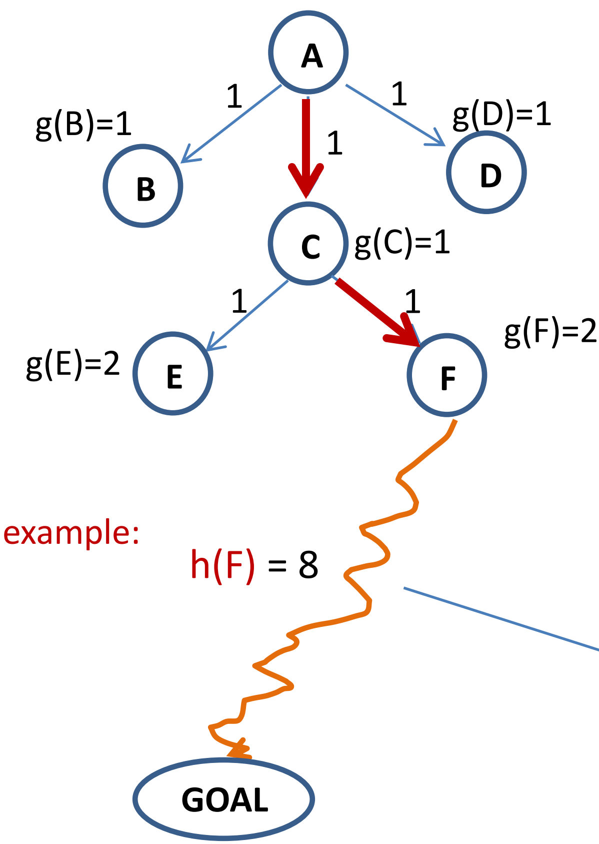
Heuristic

Informed (heuristic) search: one that uses problem-specific knowledge to guide search.

Heuristic function: *A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood*



g(n): cost function (the cost of a path from A to n)

g(F) = 2 because all operators have cost = 1

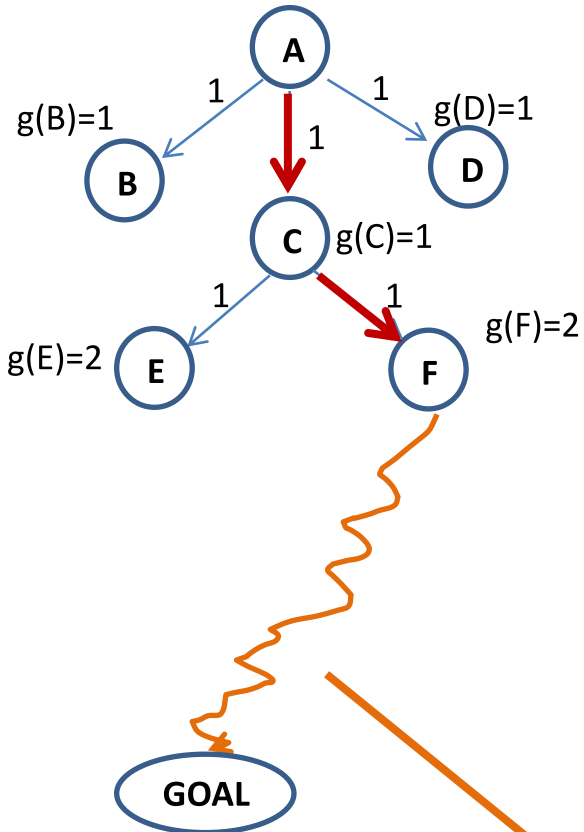
h(n): heuristic funcion → function that **estimates** the cost of a node **n** to a GOAL node.

For example:

h(F) = 8

It means there is an estimation that the cost of the optimal path from **F** to **GOAL** is 8
(8 moves or operators because all operators have the same cost = 1)

Heuristic



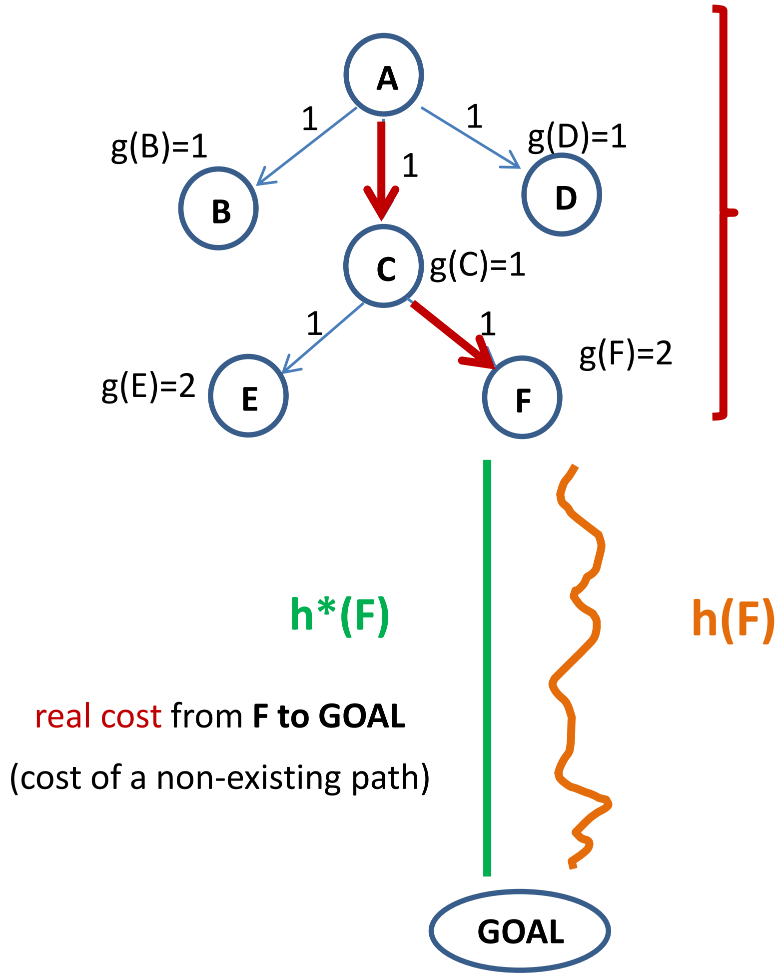
g(F) = 2 Real cost to go from A to F

h(F) = 8 Estimated cost to go from F to the goal state (approximation)

How can we come up with a cost estimate if this part of the tree is not generated yet?

Exploiting problem-specific knowledge

In summary …



g(F) = 2

Real cost to go from **A** to **F**

(cost of an existing path)

Estimated cost from **F** to **GOAL**
(approximation)

In summary

It always holds $\forall n \; h(n) <= h^*(n)$

h(n) is called an admissible heuristic

Reminder

we know that if it always holds $\forall n \; h(n) <= h^*(n)$

h(n) is called an admissible heuristic

---

f(n) = g(n)+h(n) → **A algorithm**

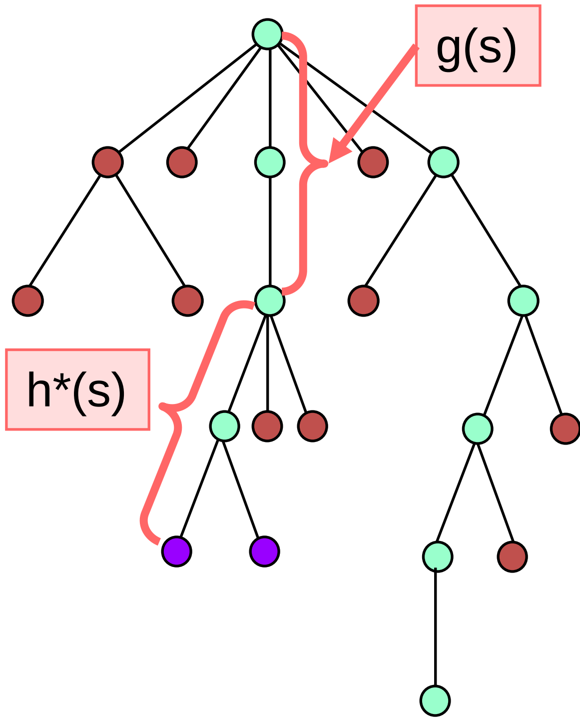
if h(n) is admissible → **A* algorithm**

↓

guarantees OPTIMAL SOLUTION

Node-Selection Heuristic

- Suppose we're searching a **tree** in which each edge (s,s') has a cost $c(s,s')$
  - If p is a path, let $c(p)$ = sum of the edge costs
  - For classical planning, this is the length of p



- For every state s, let
  - $g(s)$ = cost of the path from s_0 to s
  - $h^*(s)$ = least cost of all paths from s to goal nodes
  - $f^*(s) = g(s) + h^*(s)$ = least cost of all paths from s_0 to goal nodes that go through s

- Suppose $h(s)$ is an estimate of $h^*(s)$
  - Let $f(s) = g(s) + h(s)$
    - $f(s)$ is an estimate of $f^*(s)$
  - h is *admissible* if for every state s, $0 \leq h(s) \leq h^*(s)$
  - If h is admissible then f is a lower bound on f^*

The A* Algorithm

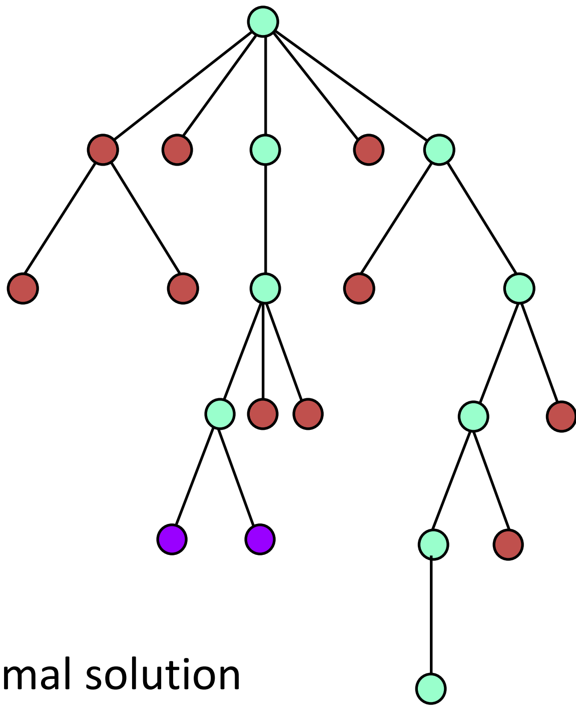
- A* on trees:

  loop

  choose the leaf node *s* such that *f*(*s*) is smallest

  if *s* is a solution then return it and exit

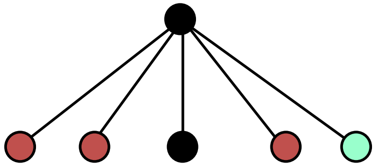
  expand it (generate its children)



- On graphs, A* is more complicated
  - additional machinery to deal with multiple paths to the same node

- If a solution exists (and certain other conditions are satisfied), then:
  - If *h*(*s*) is admissible, then A* is guaranteed to find an optimal solution
  - The more "informed" the heuristic is (i.e., the closer it is to *h**), the smaller the number of nodes A* expands
  - If *h*(*s*) is within *c* of being admissible, then A* is guaranteed to find a solution that's within *c* of optimal

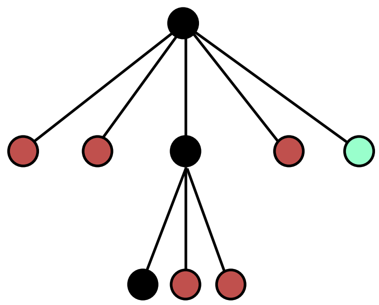
The FastForward Planner



- Use a heuristic function similar to $h(s) = \Delta^g(s,g)$
  - Some ways to improve it (I'll skip the details)
- Don't want an A*-style search (takes too much memory)
- Instead, use a greedy procedure:

until we have a solution, do
  expand the current state *s*
  s := the child of *s* for which *h(s)* is smallest
    (i.e., the child we think is closest to a solution)

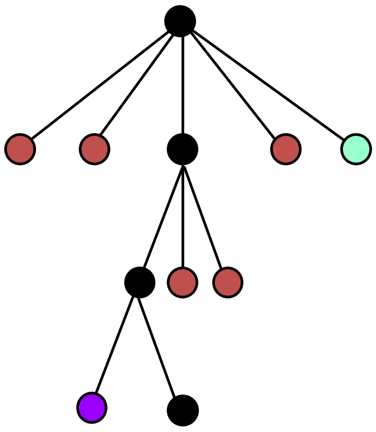
The FastForward Planner



- Use a heuristic function *h*(*s*) = relaxed plan
- Don't want an A*-style search (takes too much memory)
- Instead, use a greedy procedure:

until we have a solution, do
    expand the current state *s*
    s := the child of *s* for which *h(s)* is smallest
        (i.e., the child we think is closest to a solution)

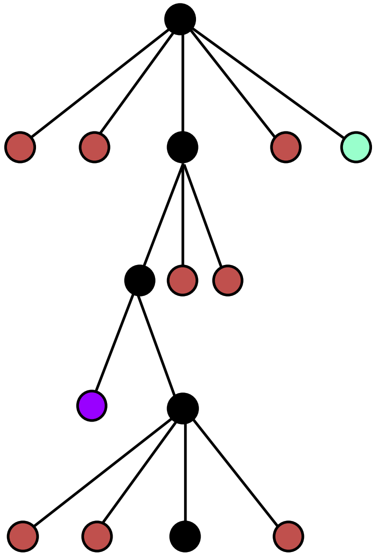
The FastForward Planner



- Use a heuristic function $h(s)$ = relaxed plan
- Don't want an A*-style search (takes too much memory)
- Instead, use a greedy procedure:

until we have a solution, do
    expand the current state *s*
    s := the child of *s* for which *h(s)* is smallest
        (i.e., the child we think is closest to a solution)

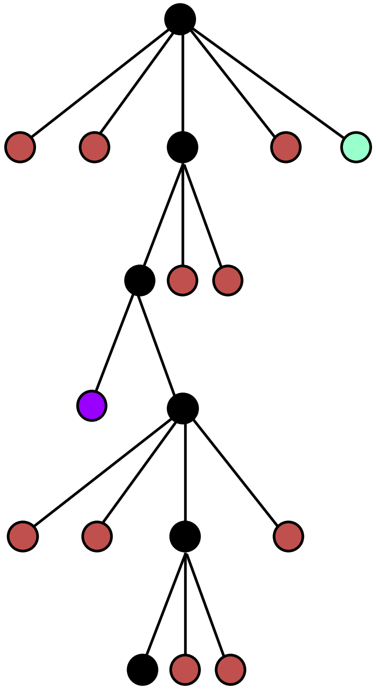
The FastForward Planner



- Use a heuristic function $h(s)$ = relaxed plan
- Don't want an A*-style search (takes too much memory)
- Instead, use a greedy procedure:

until we have a solution, do
    expand the current state *s*
    s := the child of *s* for which *h(s)* is smallest
        (i.e., the child we think is closest to a solution)

The FastForward Planner



- Use a heuristic function similar to *h*(*s*) = relaxed plan
- Don't want an A*-style search (takes too much memory)
- Instead, use a greedy procedure:

  until we have a solution, do
      expand the current state *s*
      s := the child of *s* for which *h(s)* is smallest
        (i.e., the child we think is closest to a solution)

- There are some ways FF improves on this
  - e.g. a way to escape from local minima
    - breadth-first search, stopping when a node with lower cost is found
- Can't guarantee how fast it will find a solution, or how good a solution it will find
  - However, it works pretty well on many problems