Agents that Plan Ahead: Search

Russell and Norvig: Chapter 3.1-3.4, 3.5-3.6

CSE 240: Winter 2023

Lecture 2

Announcements

Announcements

- Assignment 1 is up
- Prof. Marinescu lecturing on Thursday.
  - I am planning to hold office hours on Thursday (might be subject to change).
- Use slack (#questions)

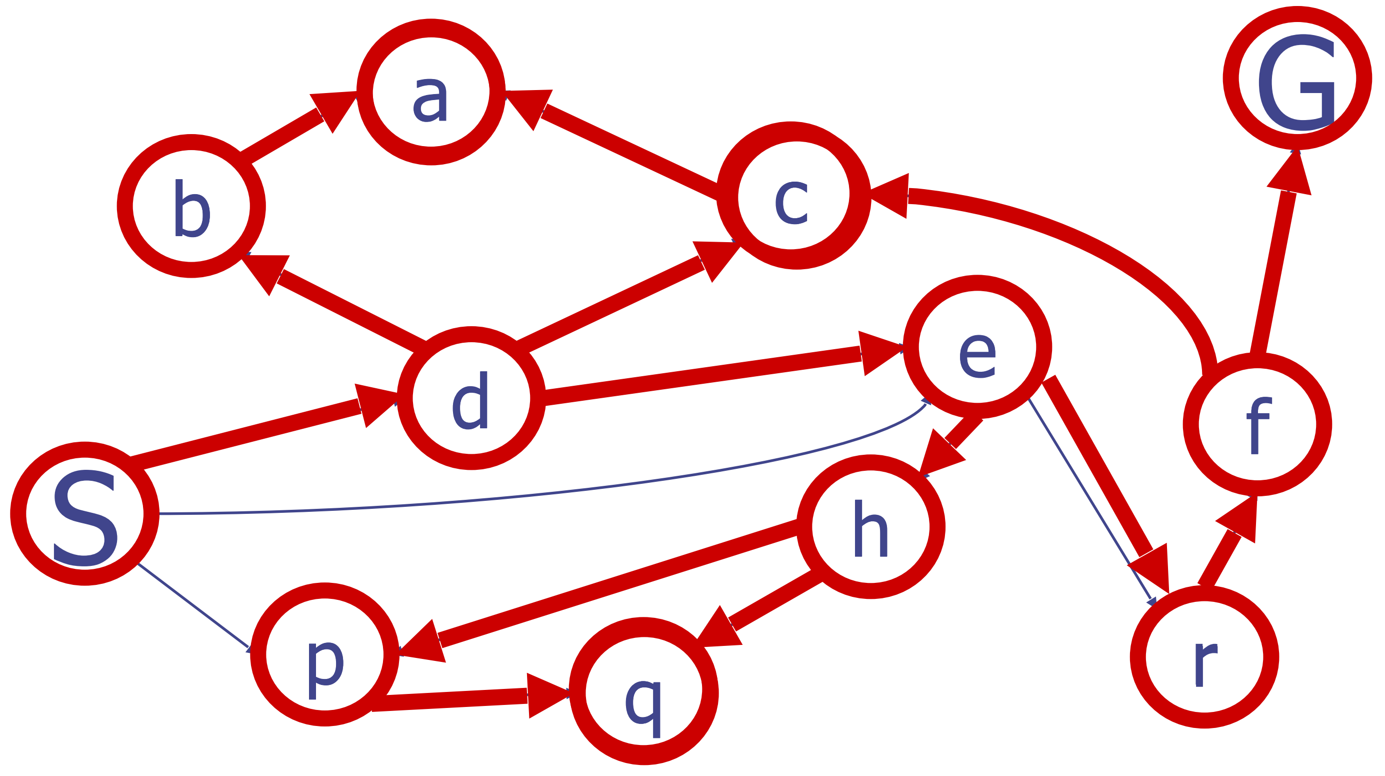
Recap and Upcoming

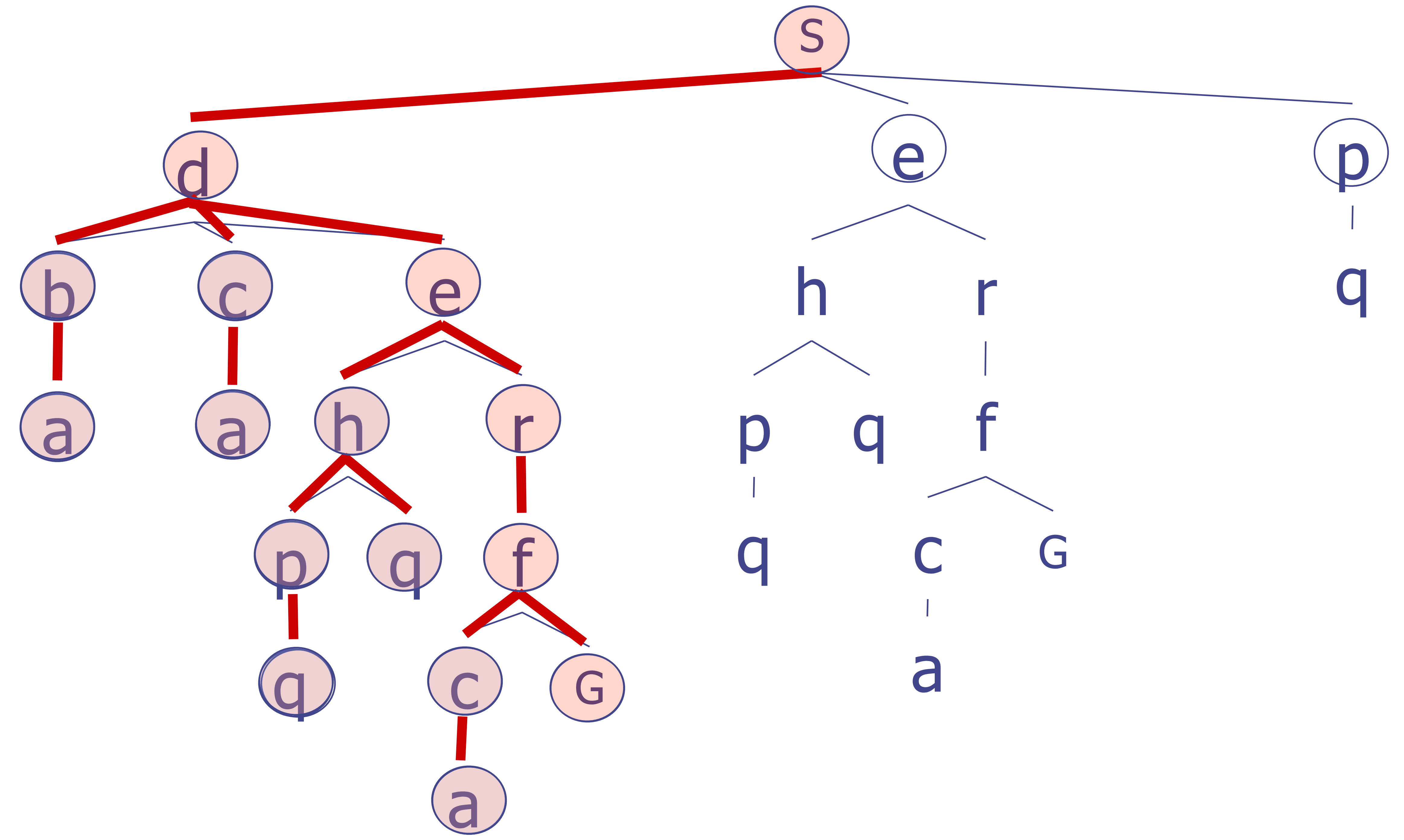
Today

- Uninformed search strategies
  - ID-DFS
  - Uniform Cost Search UCS
- Informed search strategies
  - Heuristics functions
  - Greedy Search algorithms
  - A* search algorithm

Review: Depth First Search

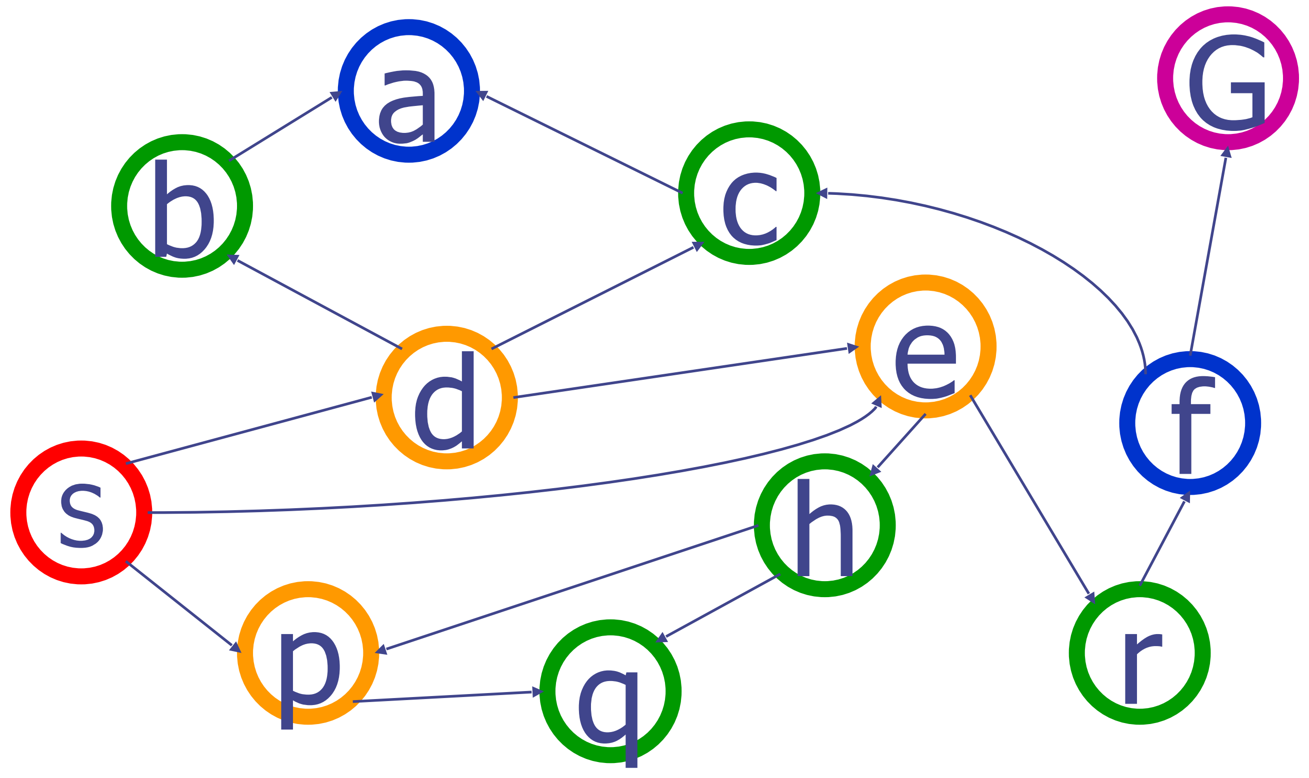
Strategy: expand deepest node first
Implementation: Fringe is a LIFO stack

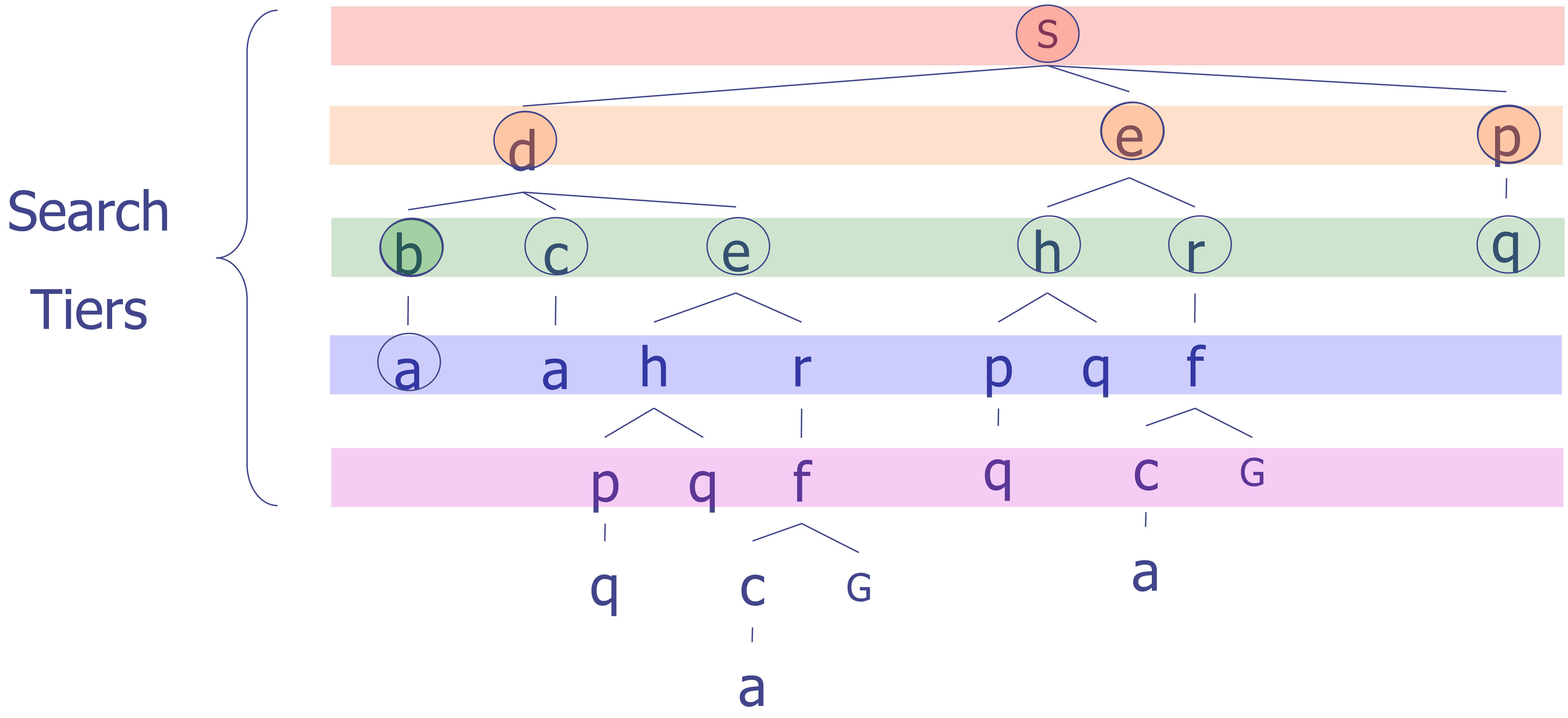




Breadth First Search

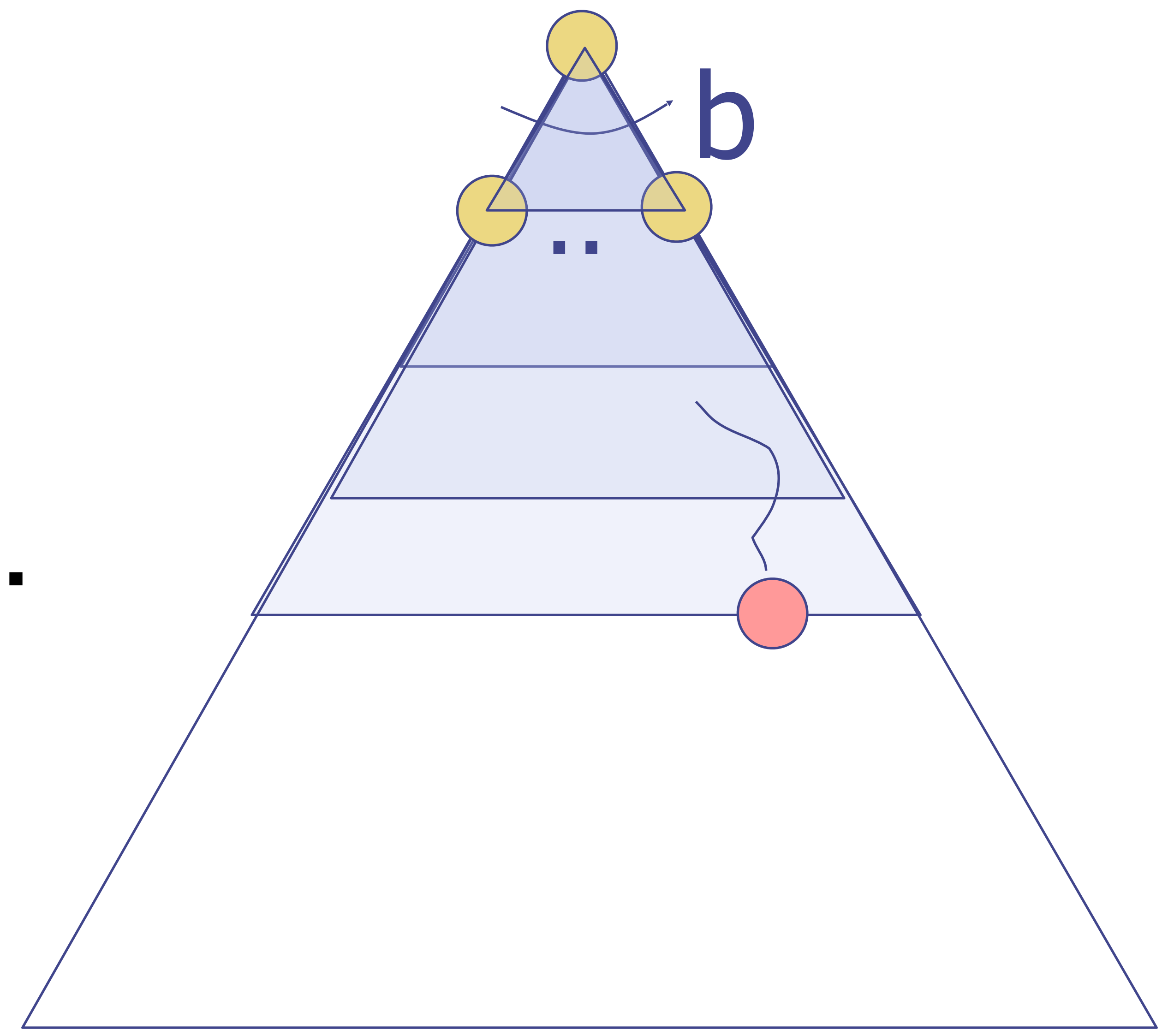
Strategy: expand shallowest node first
Implementation: Fringe is a FIFO queue





Iterative Deepening DFS

- Idea: get DFS's space advantage with BFSs 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 deepest level searched, so not so bad!

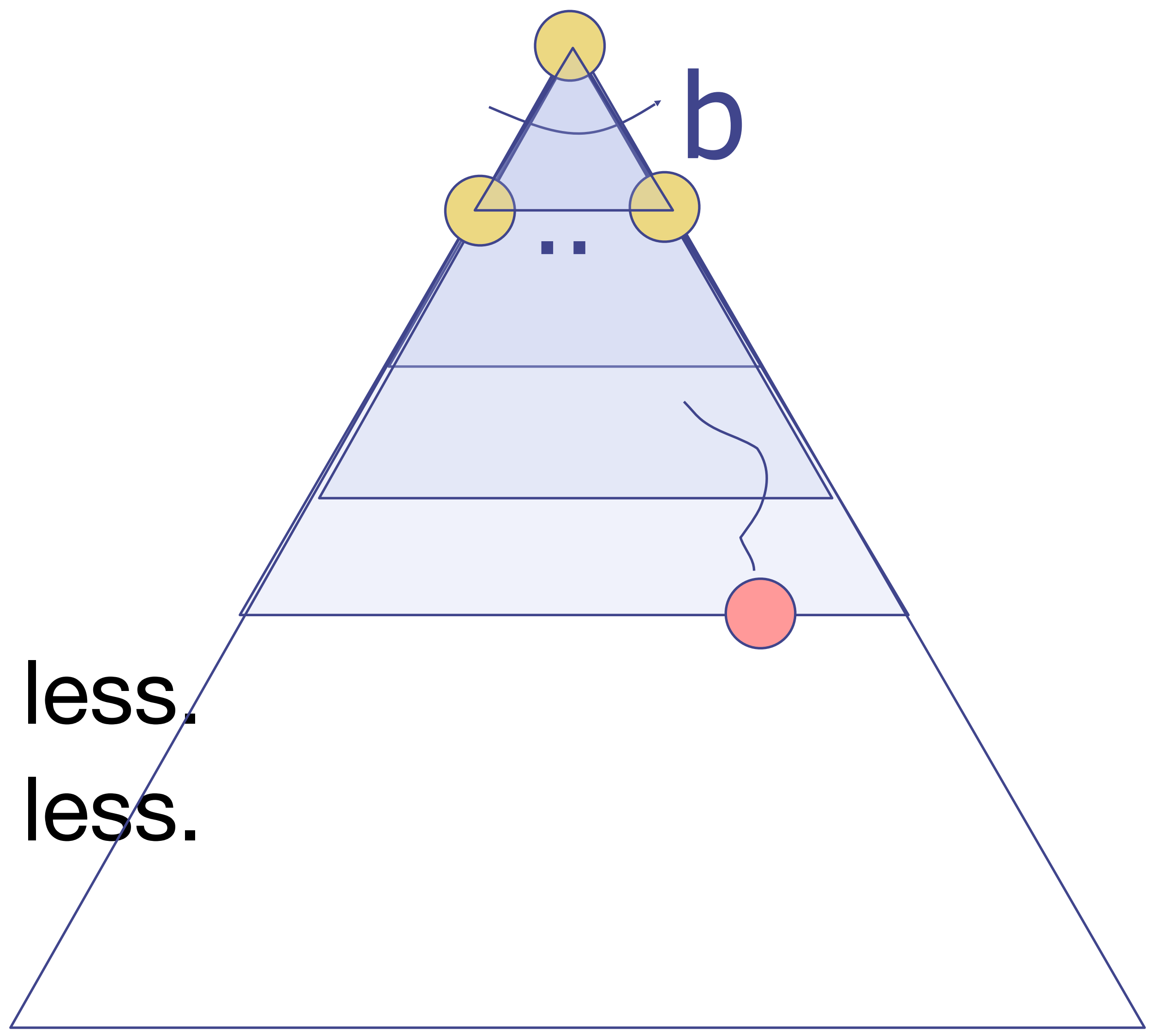


Iterative Deepening

Iterative deepening uses DFS as a subroutine:

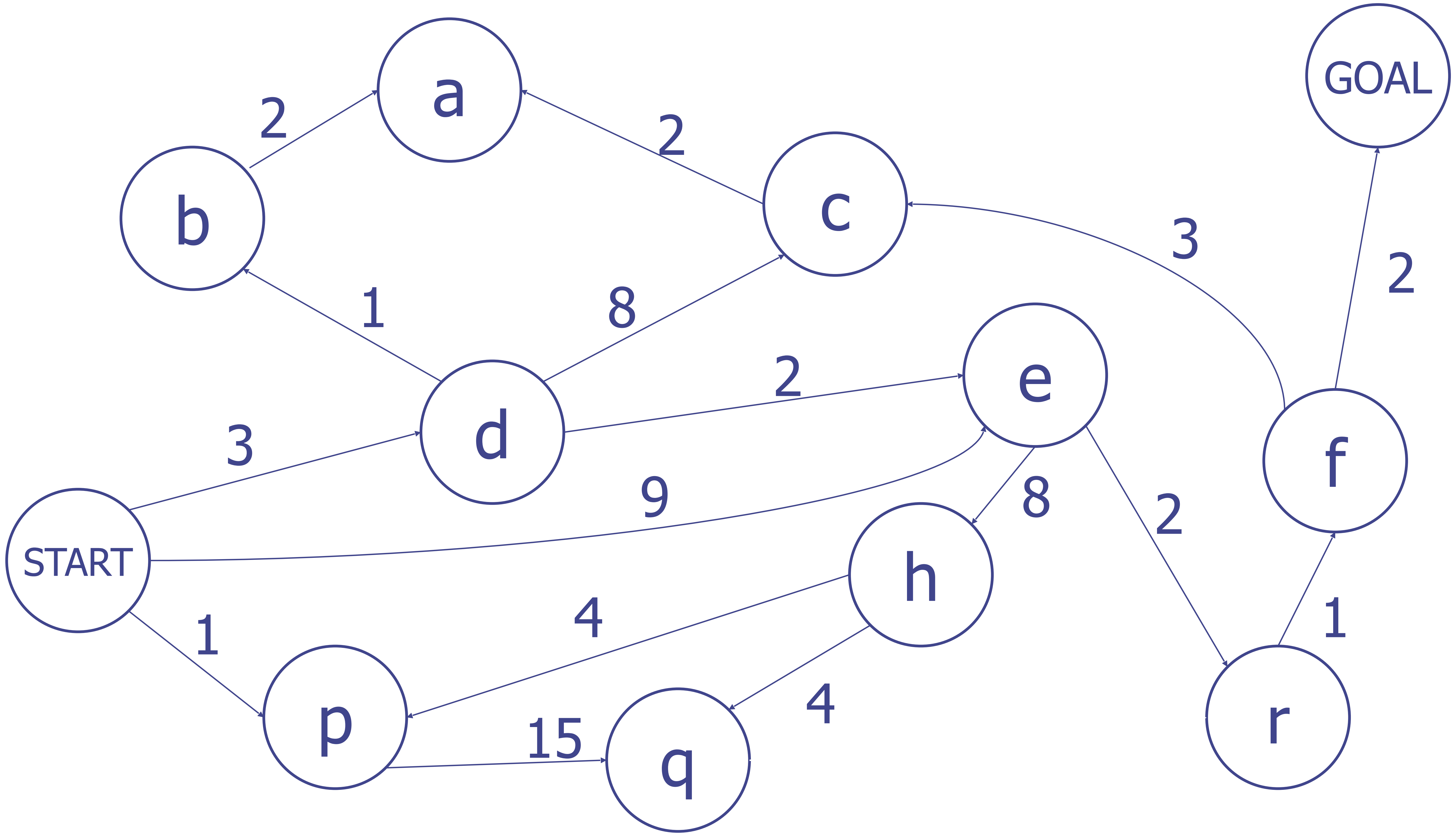
1. Do a DFS which only searches for paths of length 1 or less.
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N*	$O(b^{s+1})$	$O(b^s)$
ID		Y	N*	$O(b^{s+1})$	$O(bs)$

Cost-Sensitive Search



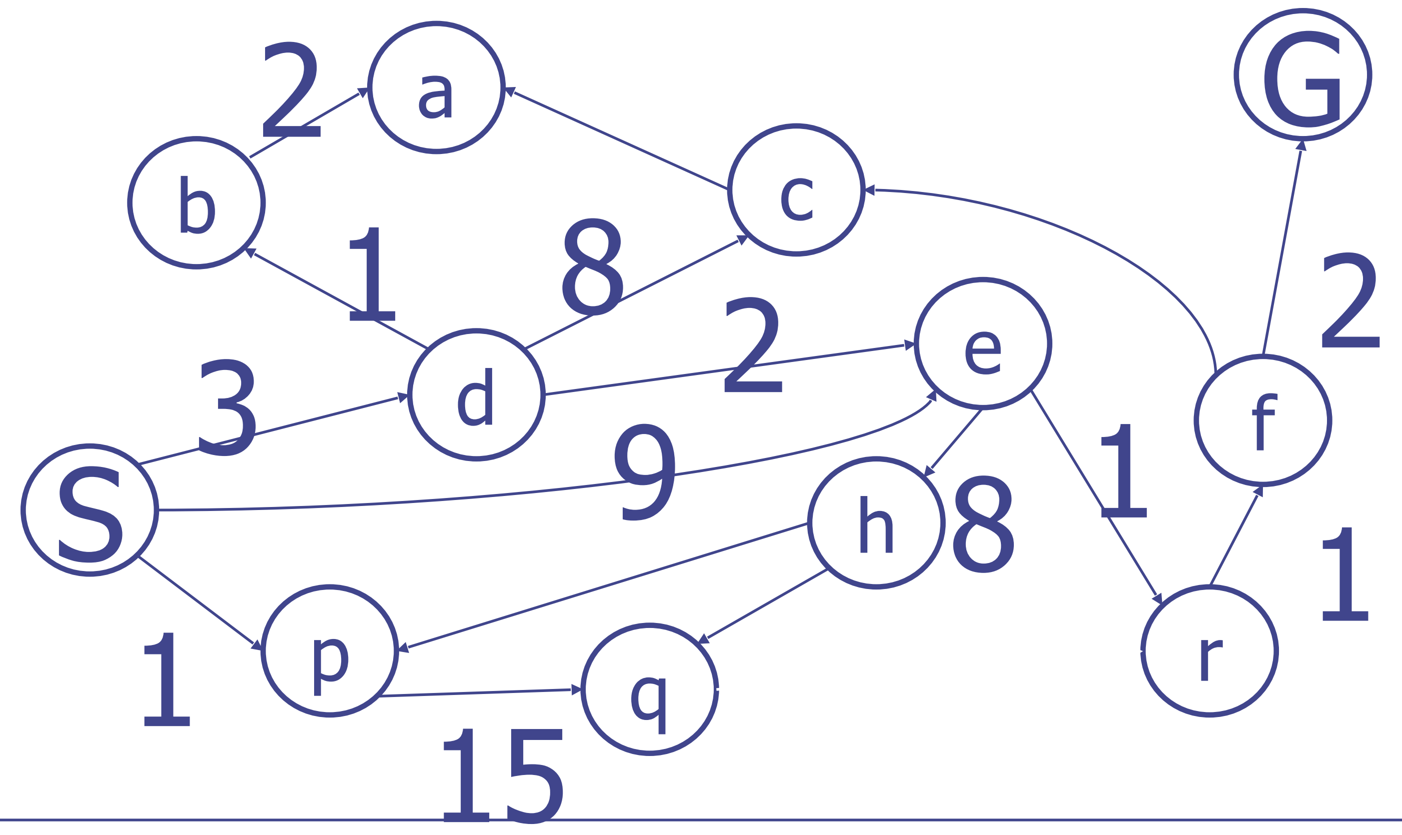
Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

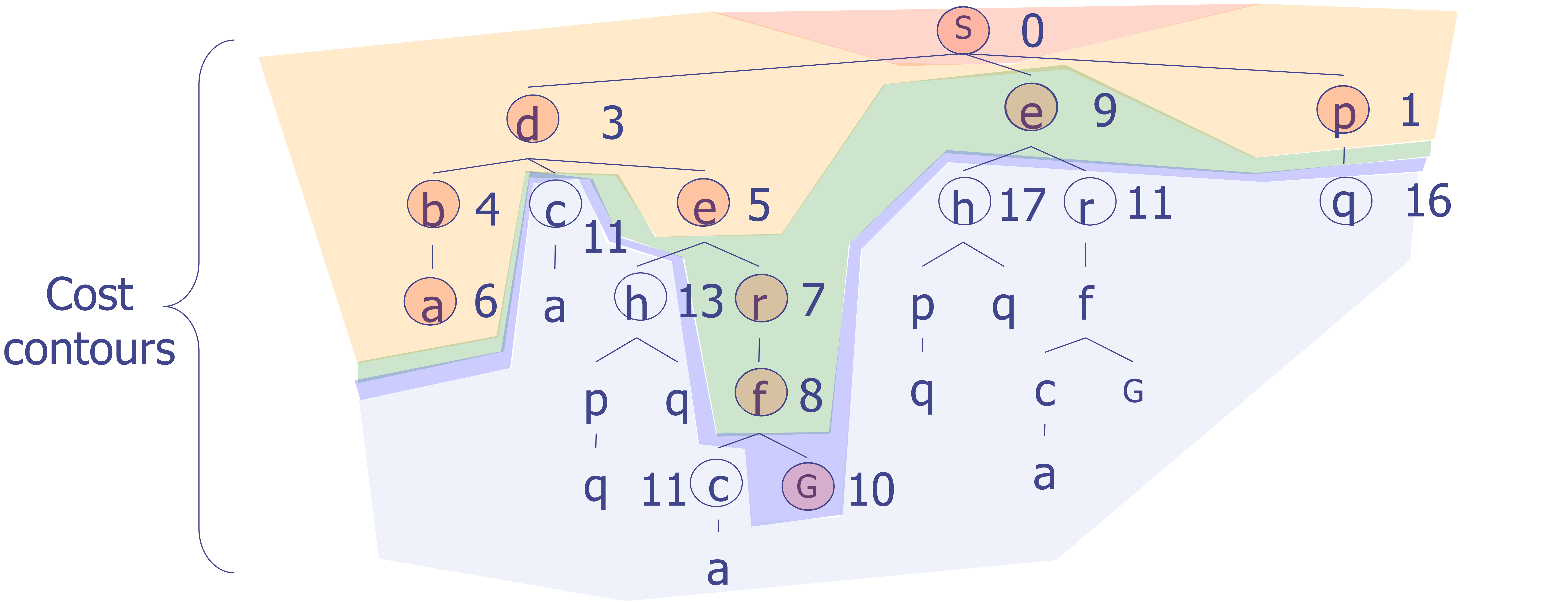
We will quickly cover an algorithm which does find the least-cost path.

Uniform Cost Search

Uniform Cost Search

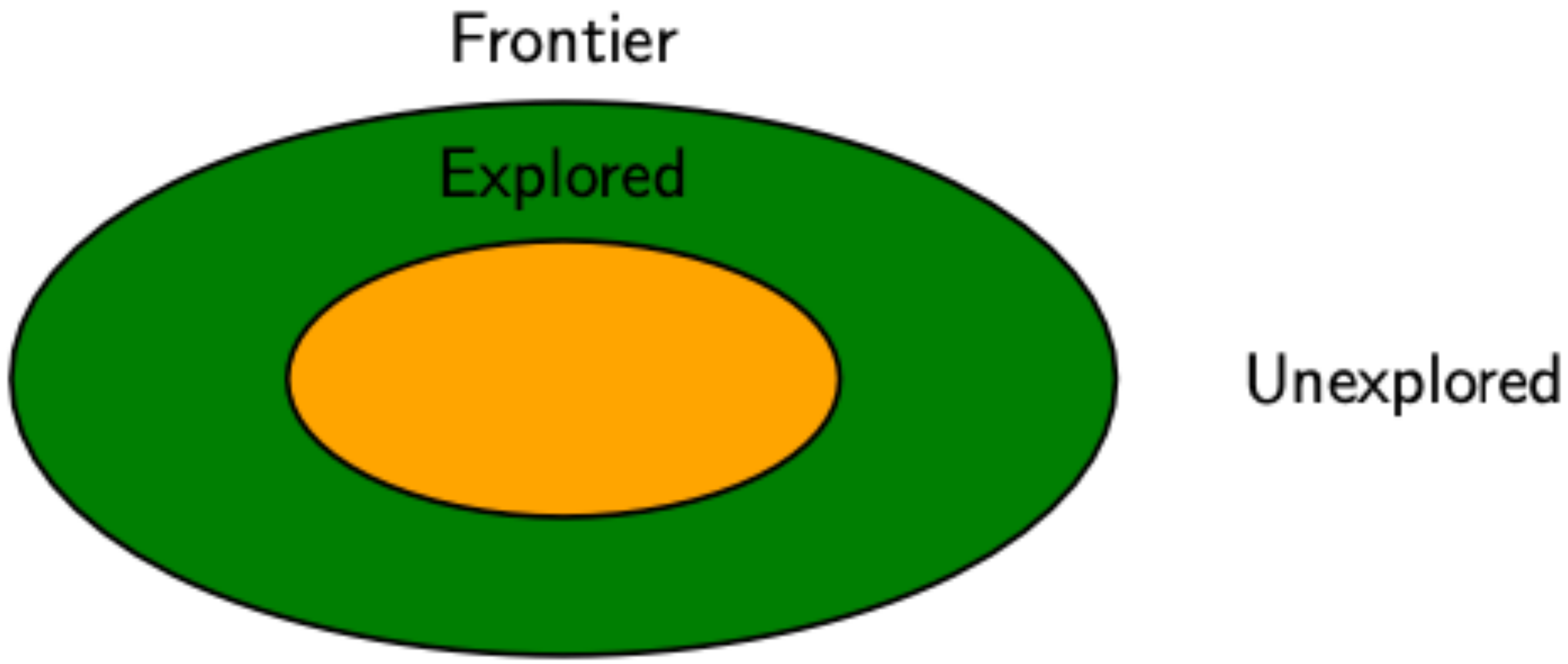
Expand cheapest node first:
Fringe is a priority queue





High level Strategy

- Explored: states we have found the optimal path to
- Frontier: states we have seen, still figuring out how to get there cheaply
- Unexplored: states we have not seen

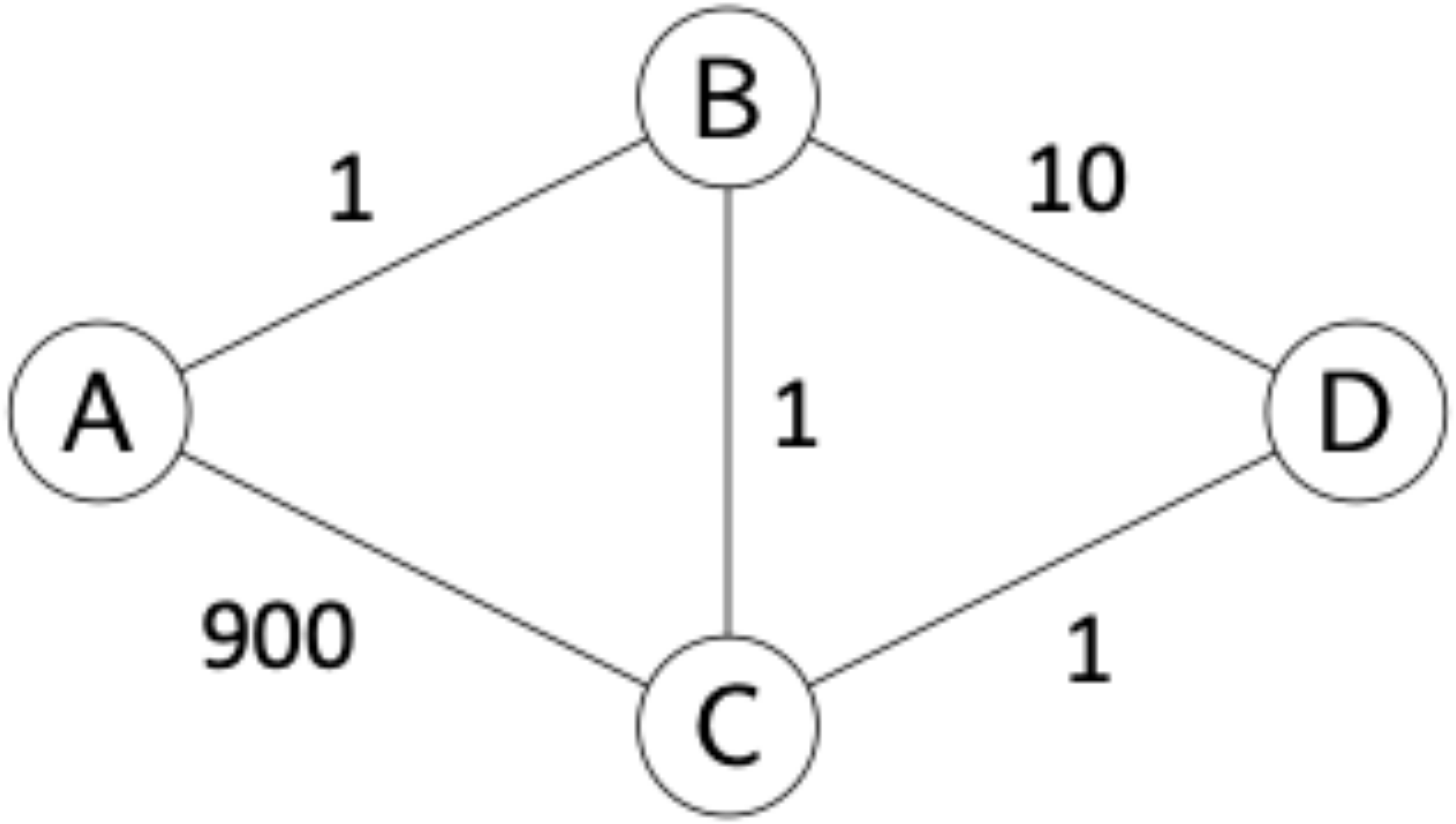


Algorithm

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
          else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

UCS Example

Example: UCS example



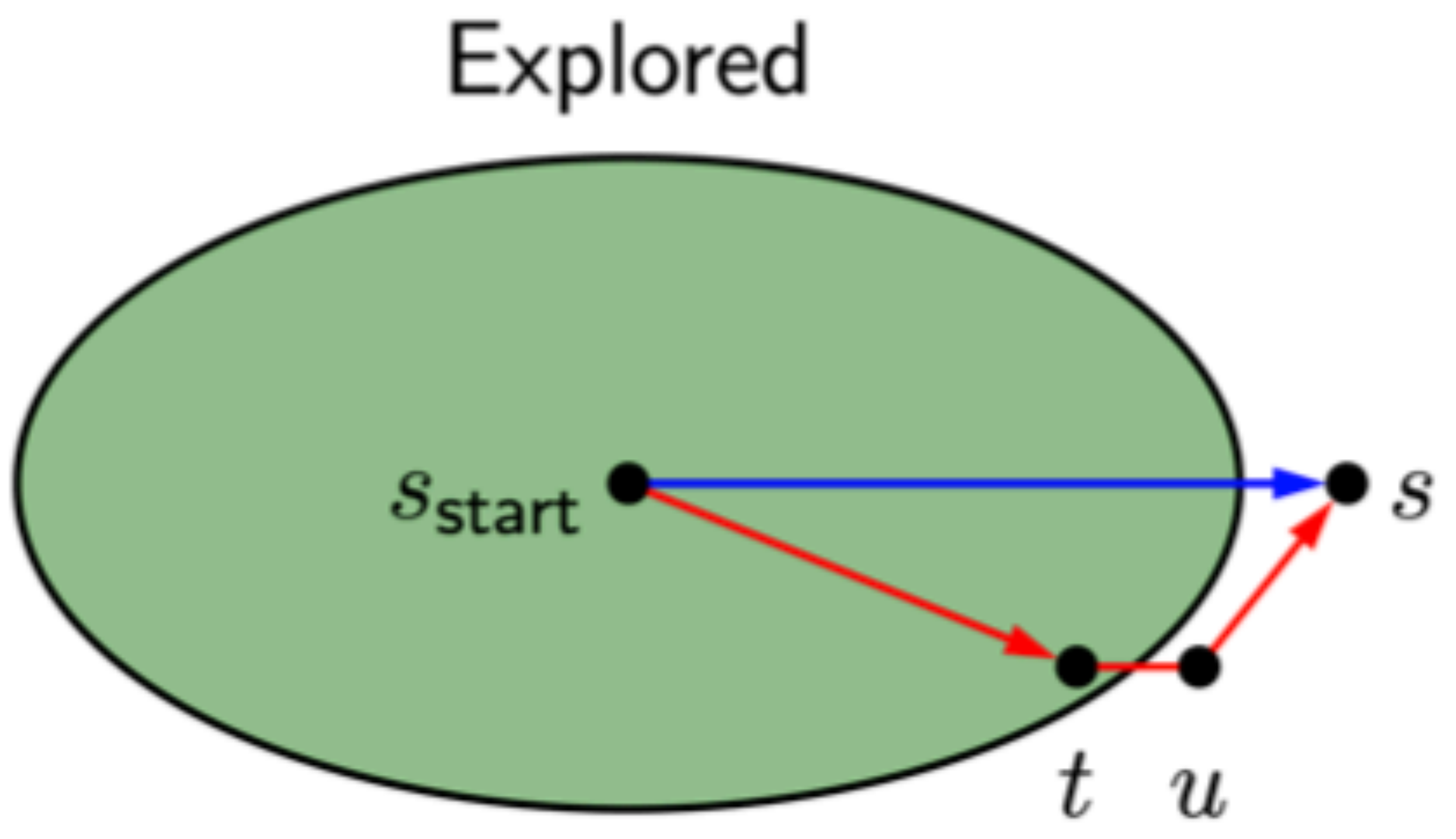
Start state: A, end state: D

Analysis of UCS

Theorem: correctness

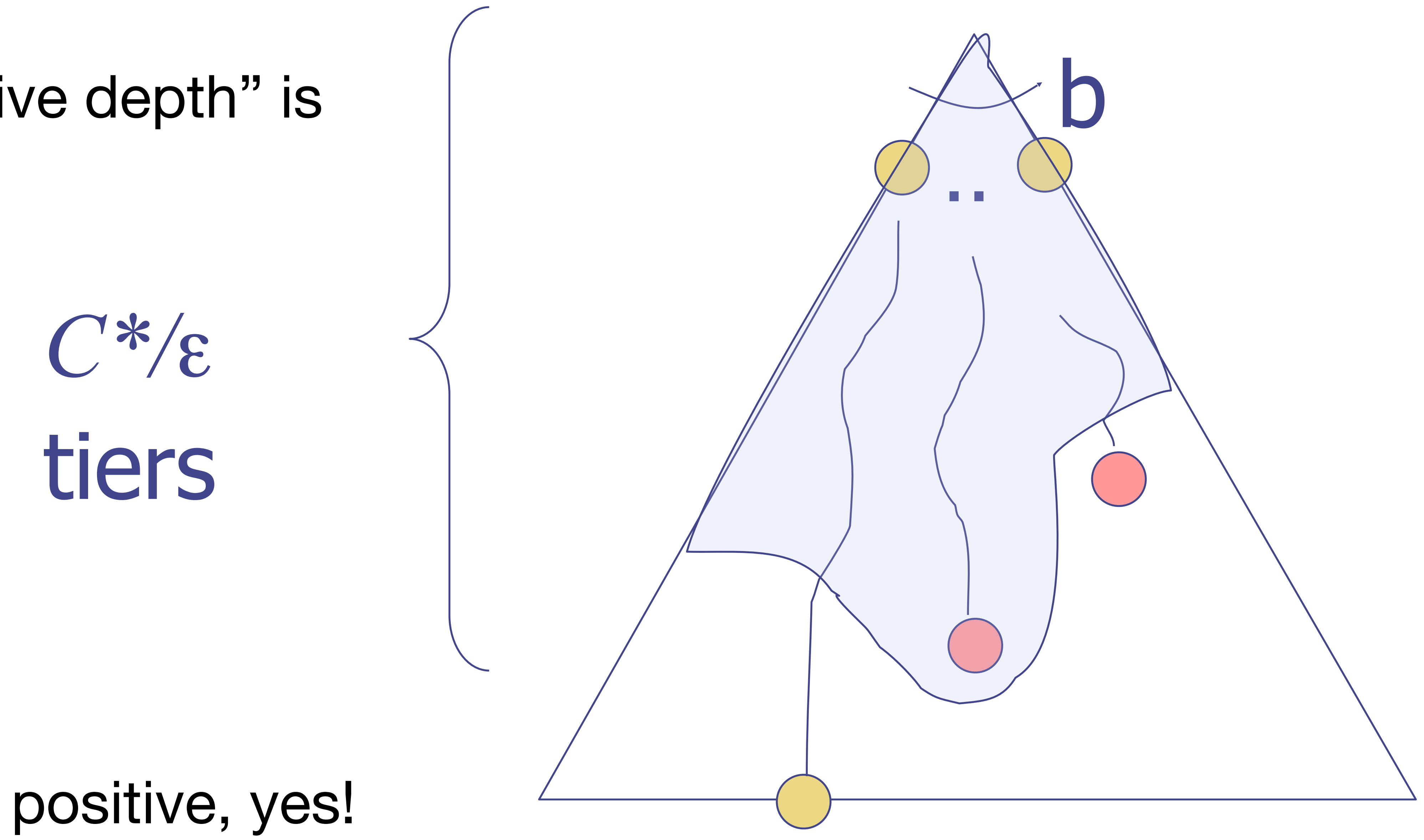
When a state s is popped from the frontier and moved to explored, its priority is PastCost(s), the minimum cost to s.

Proof:



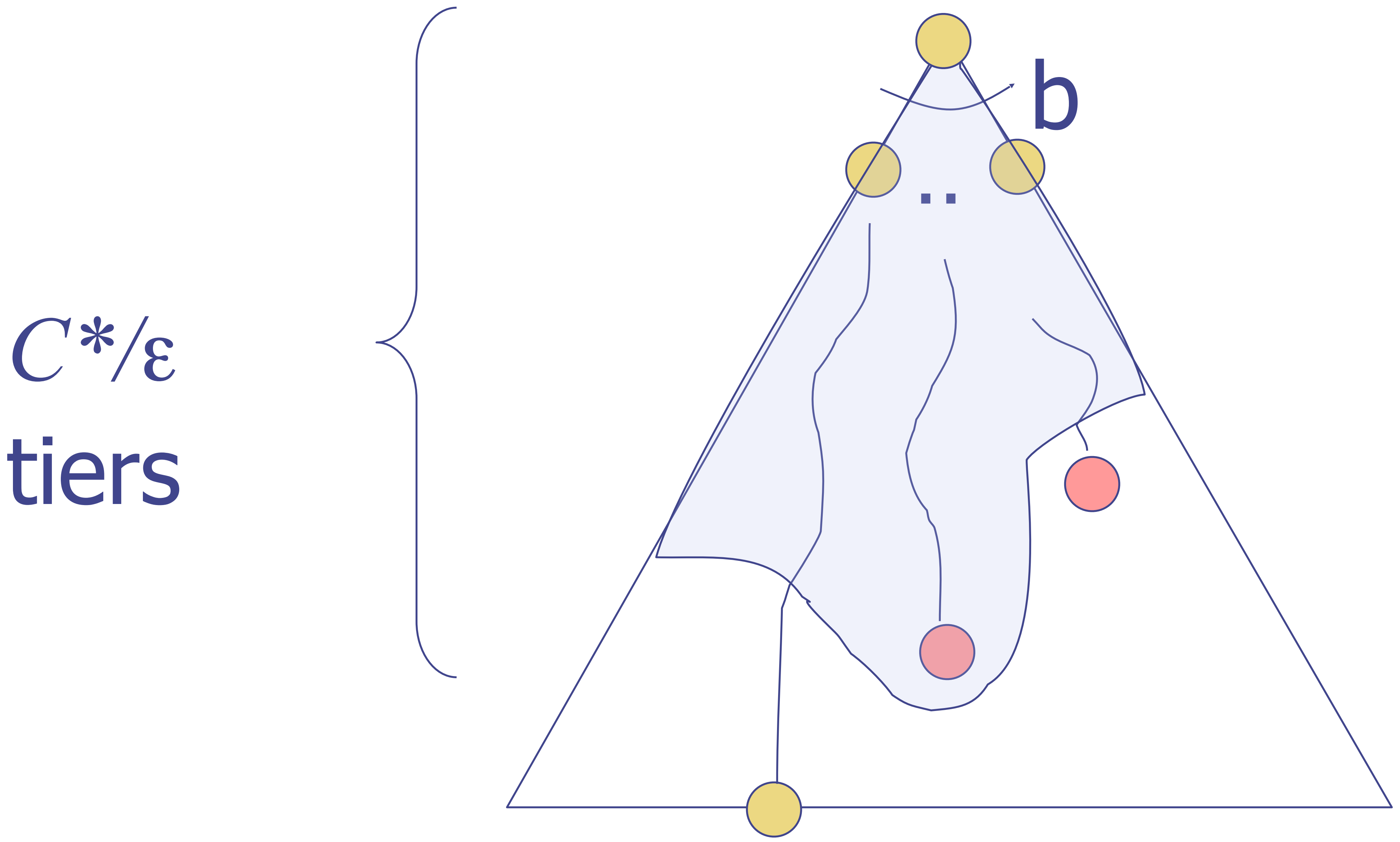
Uniform Cost Search (UCS) Properties

- 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 $C*/\epsilon$
  - Takes time $O(b^{C*/\epsilon})$ (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C*/\epsilon})$
- • Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  • Yes! (Proof in textbook)



Uniform Cost Search

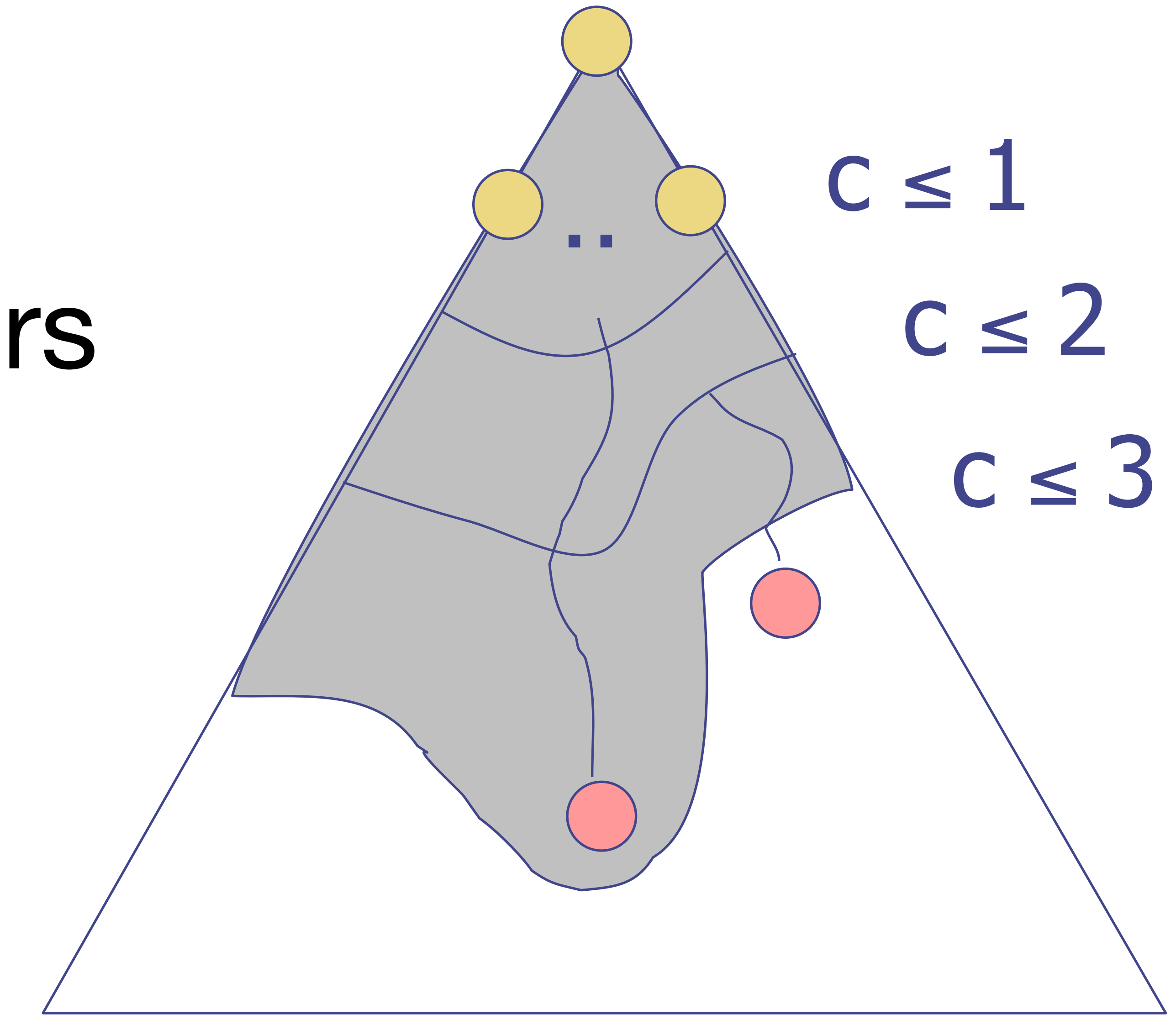
Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N	$O(b^{s+1})$	$O(b^{s})$
UCS		Y*	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$

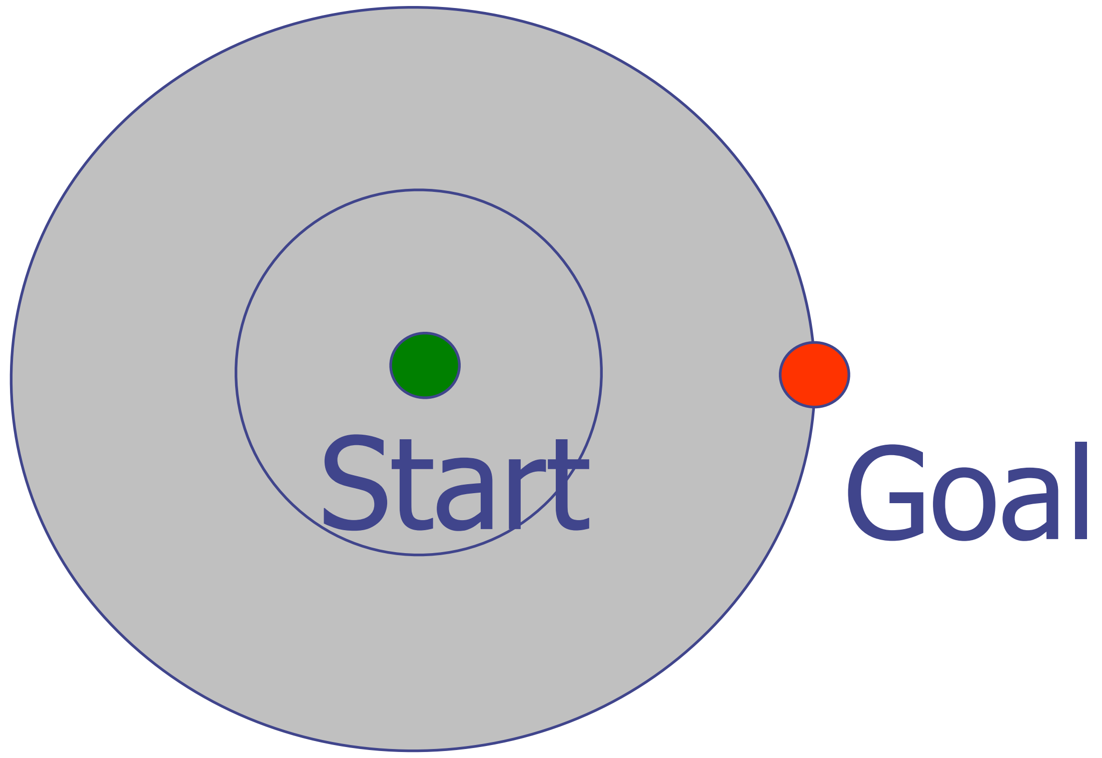


*** UCS can fail if actions can get arbitrarily cheap**

Uniform Cost Issues

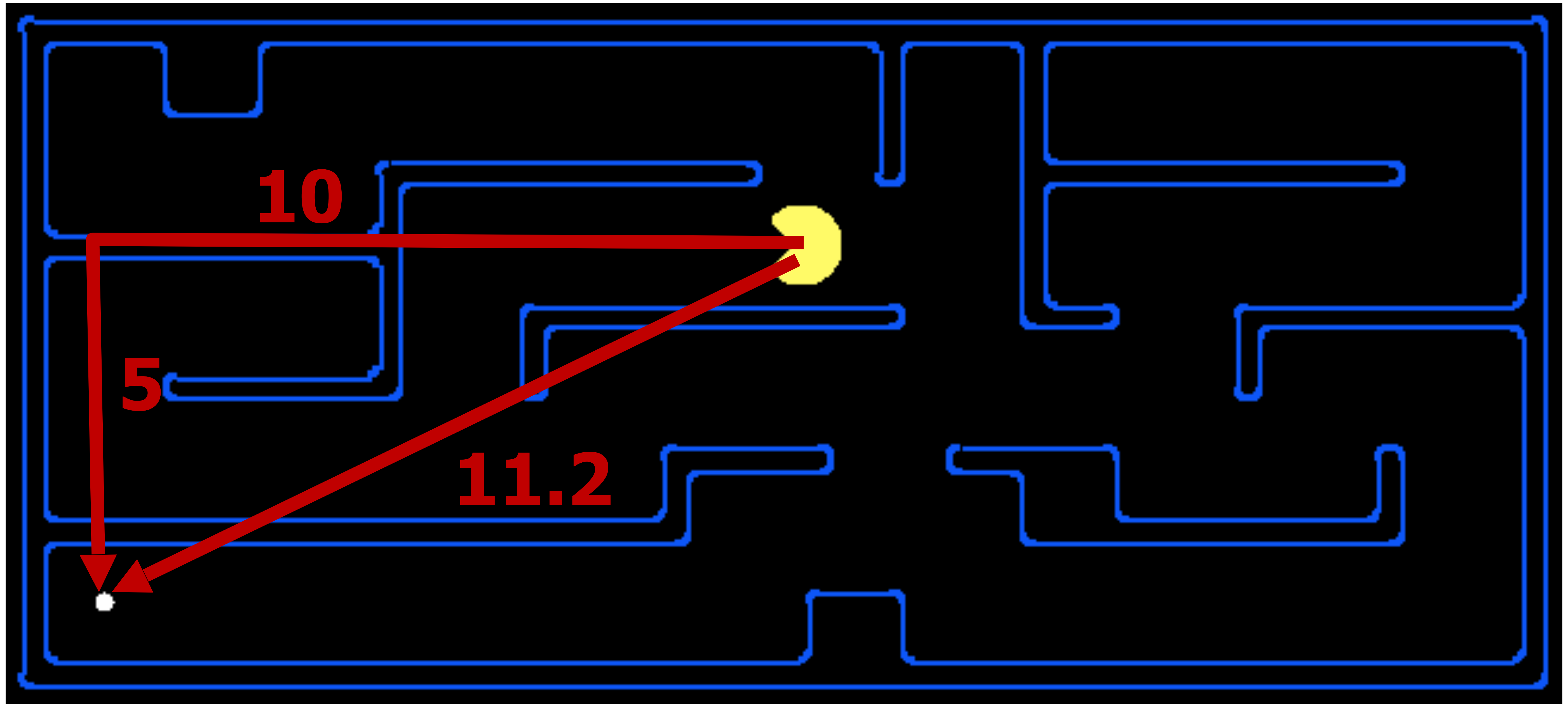
- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location





Search Heuristics

- Any estimate of how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance

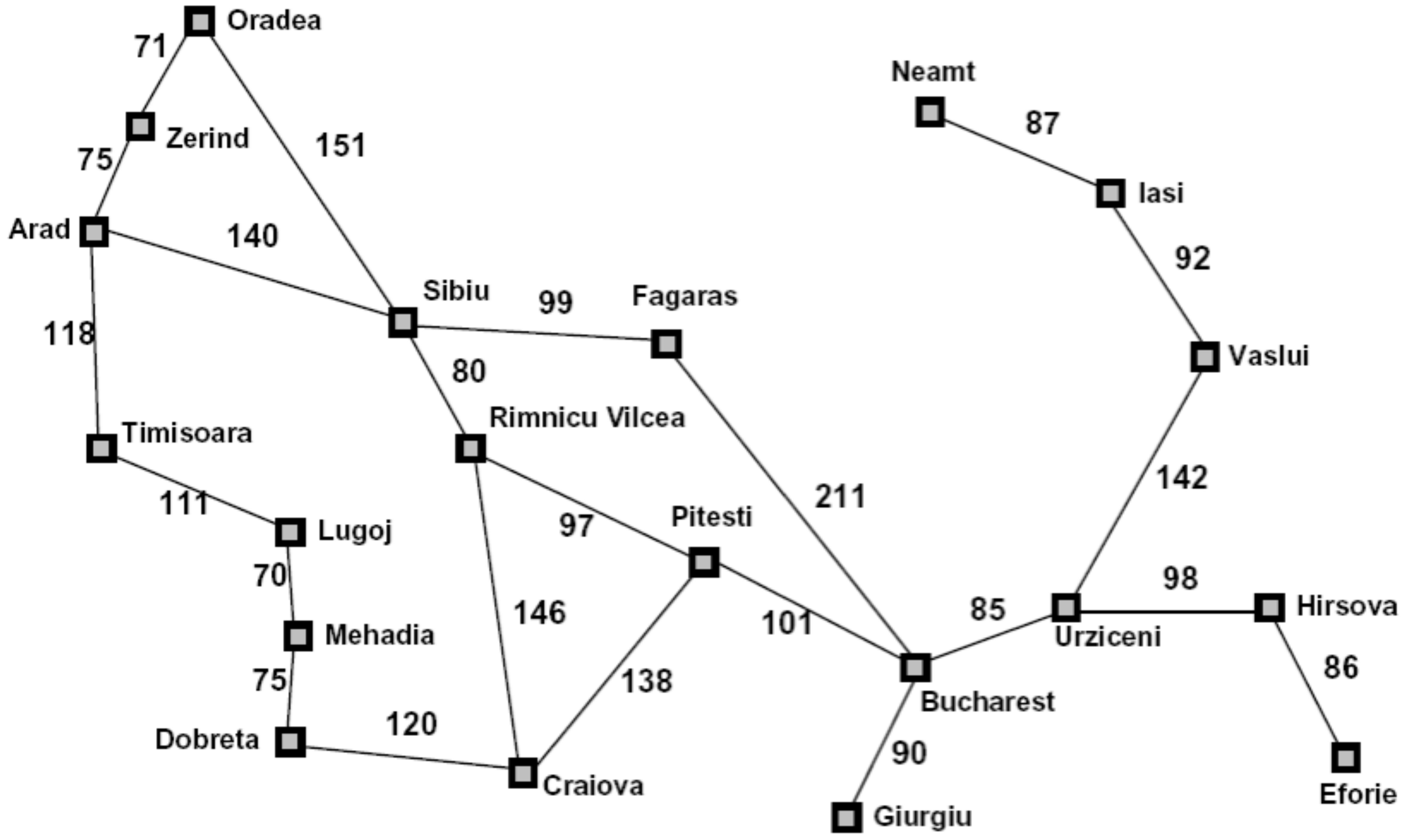


CE 3: Heuristic brainstorm

- Heuristic for a specific search problem.
- Doesn’t have to be optimal (just start thinking about it)
- Example: get a fruit and cereal from the grocery store.
- Heuristic: dist(fruit) + dist(cereal)

Take ~5 minutes

Example of a Heuristic Function



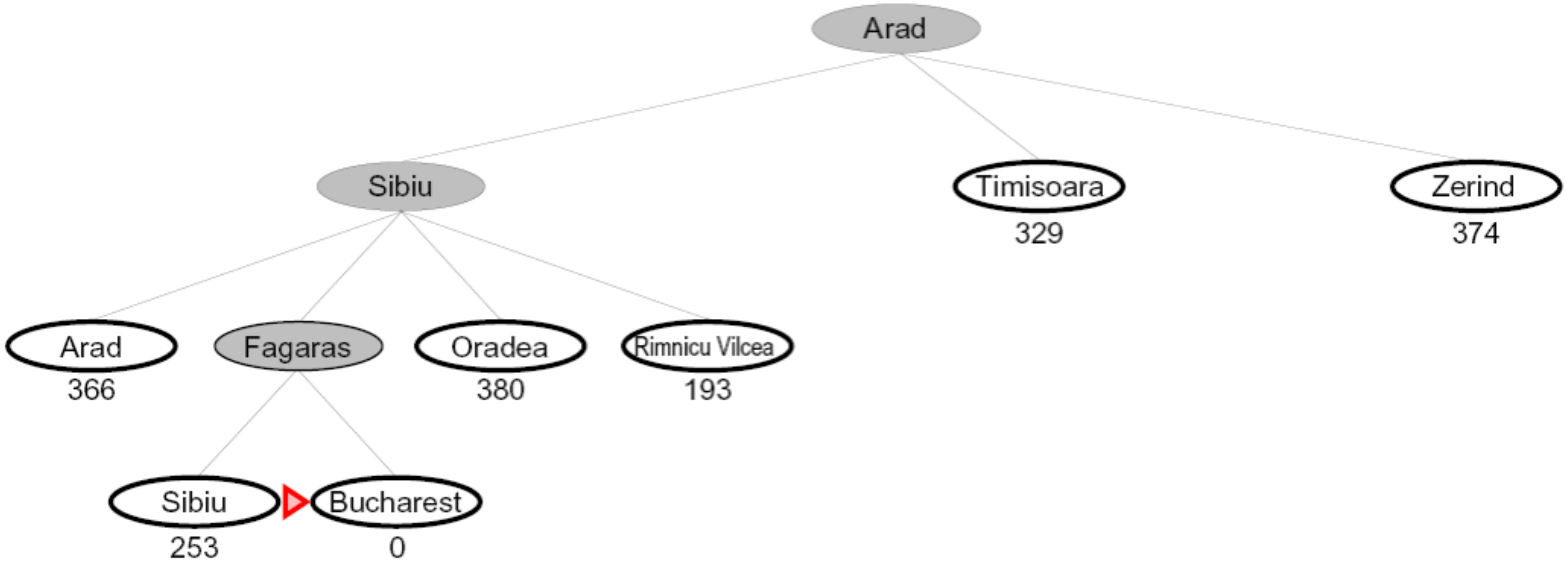
Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Search

Best First / Greedy Search

- Expand the node that seems closest…



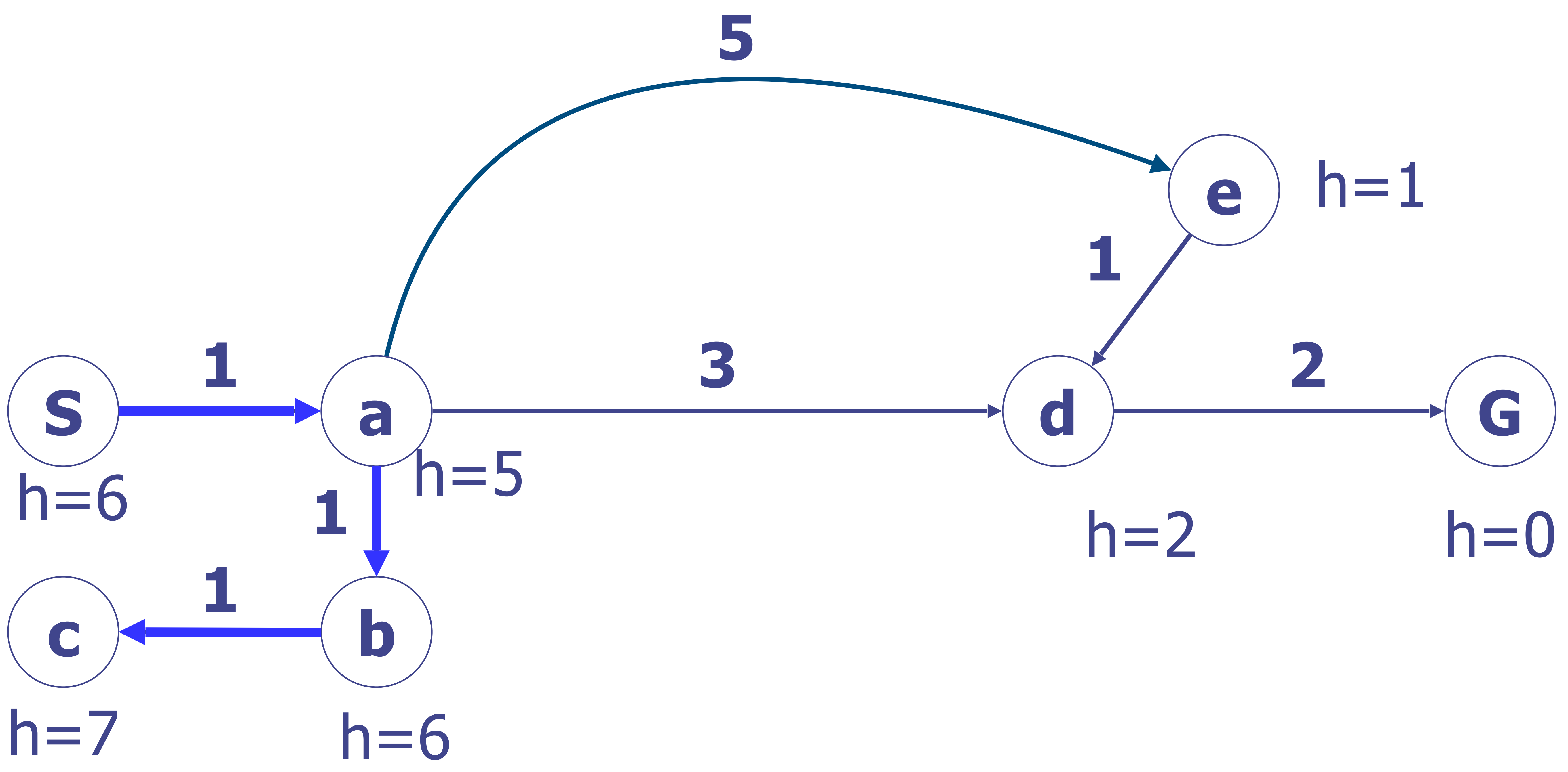
- What can go wrong?

A* Search

Combining UCS and Greedy

$$f(n) = g(n)$$

- Uniform-cost orders by path cost, or backward cost g(n)



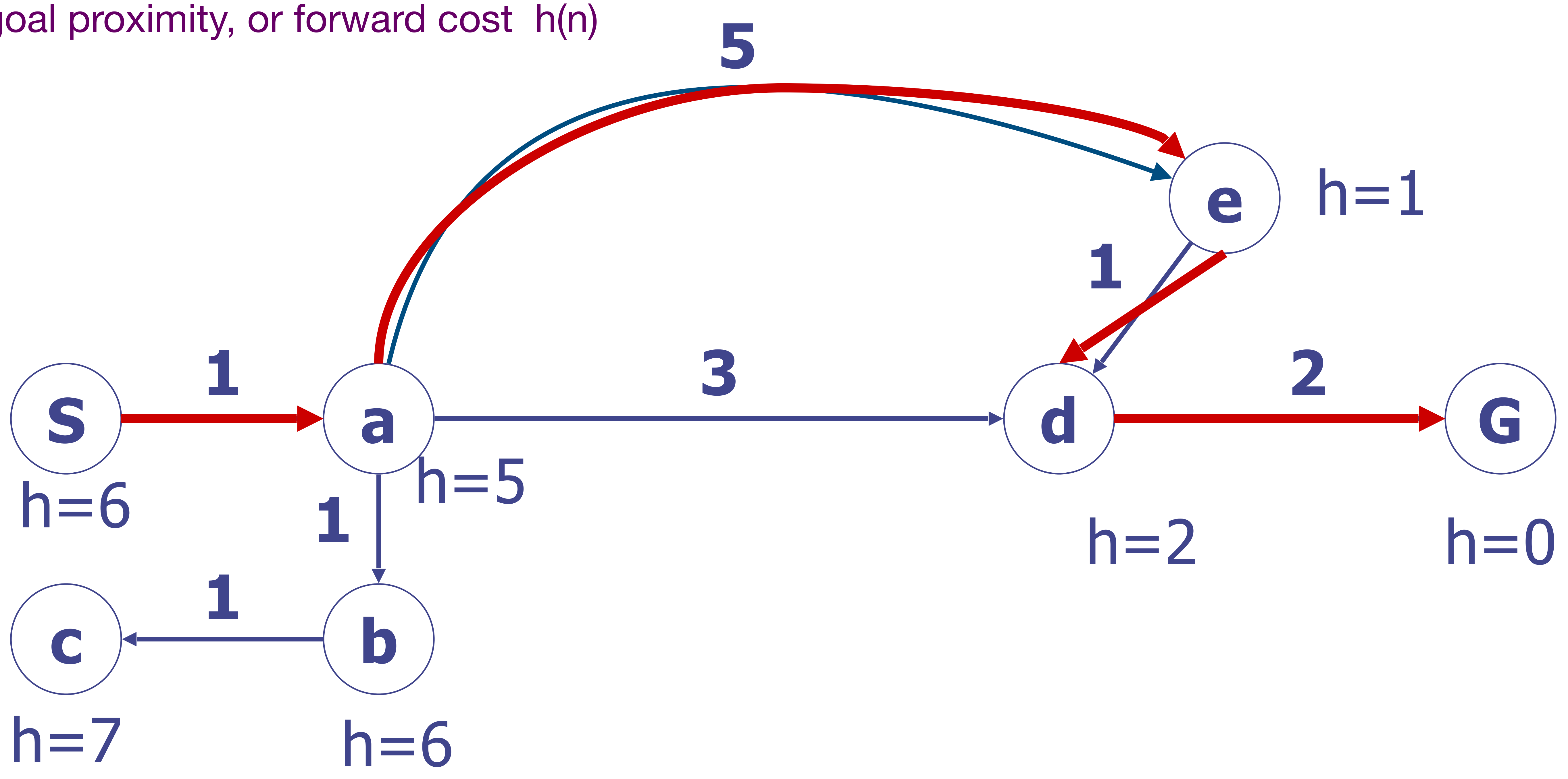
Node	Fringe	f(n)
s	s->a	1
s->a	s->a->b	2
s->a	s->a->d	4
s->a	s->a->e	6

Example: Teg Grenager

Combining UCS and Greedy

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)

f(n) = h(n)



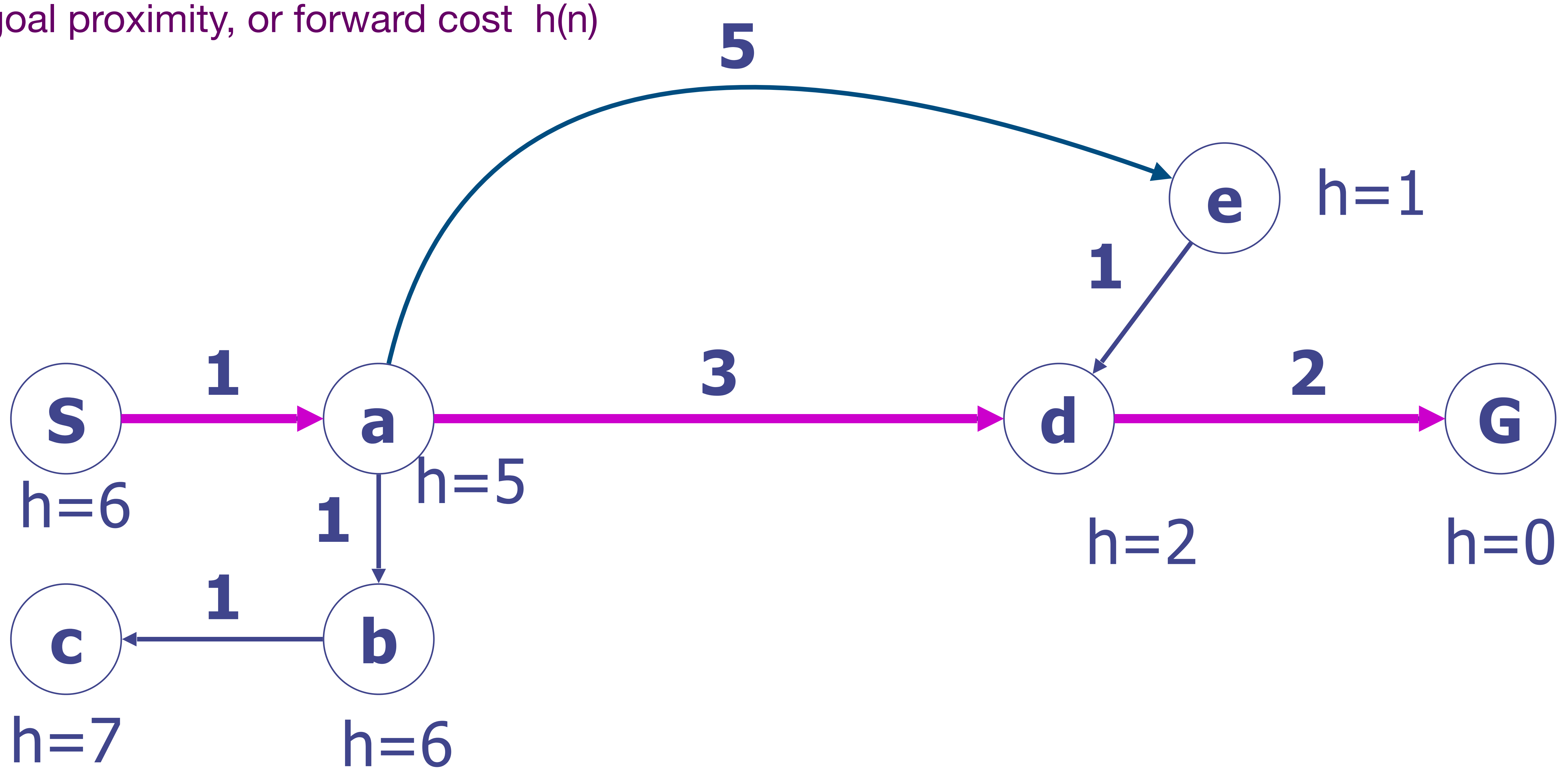
Node	Fringe	f(n)
s	s->a	**5**
s->a	s->a->b	**6**
s->a	s->a->d	**2**
s->a	s->a->e	**1**

Example: Teg Grenager

Combining UCS and Greedy

f(n) = g(n) + h(n)

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)



Node	Fringe	f(n)
s	s->a	**6**
s->a	s->a->b	**8**
s->a	s->a->d	**6**
s->a	s->a->e	**7**

- A* Search orders by the sum: f(n) = g(n) + h(n)

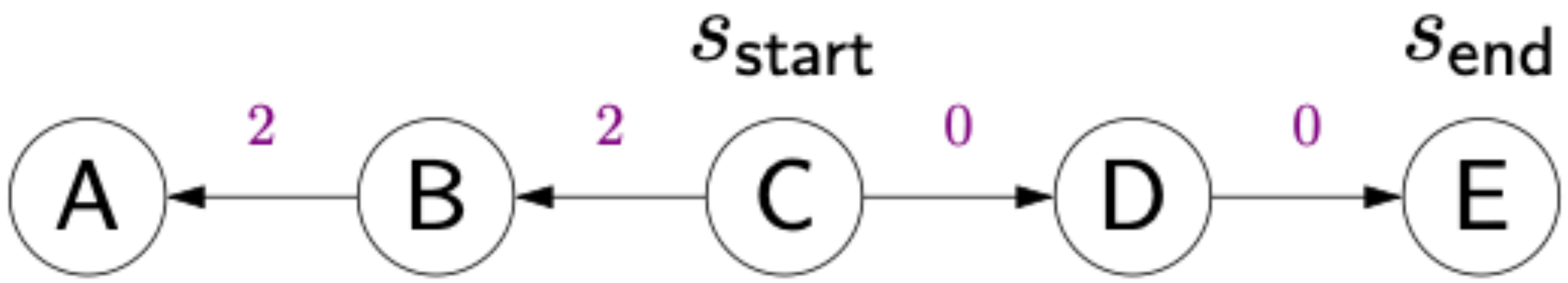
Example: Teg Grenager

Another Way to Implement A*

Run UCS with modified edge costs in order to account for closeness to the goal state

$$Cost'(s,a) = Cost(s,a) + h(succ(s,a)) - h(s)$$

- Intuition: add a penalty for how much action 'a' takes us away from the end state

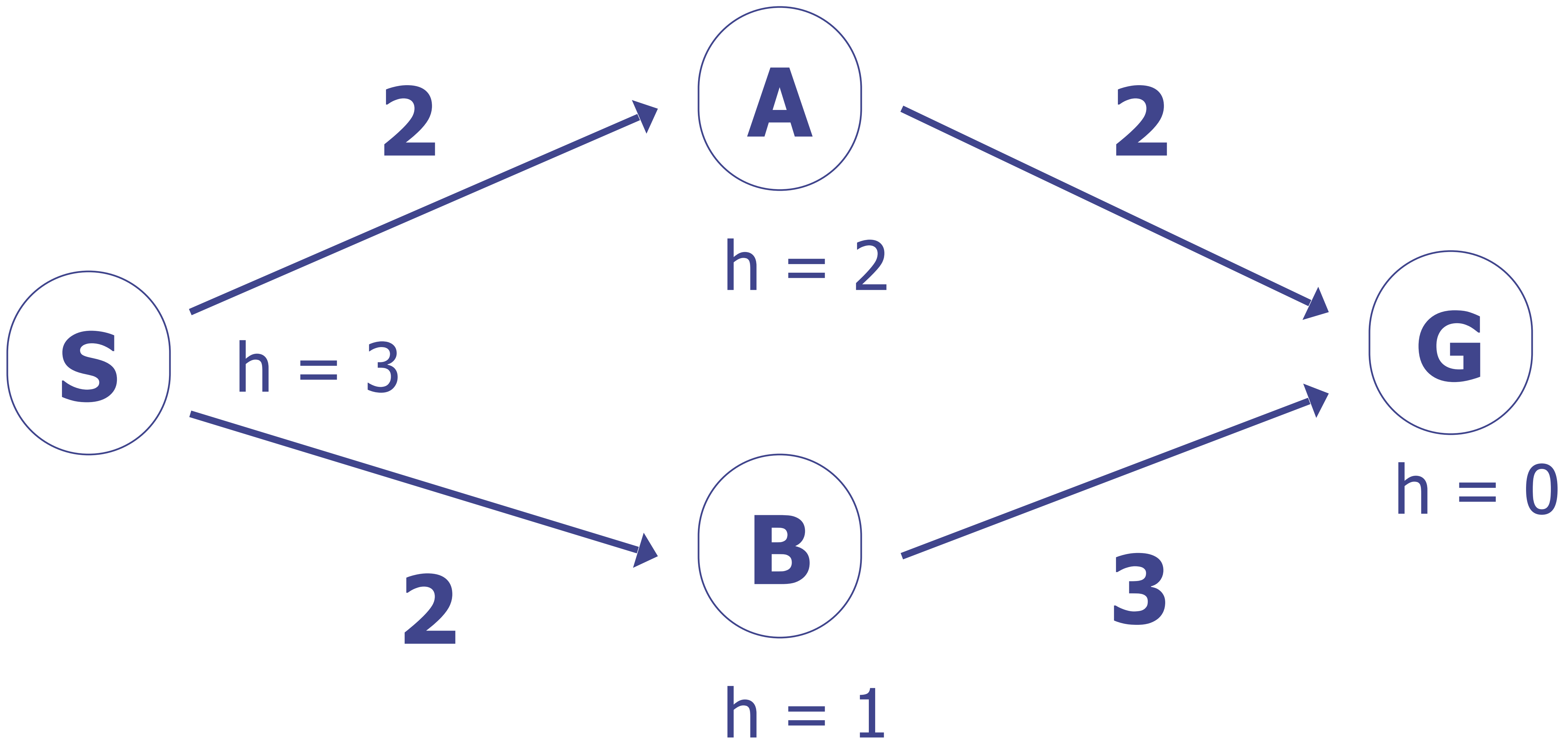


$$h(s) = 4 \quad 3 \quad 2 \quad 1 \quad 0$$

$$\text{Cost}'(C,B) = \text{Cost}(C,B) + h(B) - h(C) = 1 + (3-2) = 2$$

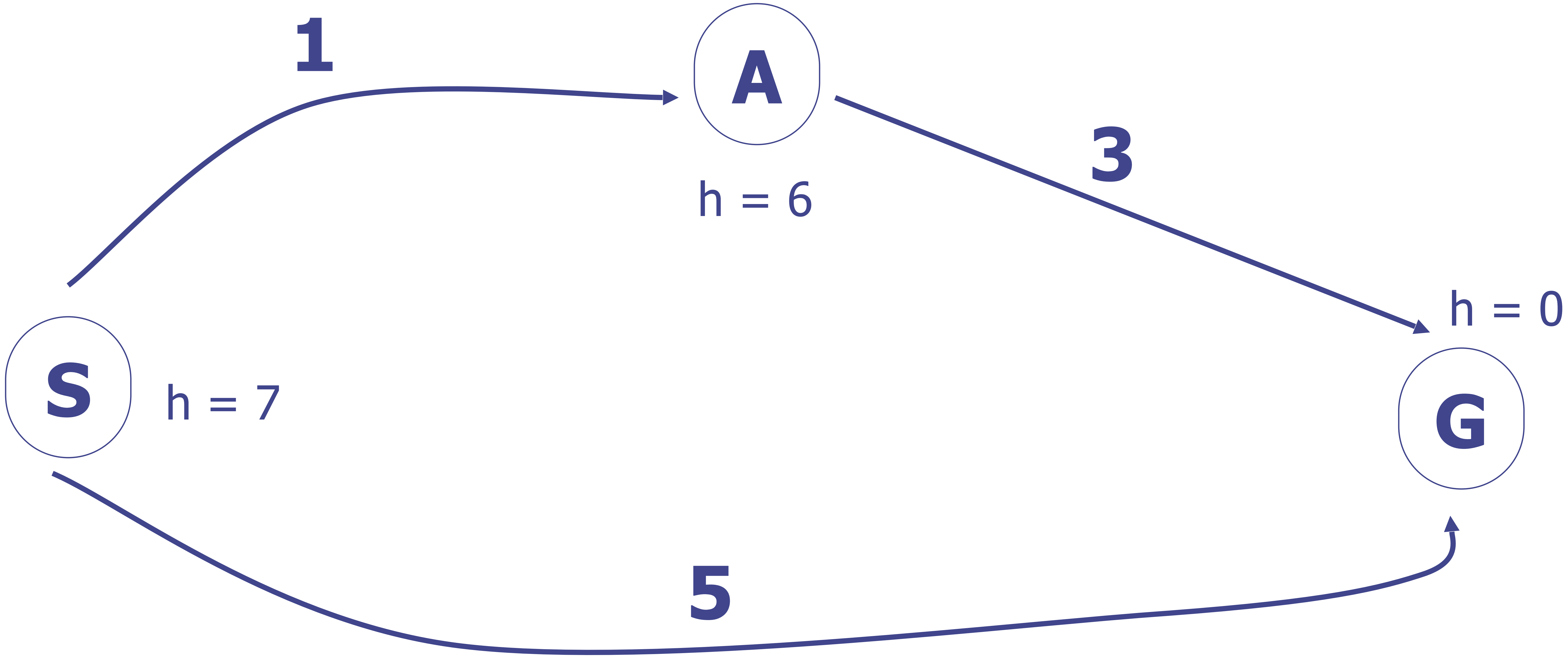
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Is A* Optimal?



- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

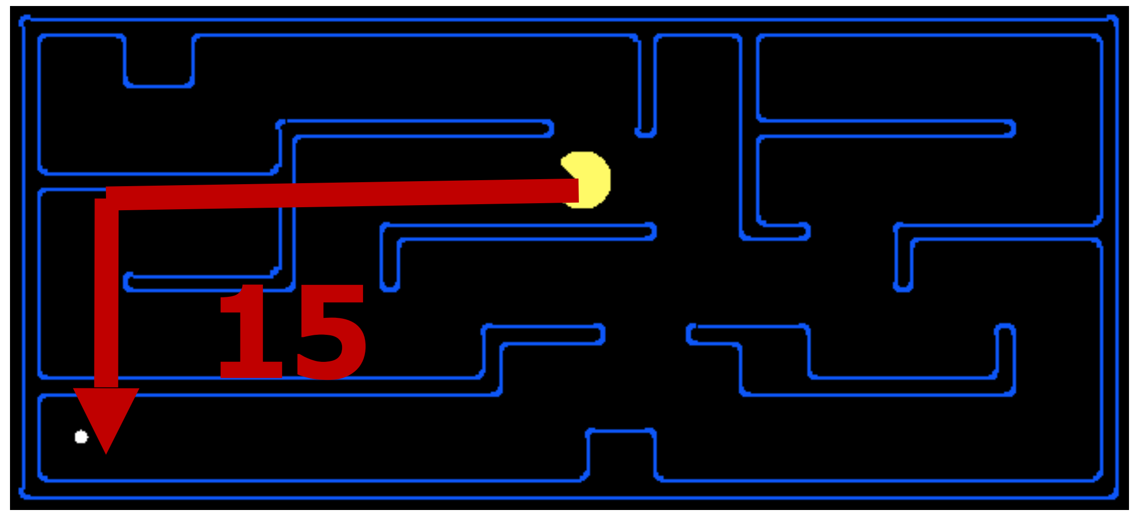
Admissible Heuristics

- A heuristic h is admissible (optimistic) if:

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

- where $h^*(n)$ is the true cost to a nearest goal

- Example:

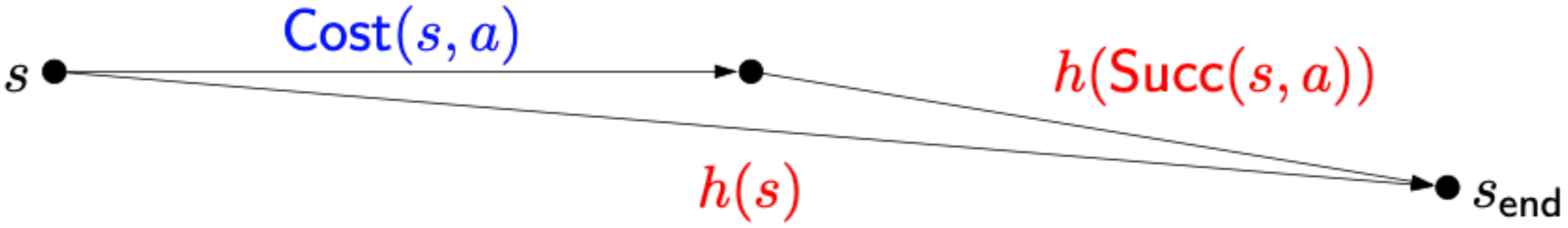


- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Consistent Heuristic

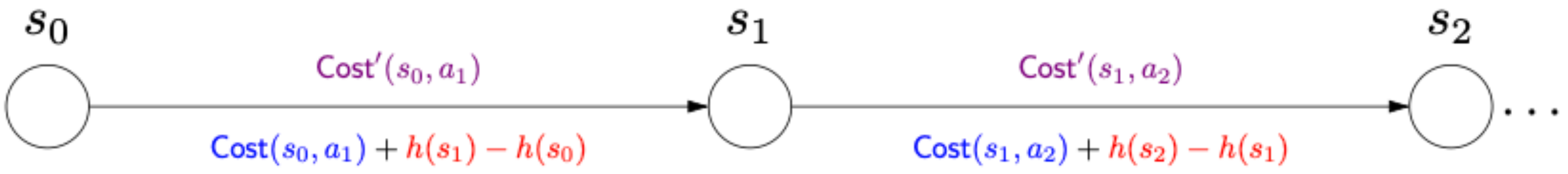
A heuristic h is "**consistent**" if

- $Cost'(s,a) = Cost(s,a) + h(succ(s,a)) - h(s) \geq 0$
- $h(s_{end}) = 0$



Correctness of A*

- If h is consistent, A* returns the minimum cost path
- Consider any path
- Key identity:



$$\underbrace{\sum_{i=1}^{L} \text{Cost}'(s_{i-1}, a_i)}_{\text{modified path cost}} = \underbrace{\sum_{i=1}^{L} \text{Cost}(s_{i-1}, a_i)}_{\text{original path cost}} + \underbrace{h(s_L) - h(s_0)}_{\text{constant}}$$

- Therefore, A* solves the original problem using UCS and therefore the algorithm has complete

Efficiency of A*

A* explores all states satisfying
$$\boldsymbol{f(s) \leq f(s_{end}) - h(s)}$$

- Interpretation: the larger h(s), the better
- Proof: A* explores all nodes 's' such that

$$\boldsymbol{f(s) + h(s) \leq f(s_{end}) + h(s_{end})}$$
$$\boldsymbol{f(s) + h(s) \leq f(s_{end})}$$
$$\boldsymbol{f(s) \leq f(s_{end}) - h(s)}$$

Recap

Week 2

- **Solving problems by searching (cont.)**
  - **Informed search strategies**
  - Heuristics functions
- Search in complex environments
  - Hill climbing, simulated annealing, local beam search, evolutionary algorithm.