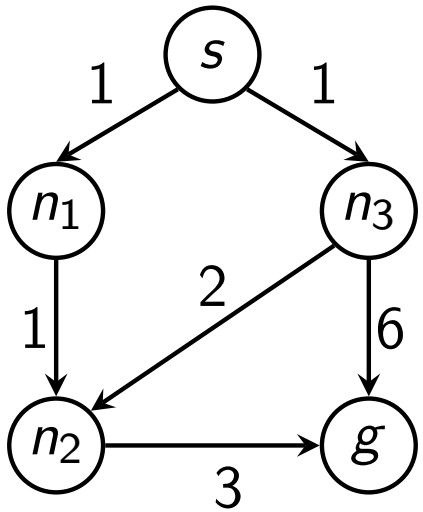
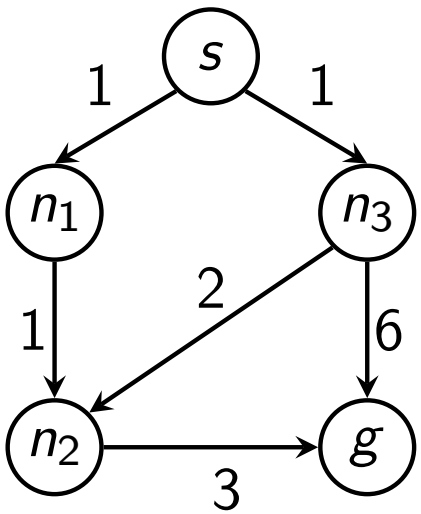
COMP4475 Graph Searching: Towards the A* Algorithm

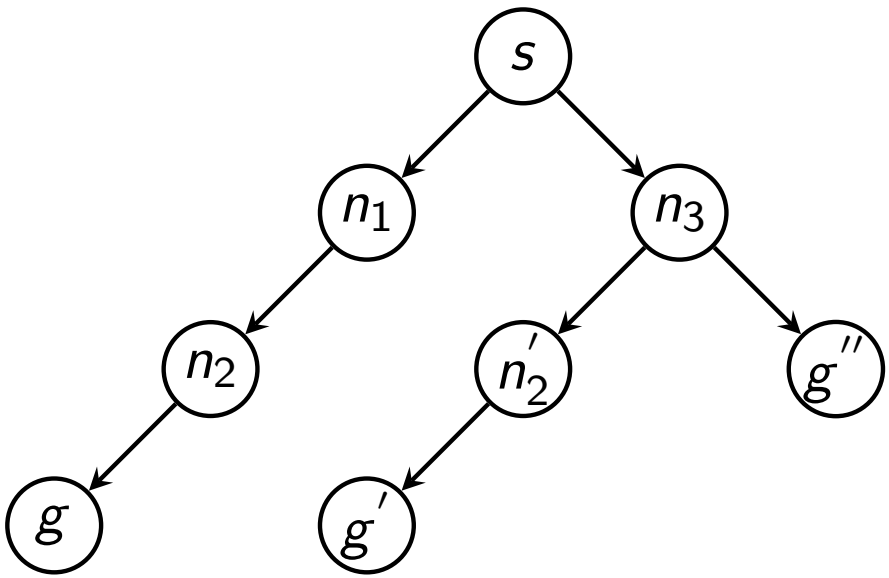
Winter 2023

An Example Graph



An Example Graph (and Its Search Tree)





Graphs: Definitions

- A **Directed Graph:** set of nodes N, set of directed edges $E \subseteq N \times N$
- If $\langle n_1, n_2 \rangle \in E$, then n_2 is a neighbor of n_1
  - written $n_1 \rightarrow n_2$, an ordered pair, not symmetric
  - corresponds to a possible move from state n_1 to state n_2
  - may be labeled to denote its properties (e.g., a move cost)
- A **Path:** a sequence of nodes $\langle n_1, n_2, \ldots, n_k \rangle$ such that each $\langle n_i, n_{(i+1)} \rangle \in E$

Search Tree

- given a search graph G and a start node s we can construct a search tree
- the search tree is a tree, such that
  - node s is its root node, and
  - each node has all its neighbors for children
- assume that the root node is at level 0
- at level k of the tree are all nodes that one can reach in exactly k moves in G
  - each path in a search graph is a path through search tree
  - nodes in a search graph can appear many times in its search tree

Graph Search Problems

- A **Graph Search Problem:** given graph (N, E), a start node $s \in N$, and a set of goal nodes $G \subseteq N$, find a path P from s to some $g \in G$ satisfying certain properties
  - any path to any g in G, or
  - best (according to some optimization criterion) path among all paths in G

Generic Search Procedure

- A **frontier of paths** F, a set of nodes we know how to reach from the start node s
  - all nodes behind it are already explored
  - all nodes beyond it are unexplored
- The procedure
  - Initially $F = \{s\}$
  - loop until frontier is empty
    - choose some node n from F; remove n from F
    - if $n \in G$ then stop; report success
    - otherwise add each neighbor (could be none) of n to F
- A **Search Strategy** defines
  - which element of F is selected at each stage, and
  - how new nodes are added to F

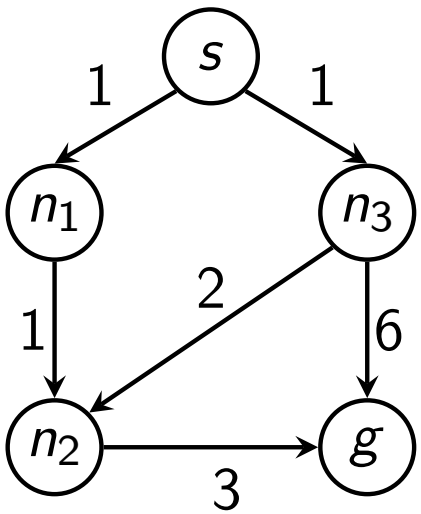
Search Strategies

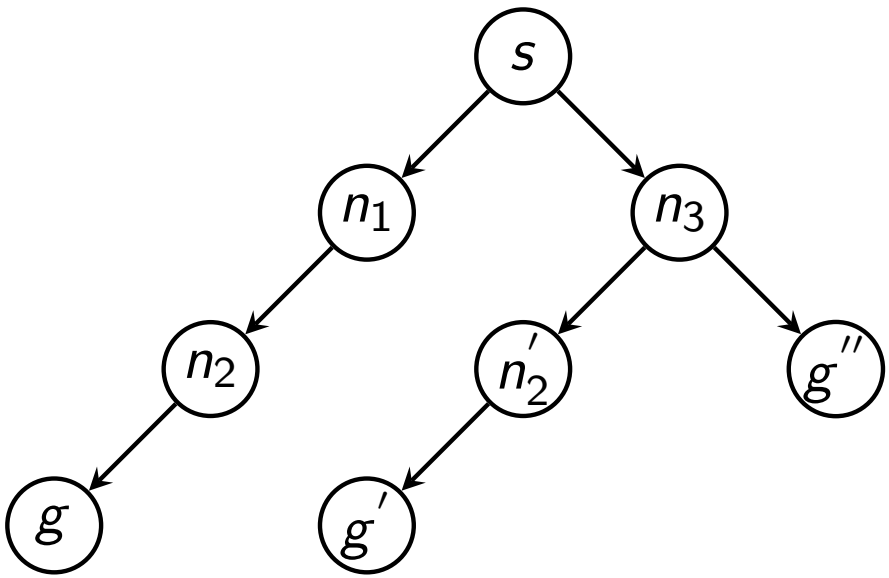
- Depth-first Search (DFS)
- Breadth-first Search(BrFS)
- **Least-cost-first Search (LCFS)**
- Best-first Search(BeFS)
- A* Search

Least-cost-first Search (LCFS)

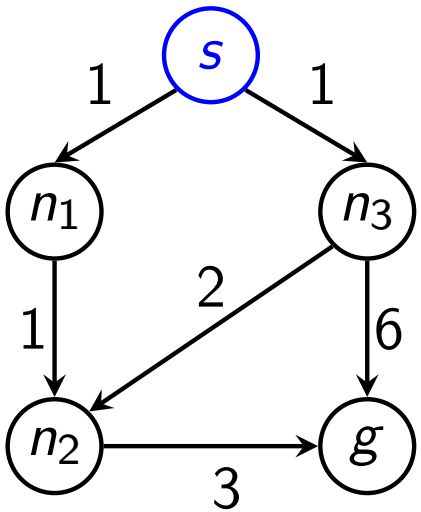
- each edge in the graph is associated with a non-negative cost; the cost of any path p from s to node n is denoted by $g(n)$
- explore paths in the order of path cost (thus ensure least-cost solutions)
- the frontier is maintained by a priority queue, where paths/nodes are in the order of cost
- always select the node from the front (i.e., with minimal cost) to expand
- always insert the neighbors to the frontier in the order of cost

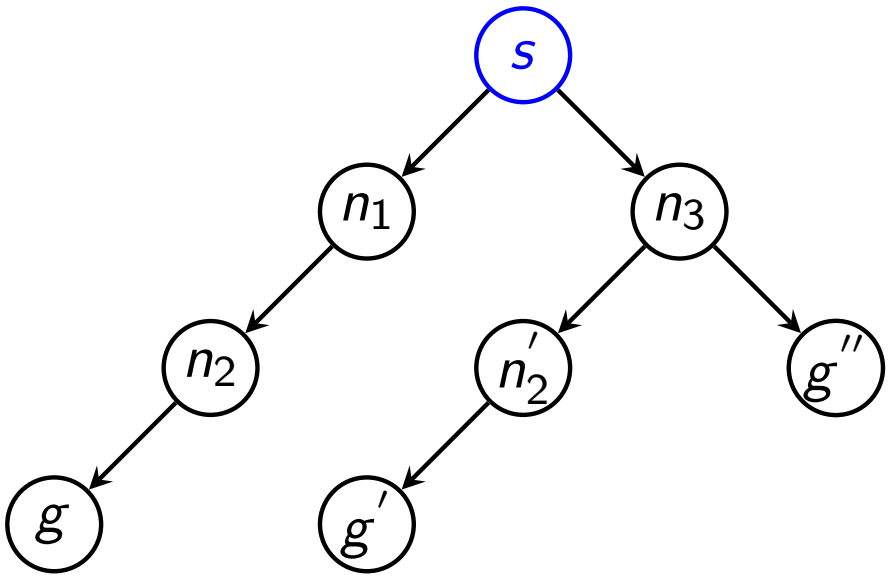
LCFS, with $g(n)$





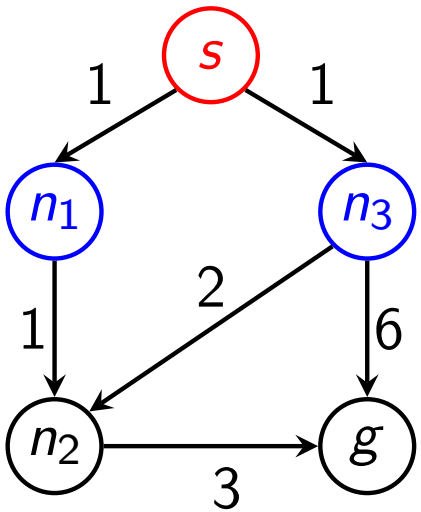
LCFS, with $g(n)$

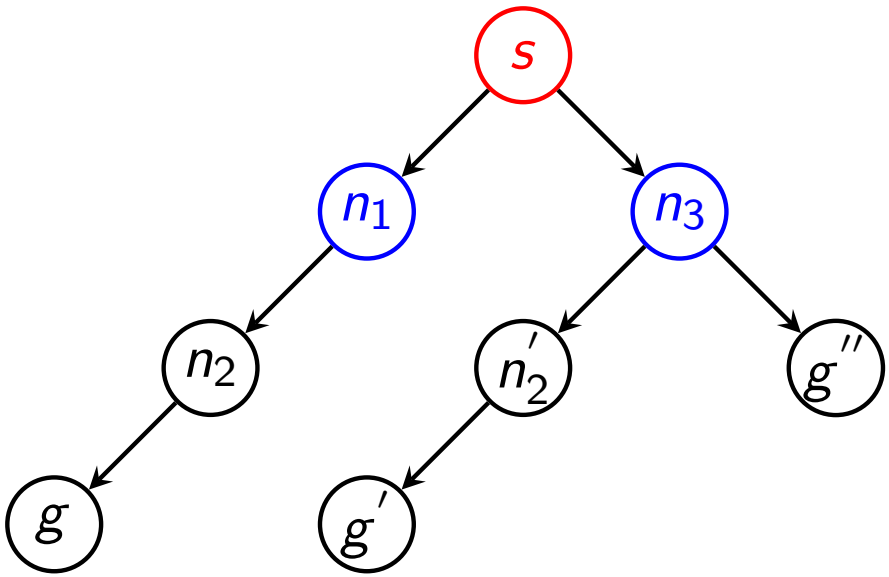




PQ: s(0)

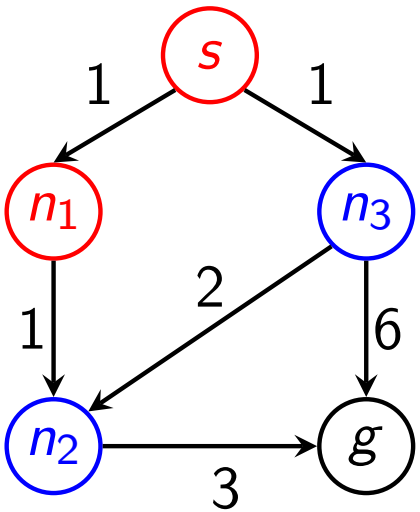
LCFS, with $g(n)$

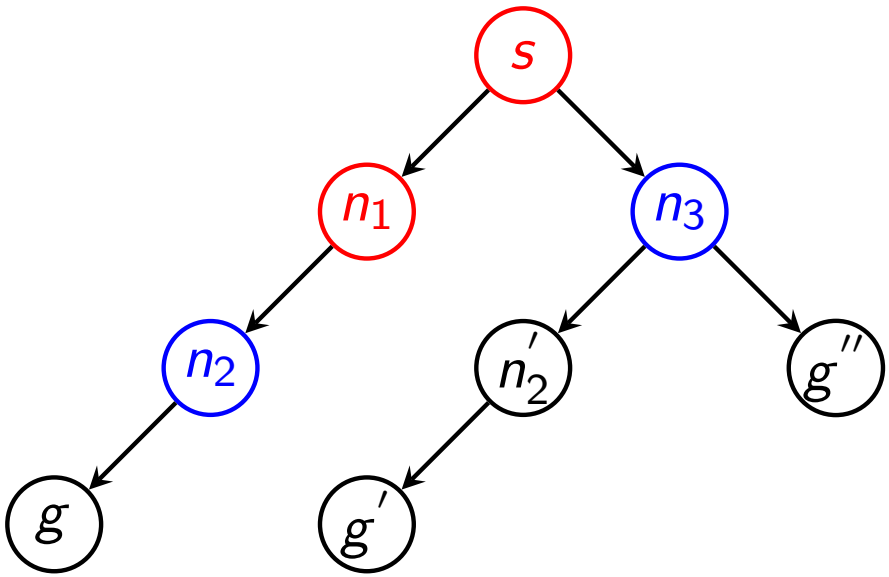




PQ: $n_1(1)$, $n_3(1)$

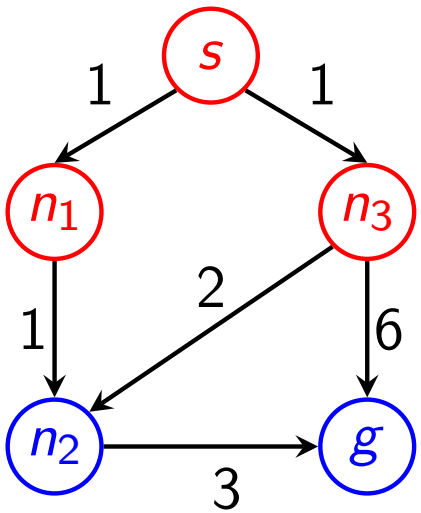
LCFS, with $g(n)$

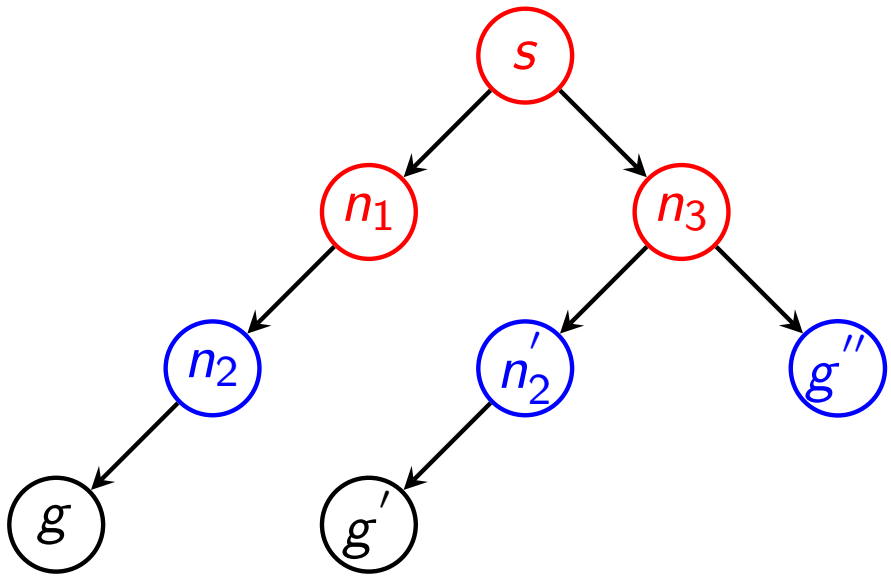




PQ: $n_3(1)$, $n_2(2)$

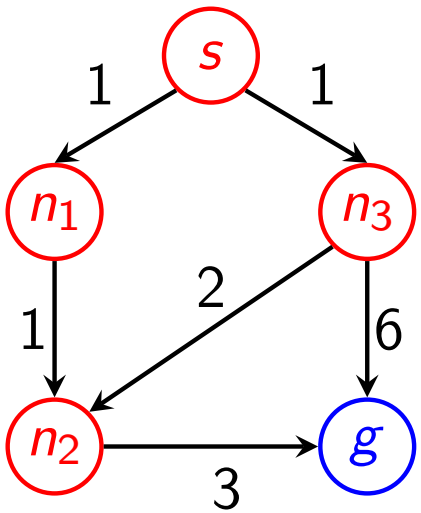
LCFS, with $g(n)$

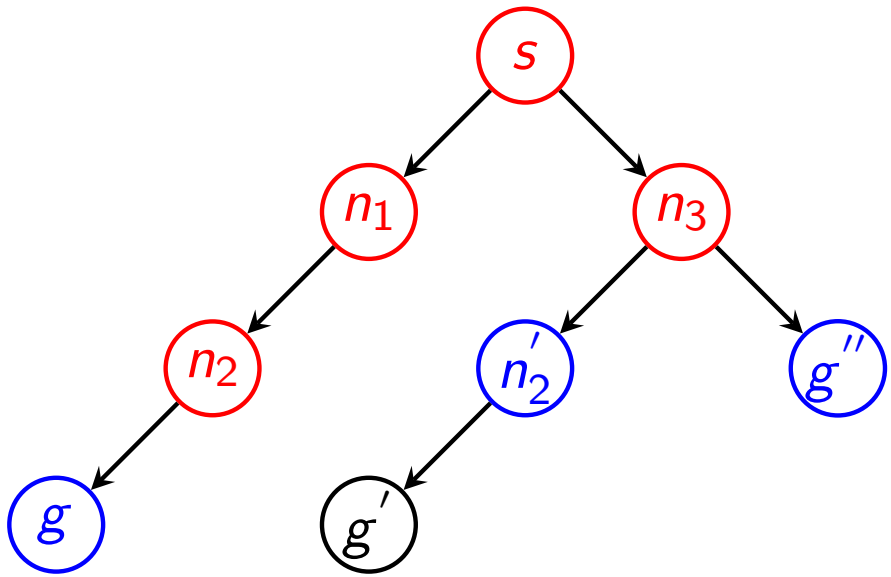




PQ: $n_2(2)$, $n_2'(3)$, $g''(7)$

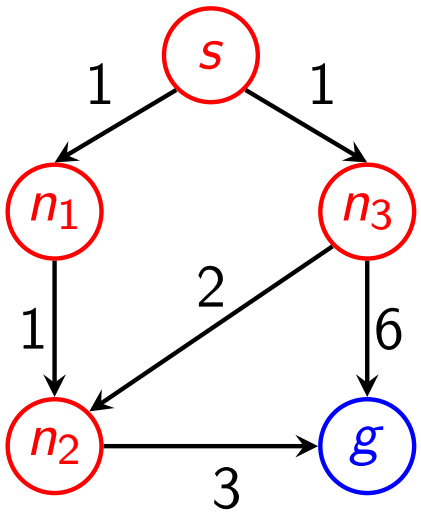
LCFS, with $g(n)$

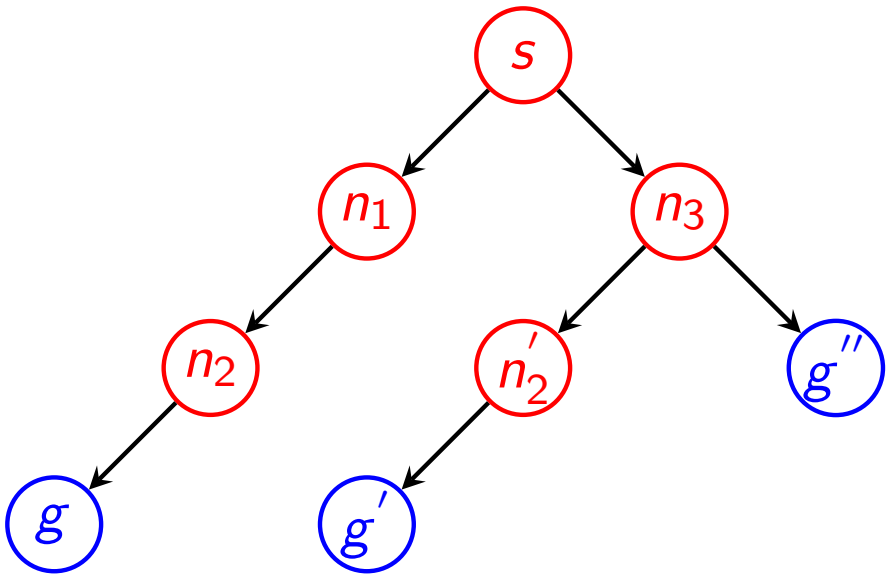




PQ: $n_2'(3)$, $g(5)$, $g''(7)$

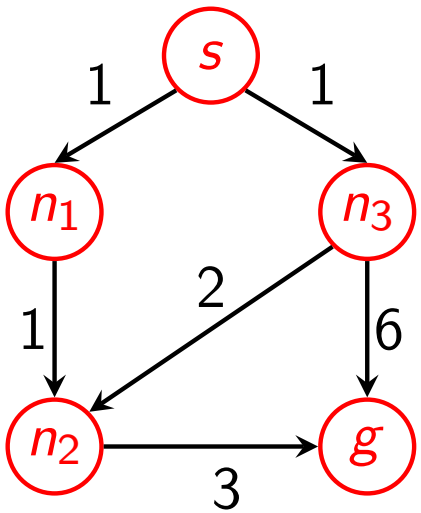
LCFS, with $g(n)$

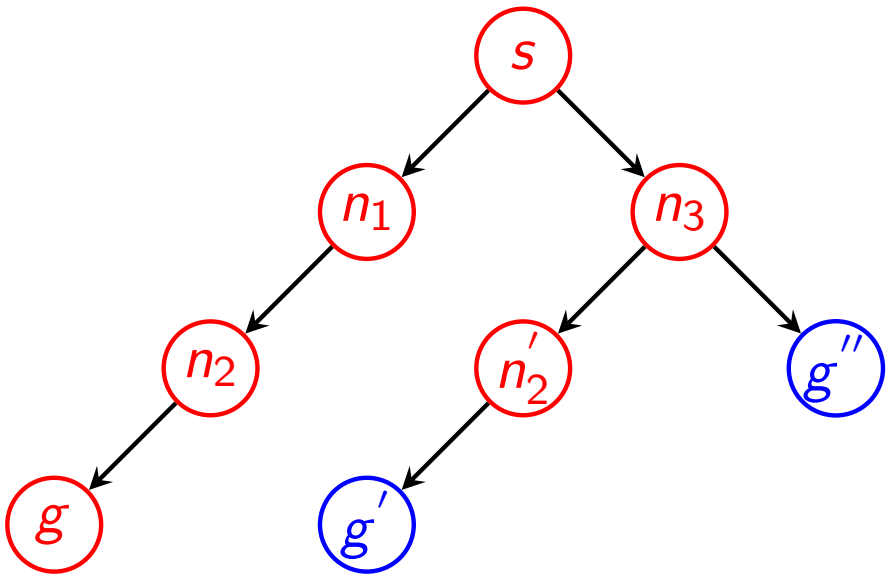




PQ: $g(5)$, $g'(6)$, $g''(7)$

LCFS, with $g(n)$





PQ: $g^{'}(6)$, $g^{''}(7)$

Solution found: $s \rightarrow n_1 \rightarrow n_2 \rightarrow g$

cost of the path is $1 + 1 + 3 = 5$

Properties of LCFS

- guaranteed to find least-cost path if all edge costs are greater than zero and a goal node exists
- why? If there were a better path than the first solution found, it would have been selected from the frontier eariler
- (an uninformed search strategy) LCFS is not influenced by the goal: whatever the goal is, the paths LCFS decide to explore will be the same, up to the point that the goal is found; They do not use any information about where they are trying to go until they happen to encounter a goal

Search Strategies

- Depth-first Search (DFS)
- Breadth-first Search(BrFS)
- Least-cost-first Search (LCFS)
- **Best-first Search(BeFS)**
- A* Search

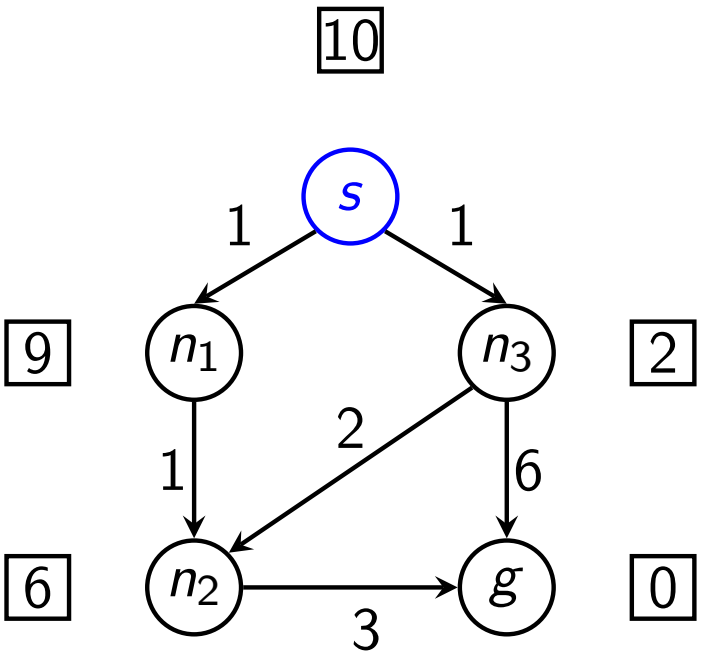
Heuristics

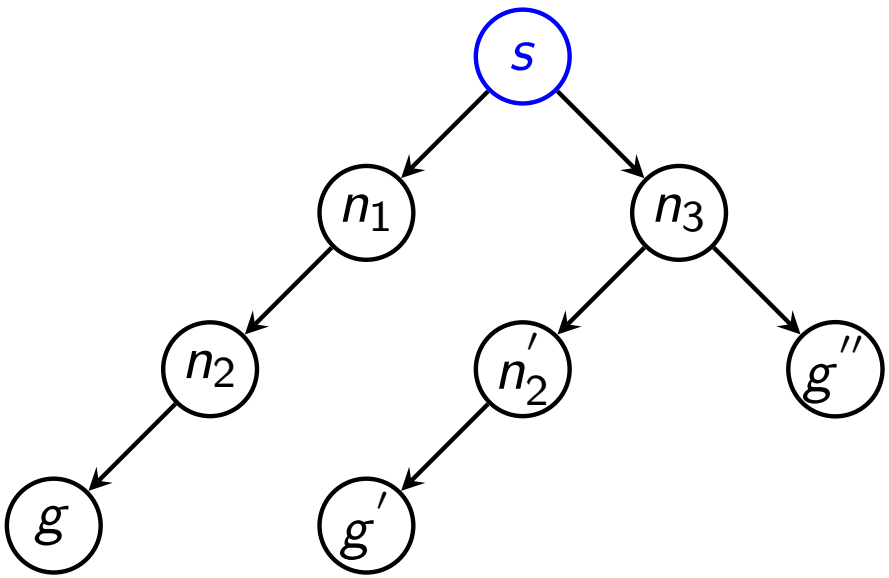
- **Heuristics** refer to any rules of thumb that provide some help with solving a problem
- in graph search, a **heuristic function** $h(n)$ is an estimate of cost from node n to the goal g
- where heuristics come from depends on the problem we are trying to solve
- a good heuristic function should be
  - somewhat accurate
  - easy to compute
  - underestimate true cost

Best-first Search (BeFS)

- explore paths in order of heuristics $h(n)$ instead of $g(n)$ (in LCFS)
- the frontier is maintained by a priority queue in the order of heuristic values
- ignore edge costs together: not influenced by $g(n)$, the cost of path so far to node n

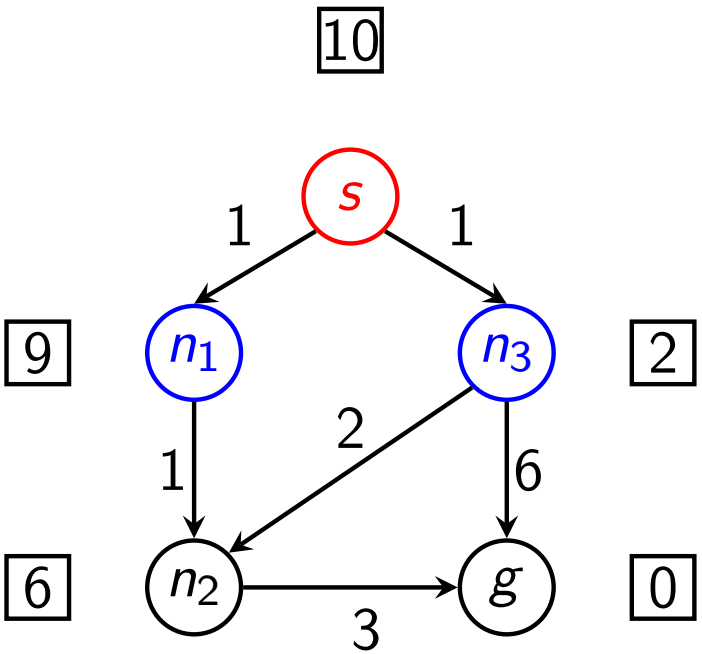
BeFS, with $h(n)$

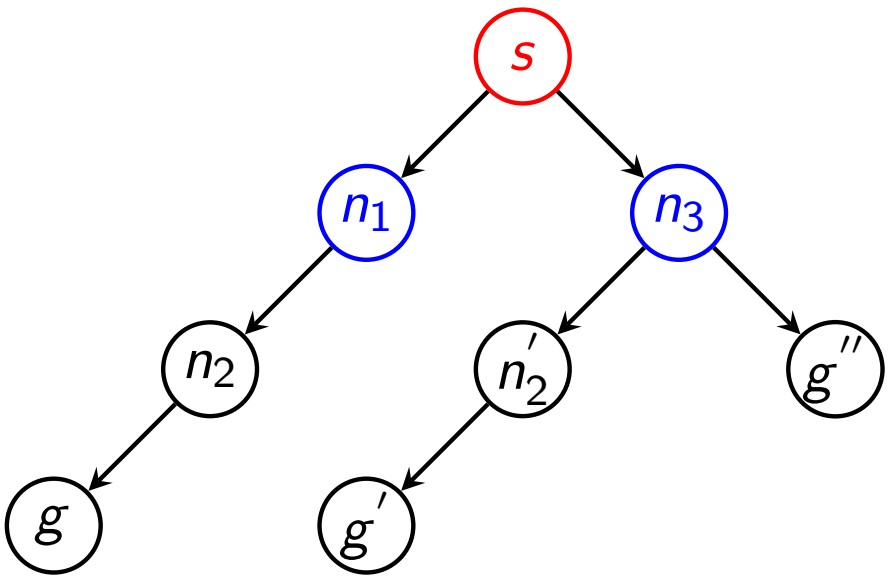




PQ: $s(10)$

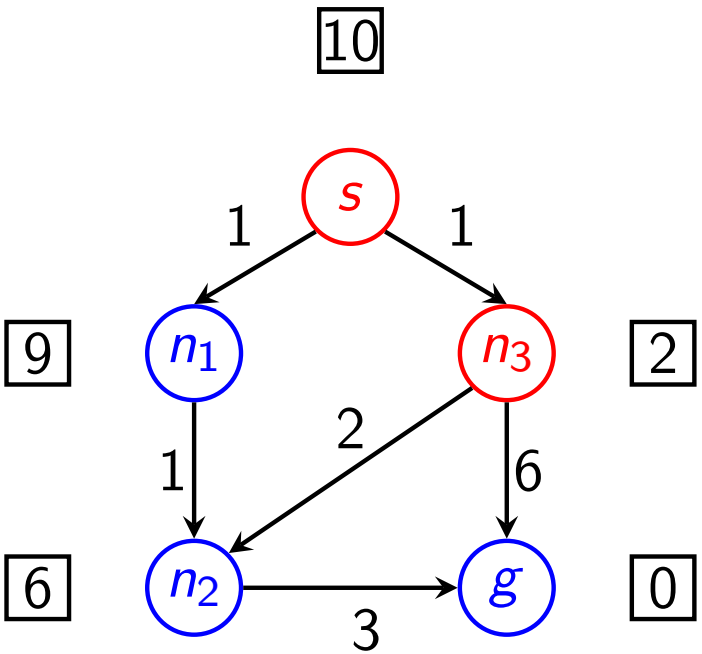
BeFS, with $h(n)$

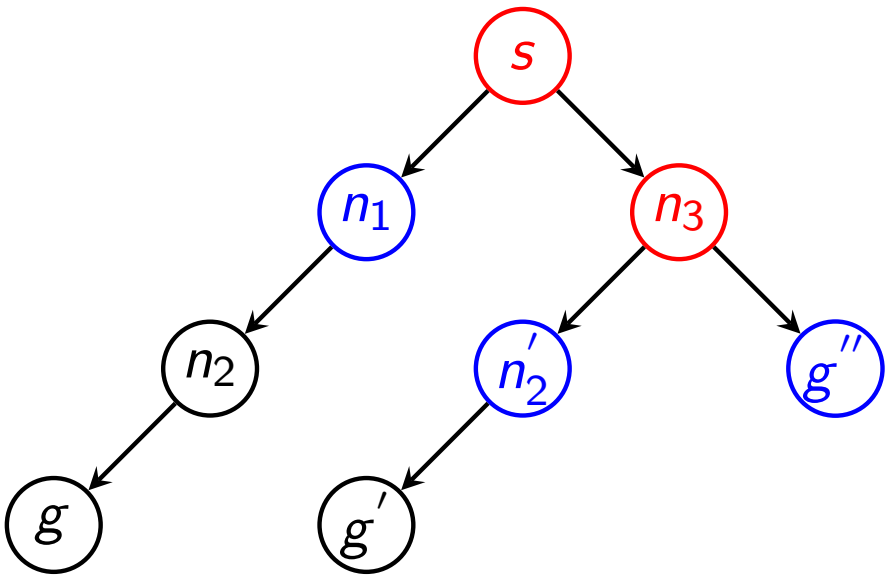




PQ: $n_3(2)$, $n_1(9)$

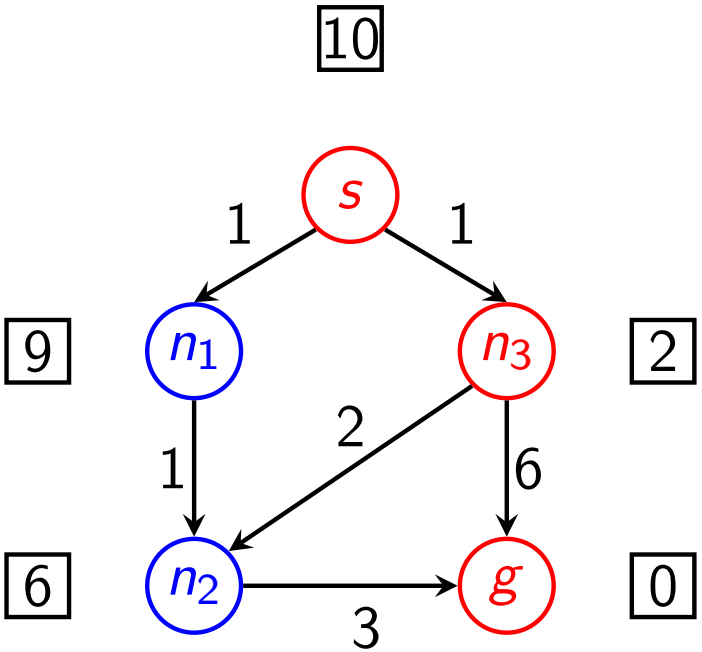
BeFS, with $h(n)$

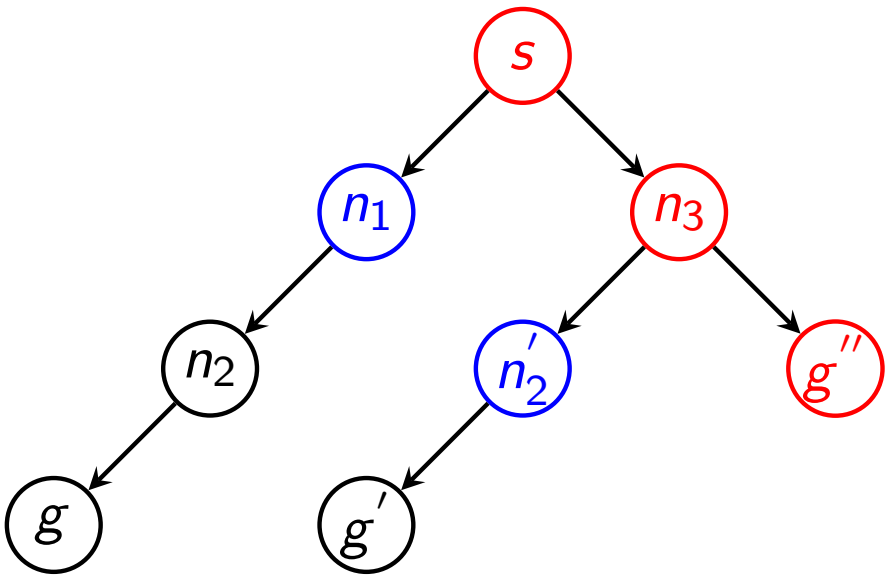




PQ: $g''(0)$, $n_2'(6)$, $n_1(9)$

BeFS, with $h(n)$





PQ: $n_2'(6)$, $n_1(9)$

Solution found: $s \rightarrow n_3 \rightarrow g''$

cost of the path is $1 + 6 = 7$, actually non-optimal

Search Strategies

- Depth-first Search (DFS)
- Breadth-first Search(BrFS)
- Least-cost-first Search (LCFS)
- Best-first Search(BeFS)
- **A* Search**

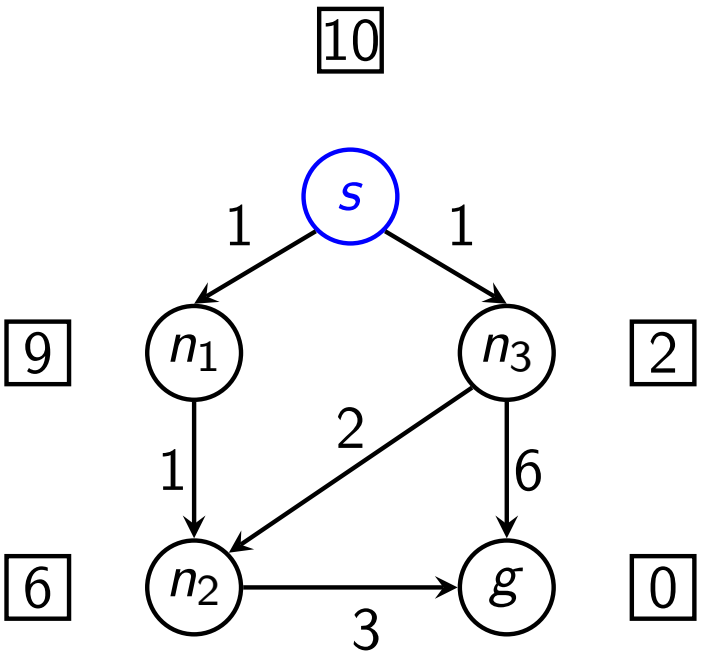
A* Search

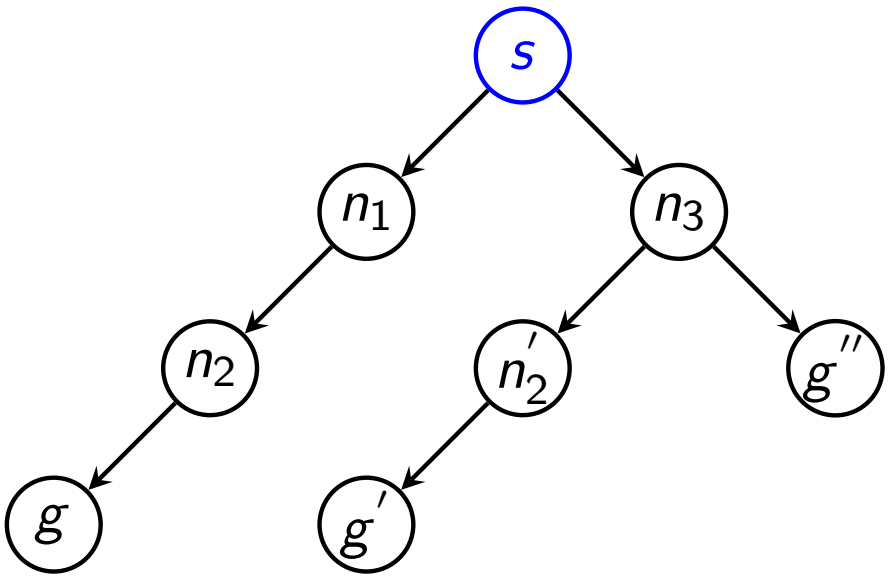
- A* search combines aspects of LCFS and BeFS, by evaluating the quality of path on frontier through the evaluation function

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

- the frontier is maintained by a priority queue according to f-value $f(n)$
- always select the node from the front (i.e., minimal f-value)
- always insert the neighbors to the frontier according to $f(n)$

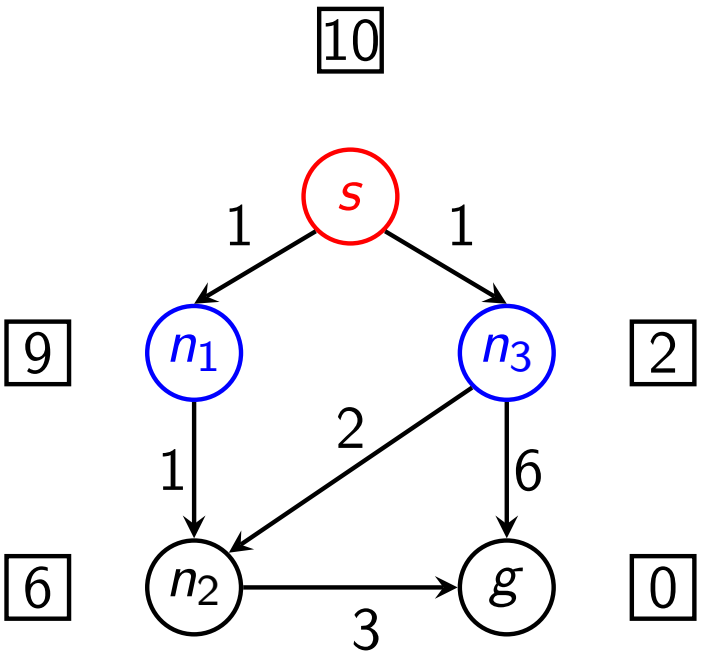
A* Search, with $f(n) = g(n) + h(n)$

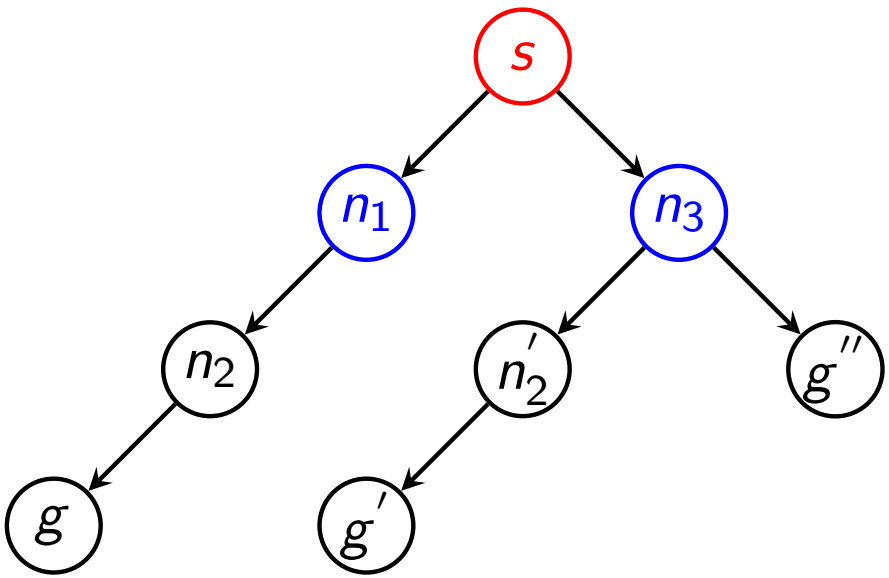




PQ: $s(10 = 0 + 10)$

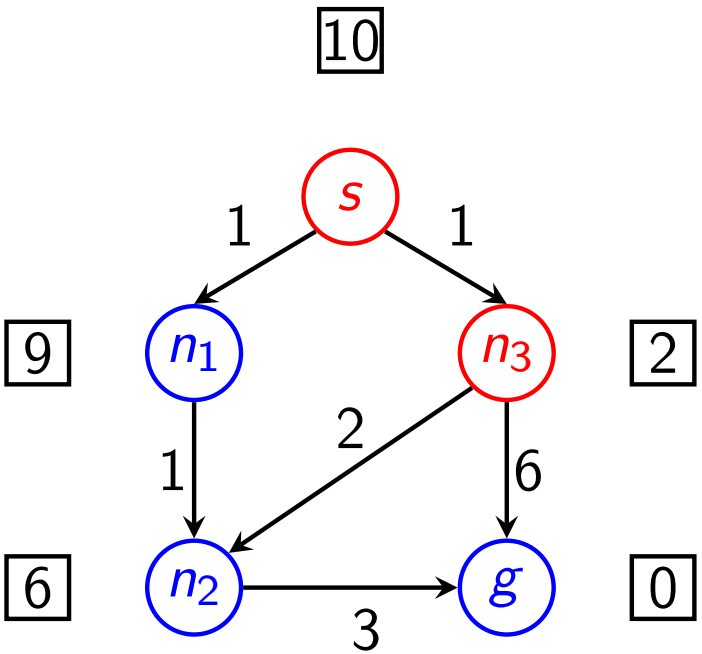
A* Search, with $f(n) = g(n) + h(n)$

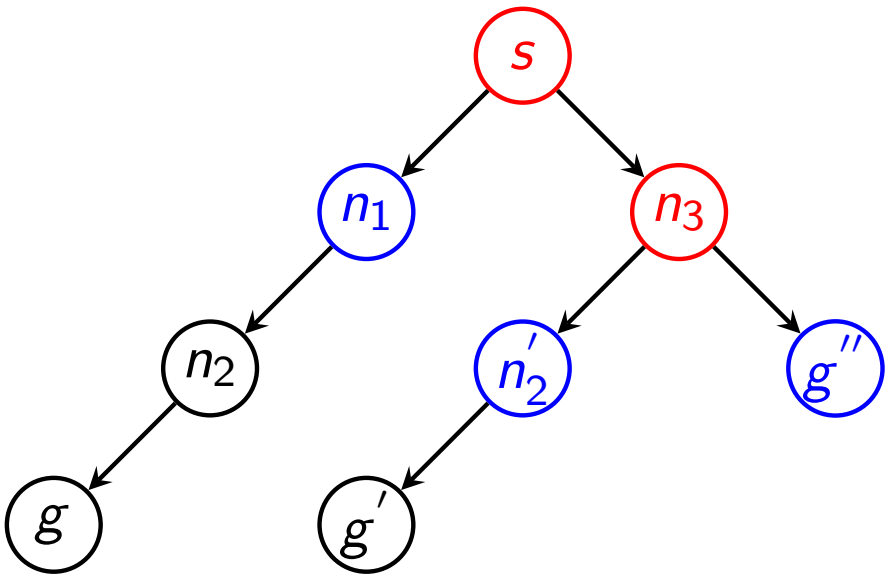




PQ: $n_3(3 = 1 + 2)$, $n_1(10 = 1 + 9)$

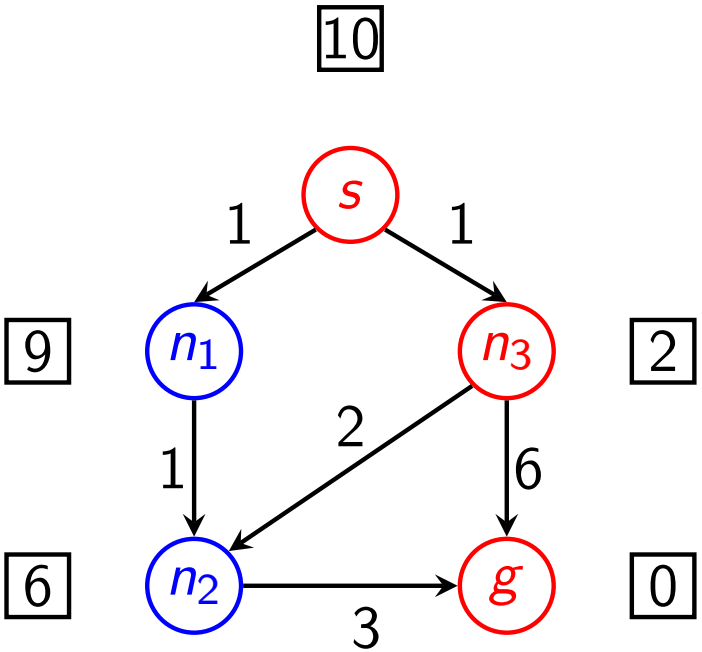
A* Search, with $f(n) = g(n) + h(n)$

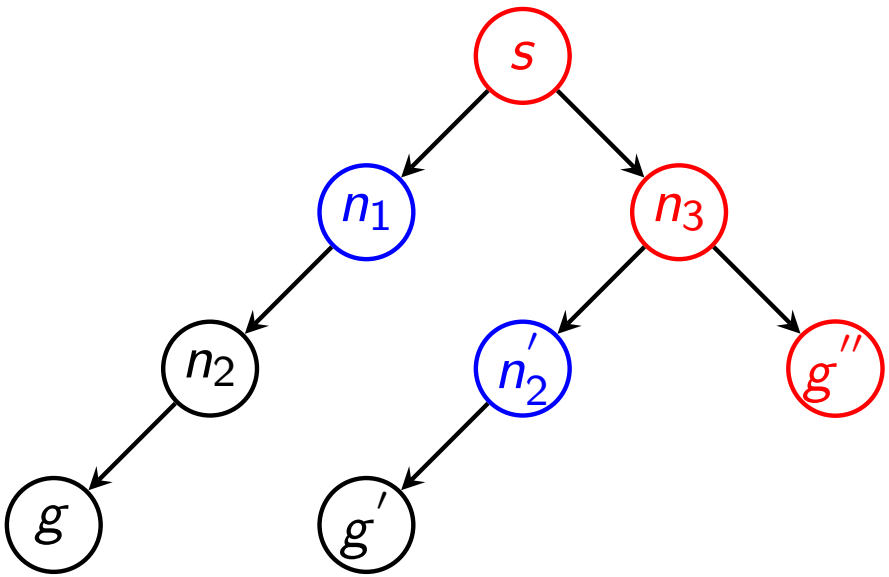




PQ: $g''(7 = 1 + 6 + 0)$, $n_2'(9 = 1 + 2 + 6)$, $n_1(10 = 1 + 9)$

A* Search, with $f(n) = g(n) + h(n)$





PQ: $n_2'(9 = 1 + 2 + 6)$, $n_1(10 = 1 + 9)$

Solution found: $s \rightarrow n_3 \rightarrow g''$

cost of the path is $1 + 6 = 7$, **Again Non-optimal**

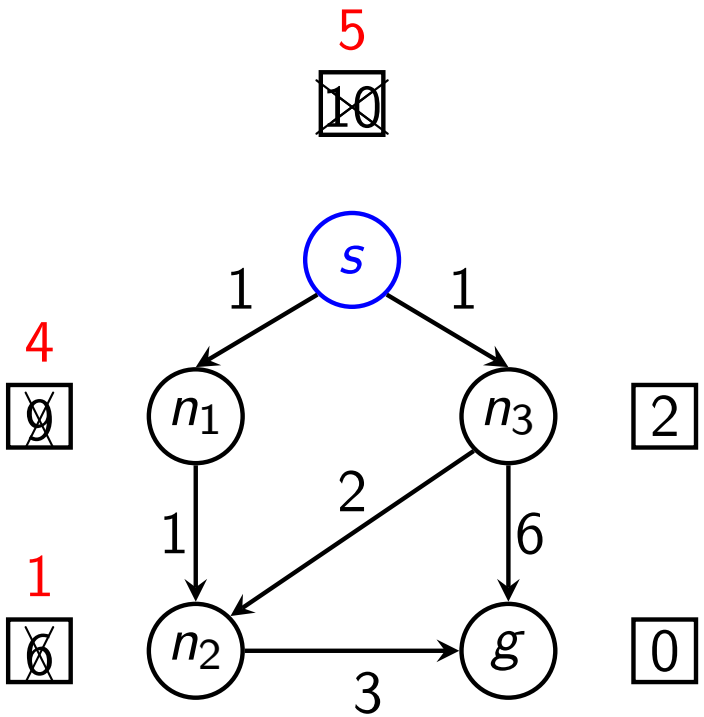
Search Strategies

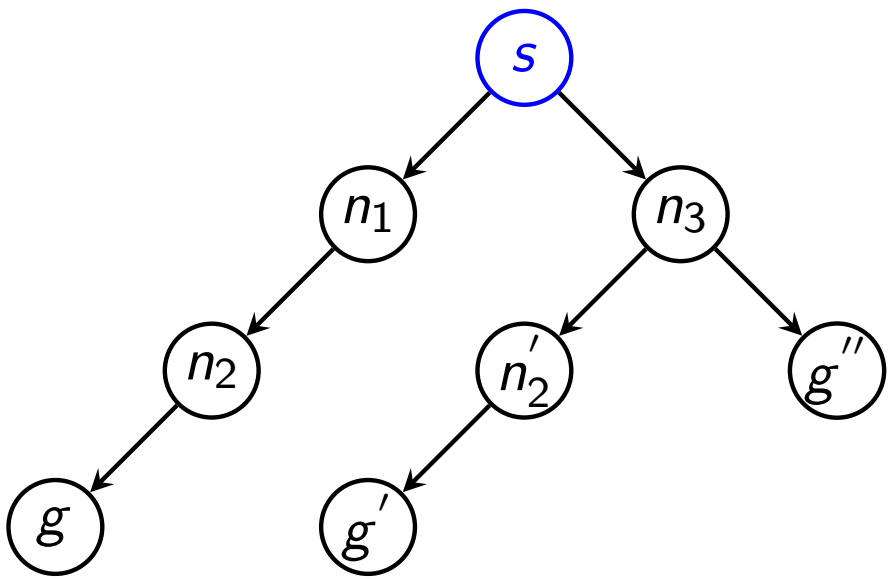
- Depth-first Search (DFS)
- Breadth-first Search(BrFS)
- Least-cost-first Search (LCFS)
- Best-first Search(BeFS)
- **A* Search**
  - Admissible Heuristics (Optimality of A*)
  - Monotone Restriction (enabling Multiple Path Checking)

Admissible Heuristics

- A* does not always find least-cost path (i.e., is A* admissible?)
- if $h(n) = 0$ for all n then A* is reduced to LCFS
- if $h(n)$ is perfect (i.e., equals to the actual minimum cost of the shortest path from n to the goal node), then A* will find optimal path directly
- In general, the more "informative" $h(n)$ is, the better A* will perform
- A* is admissible, if $h(n)$ is a lower bound on the actual minimum cost of the shortest path from n to the goal node

A* Search, with $f(n) = g(n) + h(n)$, and $h(n)$ is admissible!





PQ: $s(5)$

Multiple Path Checking in A*

- MPC: If you find a path to node n that you have already expanded, prune that path.
- MPC works for LCFS, since shortest on n is assured; In A*, you can be misled by heuristic that takes you all the way to node n along "expensive path"
- but this can happen only if

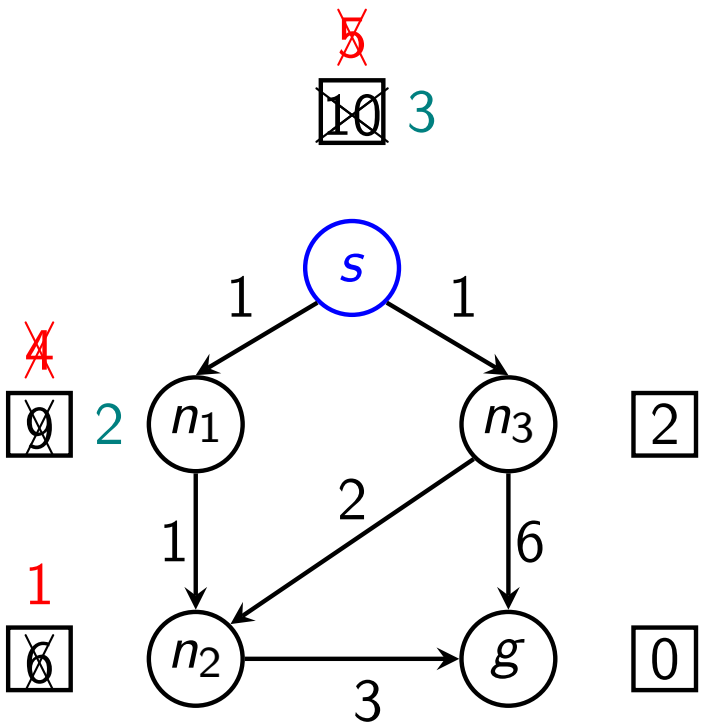
$$d(m,n) < |h(m) - h(n)|$$

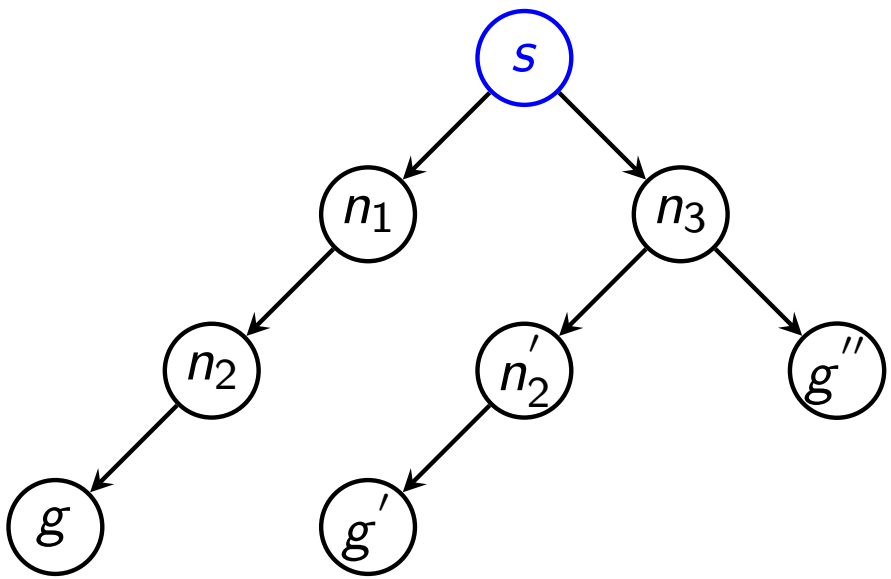
  for some node n and m
- hence, having the monotone restriction:

$$d(m,n) \geq |h(m) - h(n)|$$

  for any two nodes m and n ensures that MPC can be performed safely with A*

A* Search, with $f(n) = g(n) + h(n)$, and $h(n)$ is admissible, and MPC is applicable!





PQ: $s(3)$