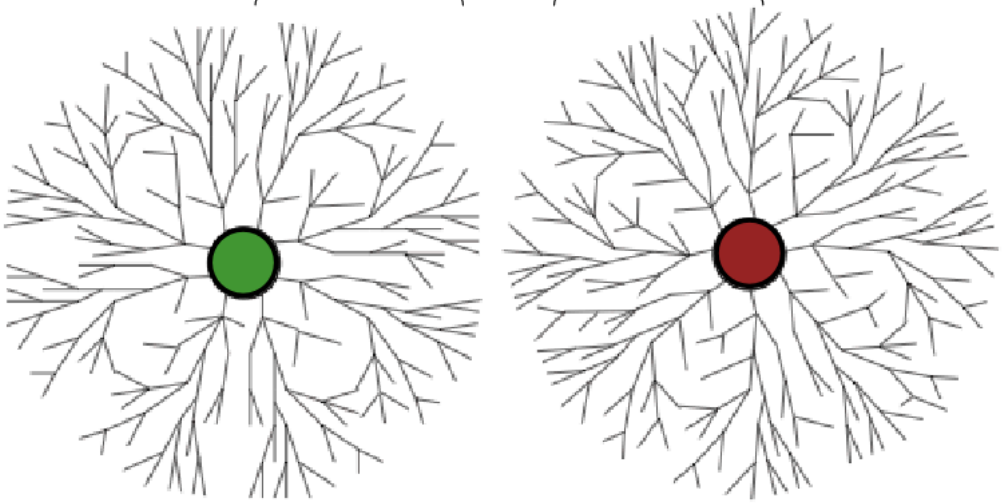
```
function DepthLimitedSearch(problem, l) -> return a solution, or failure or
cutoff
    frontier <- LIFO queue(stack)
    frontier <- {initial state}
    result <- failure
    repeat
        if fringe = null then break
        node <- POP(frontier)
        if IsGoal(node) then return correspoding solution
        if Depth(node) >= l then result <- cutoff
        else
            for a in Action(node):
            a' = Result(node, a)
            INSERT(frontier, a')
    return result
```

Iterative Deepening Search (IDS)

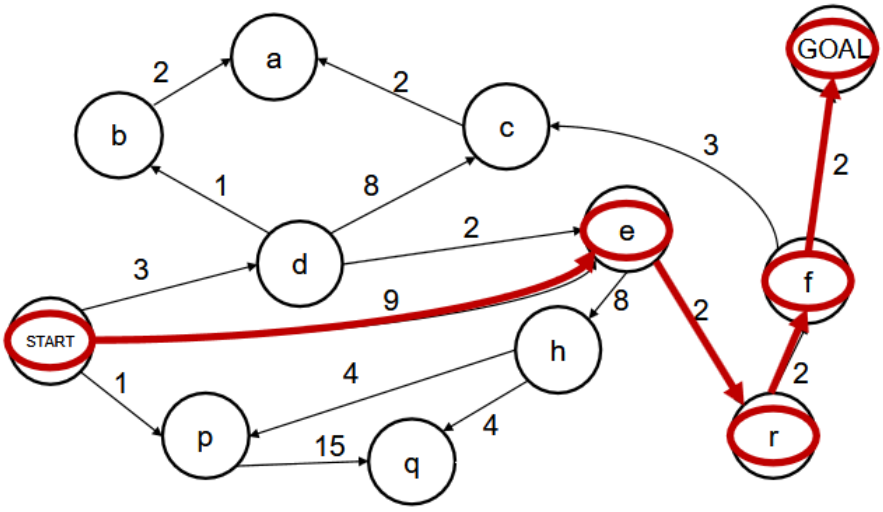
- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
    - Run a DLS with depth limit 1. If no solution...
    - Run a DLS with depth limit 2. If no solution...
    - Run a DLS with depth limit 3. .....
  - Isn't that wastefully redundant?
    - Generally, most work happens in the bottom level searched, so not so bad
  - Compare IDS, BFS ($b = 10, d = 5$):
    - N(BFS) = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
    - N(IDS) = 5 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,455

Bi-directional Search

Desirable when $O(b^{d/2})$ + $O(b^{d/2})$ < $O(b^d)$



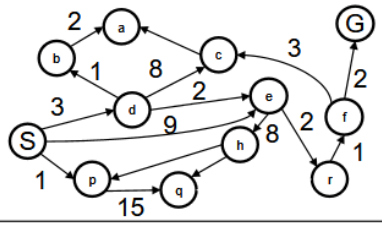
Cost-Sensitive Search



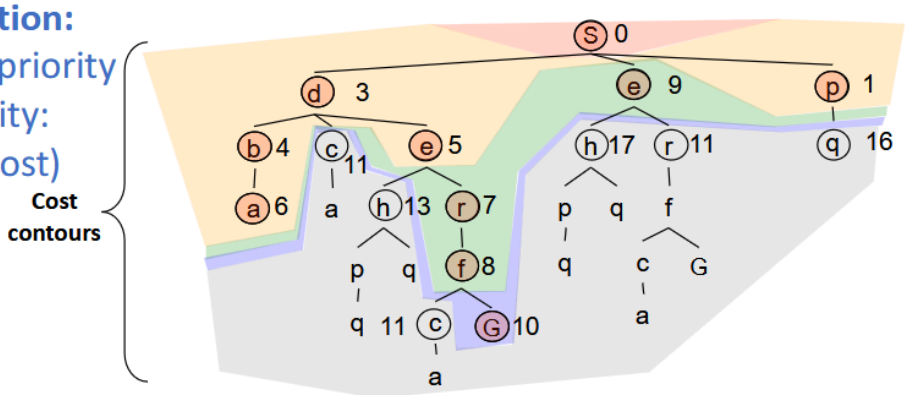
BFS finds the shortest path in terms of number of actions.
It **does not** find the least-cost path.

Uniform Cost Search

Strategy: expand cheapest node first:

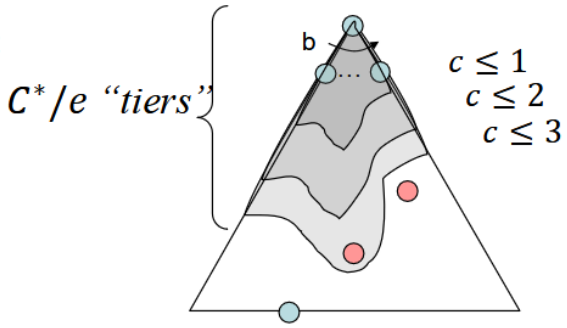


Implementation: Frontier is a priority queue (priority: cumulative cost)



Uniform Cost Search (UCS) Properties

- **Time complexity**
  - Processes all nodes with cost less than cheapest solution
  - If that solution costs C^* and actions cost at least e , then the "effective depth" is roughly C^*/e
  - Takes time $O(b^{(1+\lfloor C^*/e \rfloor)})$
- **Space complexity**
  - Has roughly the successors of the last tier, so
  - $O(b^{(1+\lfloor C^*/e \rfloor)})$
- **Is it complete?**
  - Yes, assuming best solution has finite cost and min action cost is positive
- **Is it optimal?**
  - Yes



Summary

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

- b <- branching factor
- m <- maximum depth of the search tree

- d <- depth of the shallowest solution or is m when there is no solution
- l <- is the depth limit
- $()^1$ complete if b is finite and the state space either has a solution or is finite
- $()^2$ complete if all action costs are $\geq \epsilon \geq 0$
- $()^3$ cost optimal if action costs are all identical
- $()^4$ if both direction are breadth-first or uniform-cost

BEST-FIRST SEARCH

- Choose a node n with **minimum value** of some evaluation function $f(n)$
- Priority queue ordered by f

Example:
BFS -> expand the **shallowest** node first, $f(n) =$ Depth(n)

INFORMED SEARCH

- The search algorithm uses domain specific hints about the location of goals
- hints are formularized through a **heuristic function** $h(n)$
- $h(n) =$ estimated cost of the cheapest path from the state at node n to a goal state
- Idea: Best-First Search where $f(n)$ contains $h(n)$

HEURISTIC FUNCTION:

- A **non-negative** function that estimates **how close a state is to a goal**, designed for a particular problem
- if n is a goal state $\Rightarrow h(n) = 0$
- Ex: Manhattan distance, Euclidean distance

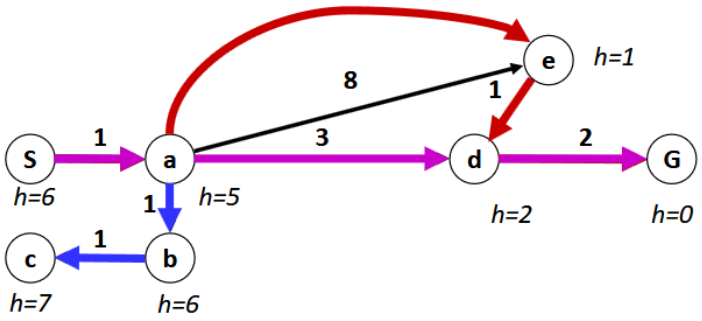
GREEDY SEARCH

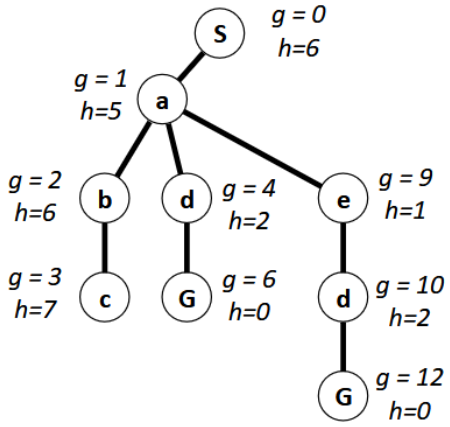
- Choose $f(n) = h(n)$
- Expand the node that seems closest
- What can wrong?

A* search

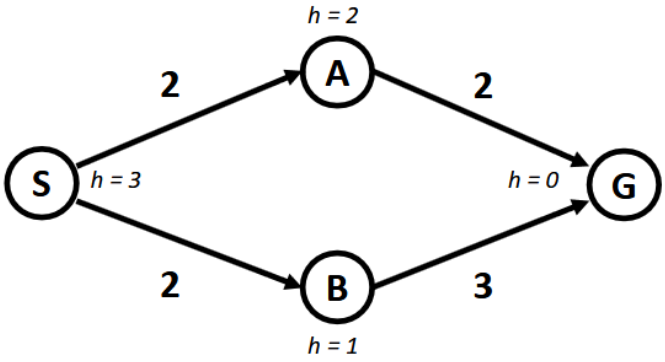
- $f(n) = g(n) + h(n)$
- $g(n)$ <- path cost from start node to current node n (Uniform cost search)
- $h(n)$ <- estimate of path cost from n to the goal (Greedy best-first search)
- Expands the node n if the estimated cost of the solution passing through it is the lowest
- Combines uniform cost and greedy search

- A* search is optimal if $h(n)$ is an underestimate of the actual cost from n to the goal
  - Uniform-cost orders by path cost, or *backward cost* $g(n)$
  - Greedy orders by goal proximity, or *forward cost* $h(n)$
  - A* Search orders by the sum: $f(n) = g(n) + h(n)$





Note, A* should not terminate when a goal state enters the frontier, instead terminate when a goal stats is expanded (exists the frontier)



Frontier	Expand
(S, 3)	(S, 3)
(A, 4) (B, 3)	(B, 3)
(A, 4) (G, 5)	(A, 4)
(G, 4) (G, 5)	(G, 4)

Return Path: S->A->G
with cost 4