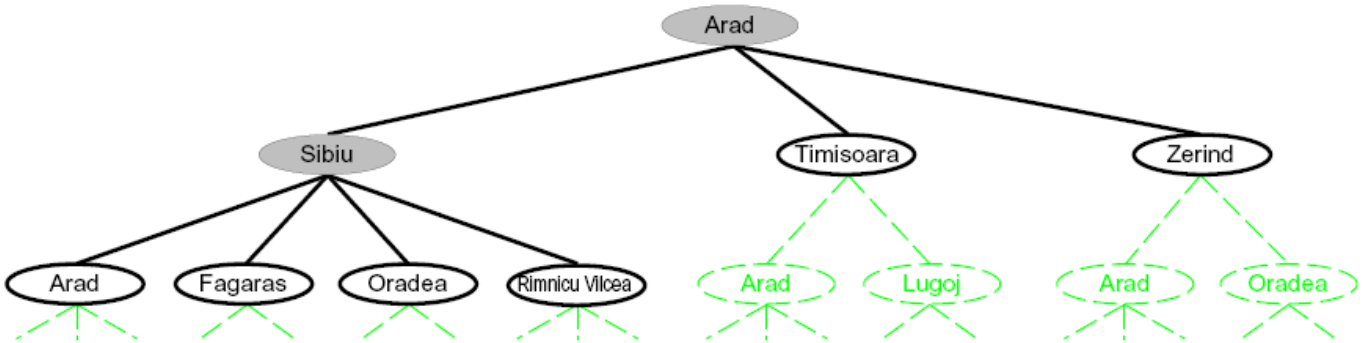
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

Searching with a Search Tree



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

Searching with a Search Tree



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

```
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
```

- Important components: (1) Fringe (2) Expansion (3) Exploration strategy
- Main question: which fringe nodes to explore?

Uninformed vs. Informed Search

- Uninformed search
  - Given no information about problem (other than its definition)
  - Find solutions to problems by systematically generating new states and testing for goal

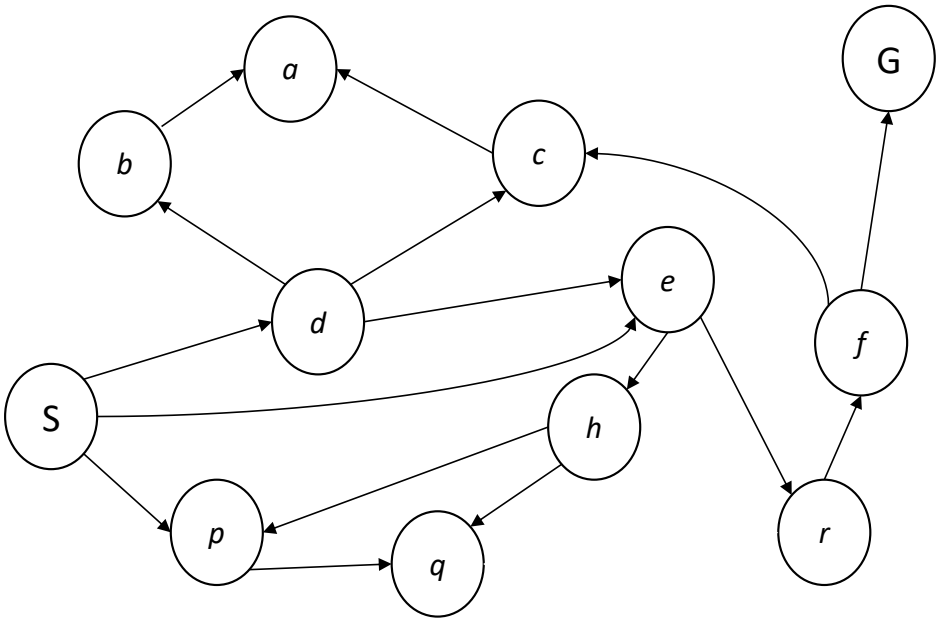
- Informed search
  - Given some ideas of where to look for solutions
  - Use problem-specific knowledge

Uninformed vs. Informed Search

- Uninformed search
  - Given no information about problem (other than its definition)
  - Find solutions to problems by systematically generating new states and testing for goal

- Informed search
  - Given some ideas of where to look for solutions
  - Use problem-specific knowledge

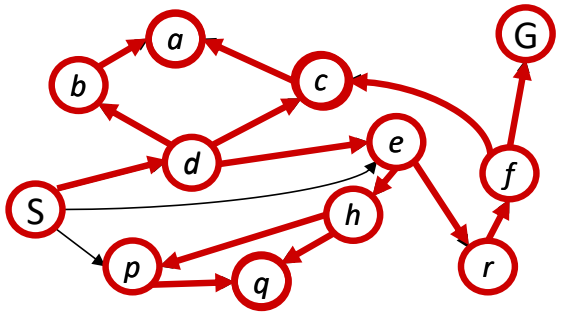
Example: Tree Search

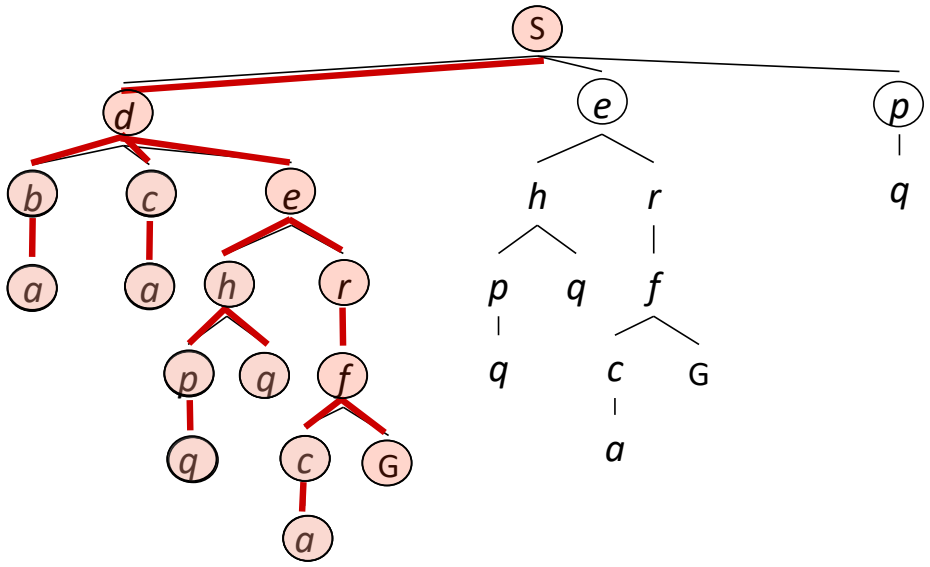


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack



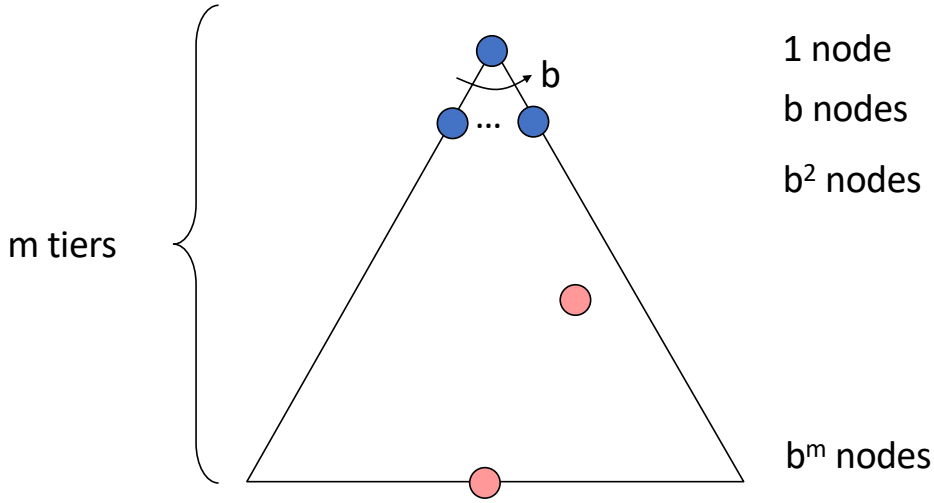


Search Algorithm Properties

- **Complete:** Guaranteed to find a solution if one exists?
- **Optimal:** Guaranteed to find the least cost path?

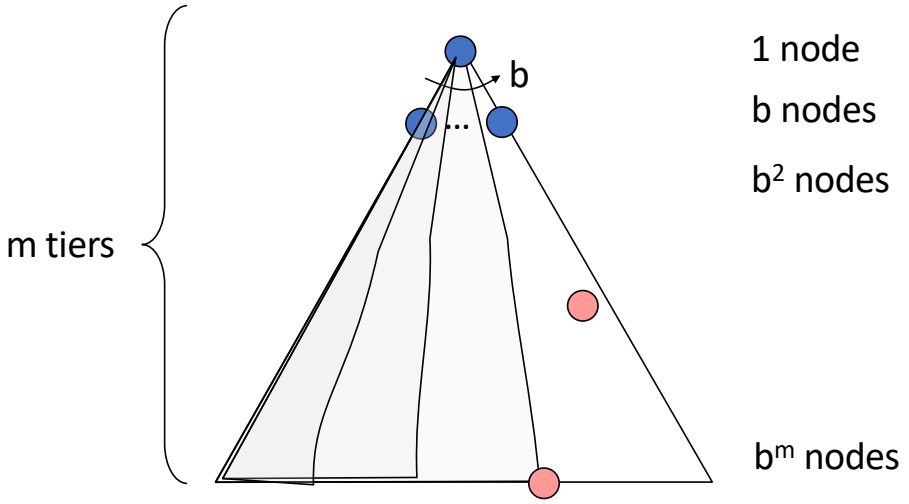
- Search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

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



Depth-First Search (DFS) Properties

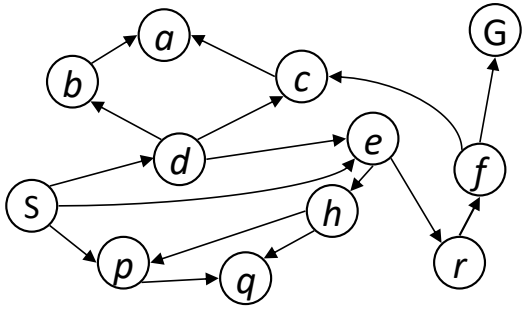
- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on the current path to root, so O(bm)
- Is it complete?
  - m could be infinite, so only if we prevent cycles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

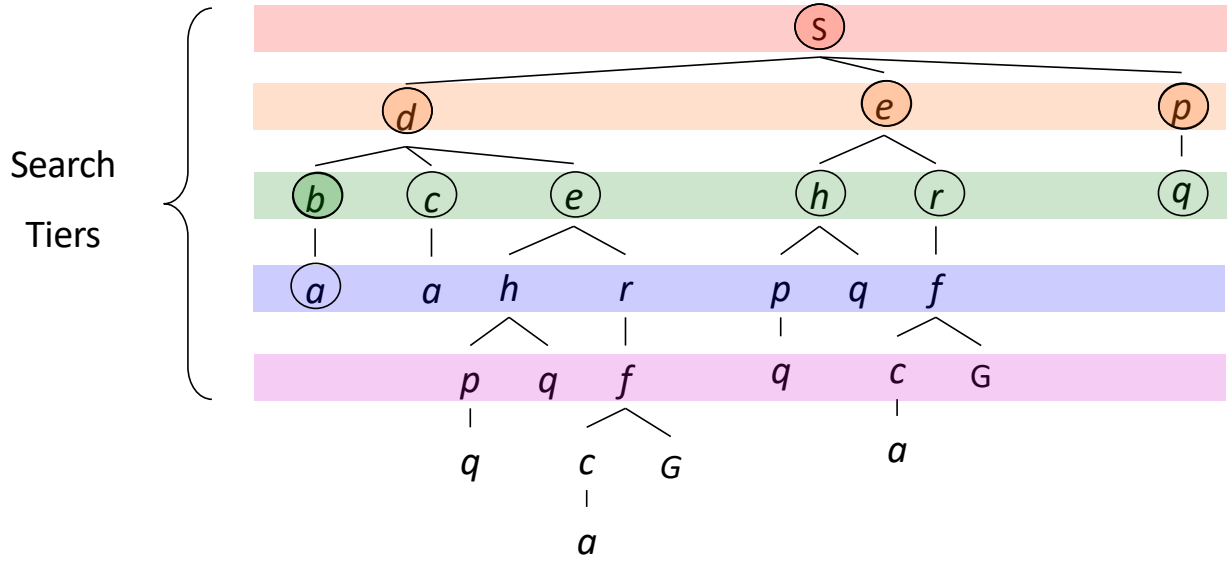


Breadth-First Search

Strategy: expand a shallowest node first

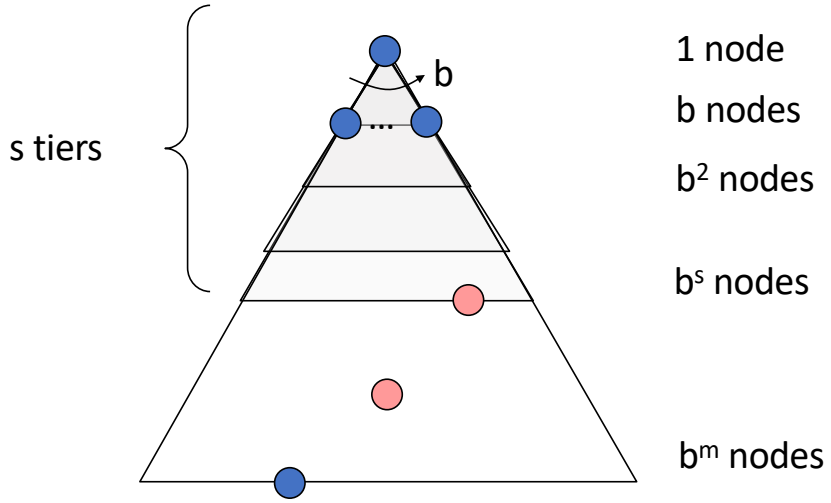
Implementation: Fringe is a FIFO queue





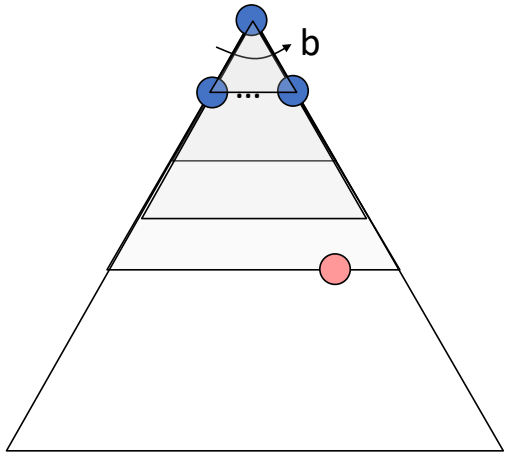
Breadth-First Search (BFS) Properties

- 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)

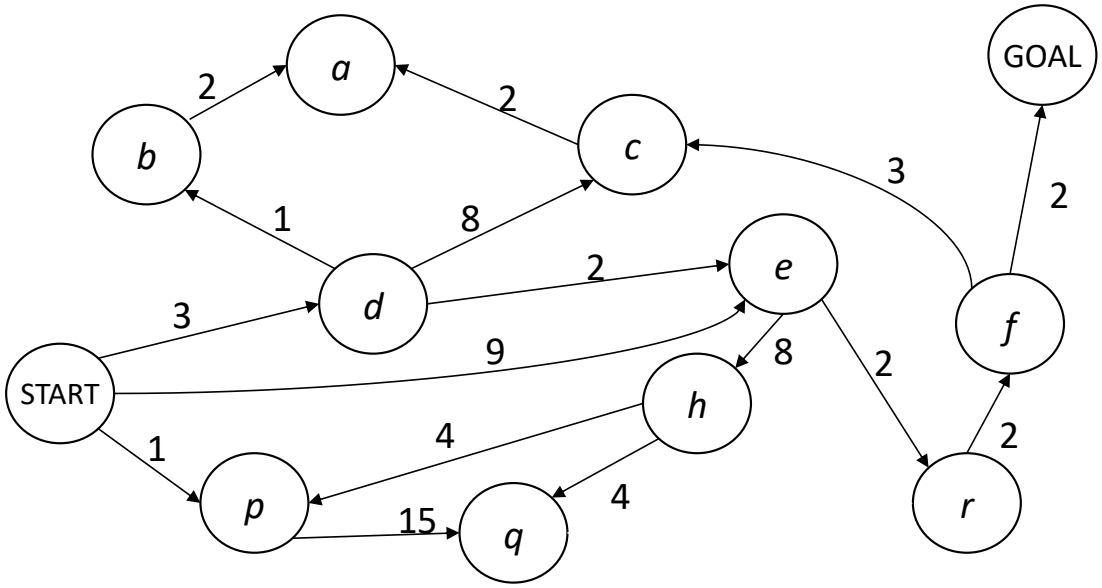


Iterative Deepening Search

- 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!
- A Preferred method with large search space and depth of solution not known



Cost-Sensitive Search

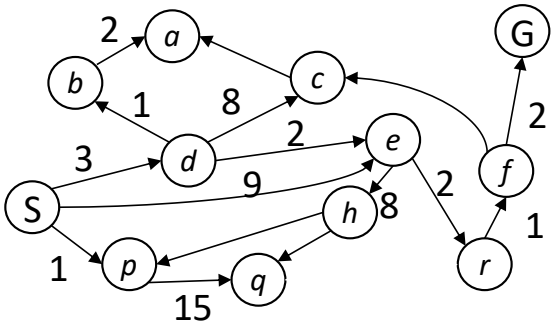


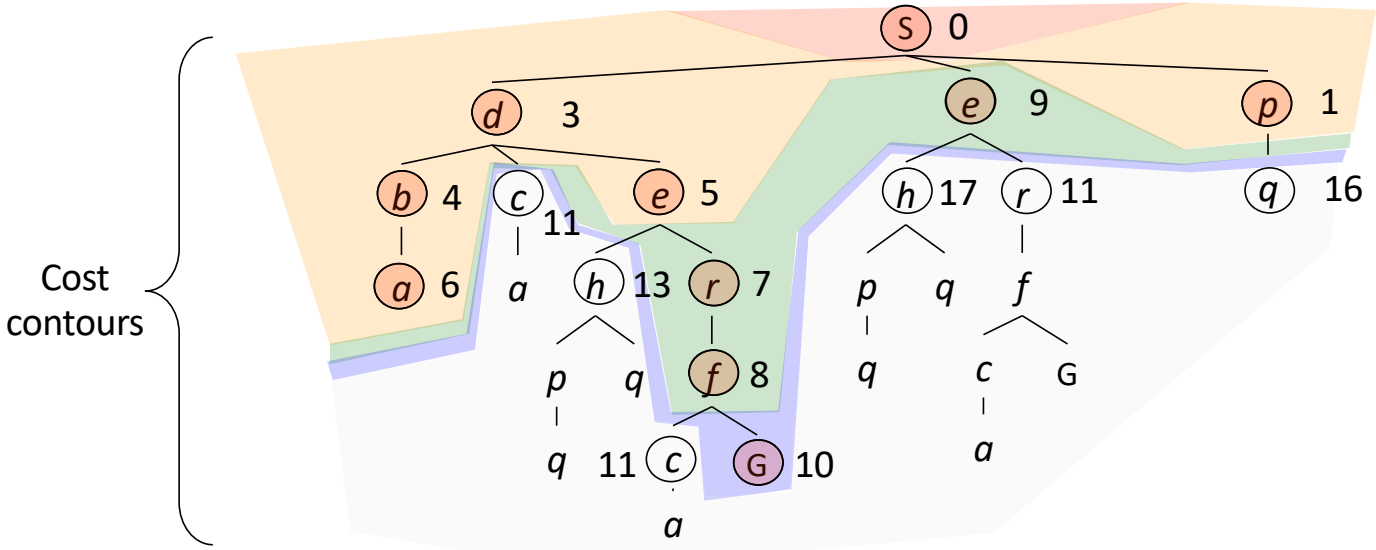
BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

Uniform Cost Search (USC)

Strategy: expand a cheapest node first:

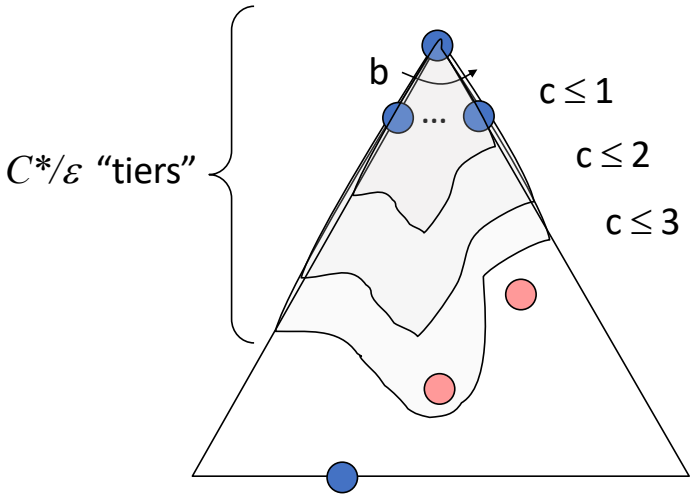
Fringe is a priority queue (priority: cumulative cost)





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)
  - Additional cost of priority queue
- 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!



The One Queue

- All the search algorithms are the same except for fringe/exploration strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object