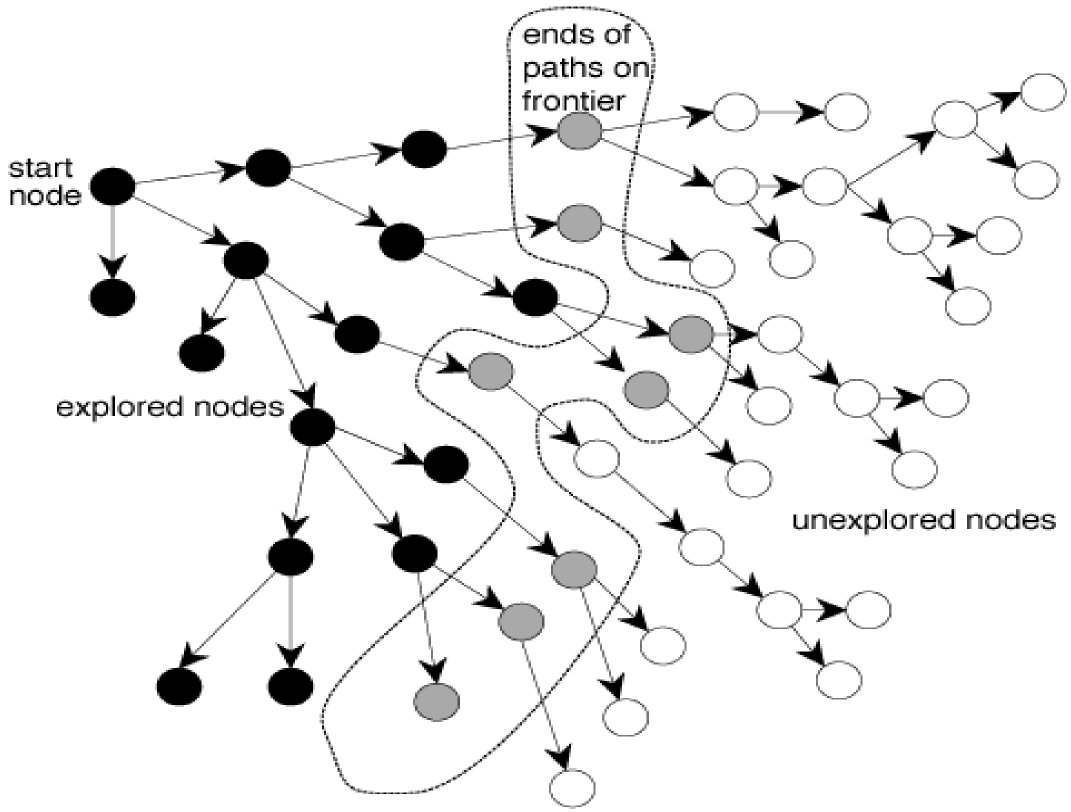
A* Graph Search

Last modified on 2022/04/1 by f.maire@qut.edu.au

Based on material by Stuart Russel, Peter Norvig, Dan Klein and Pieter Abbeel

Informed Search – A* graph search



Credits: most material borrowed from AIMA or Dan Klein and Pieter Abbeel

*Last modified on **2022/04/1** by f.maire@qut.edu.au*

Reading for this week

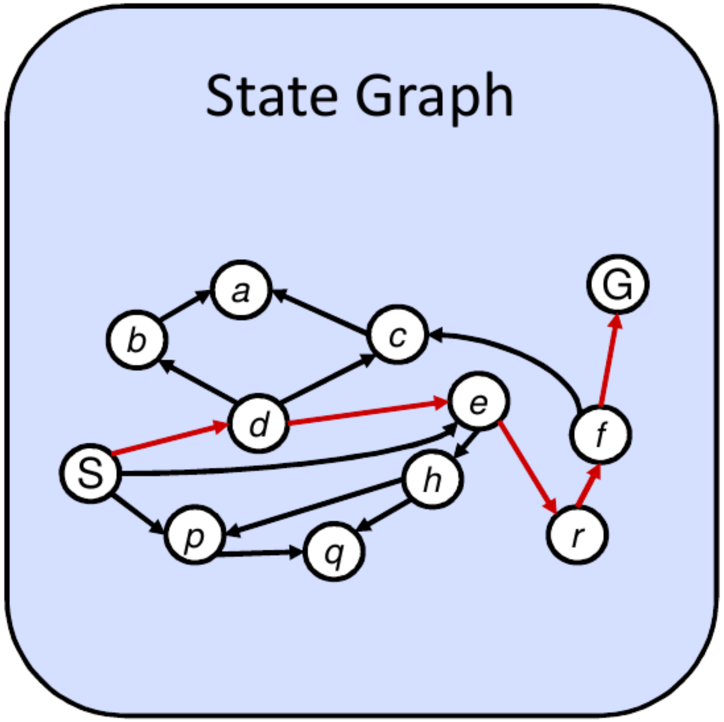
(same as last week)

- Chapter 3, Sections 3.5 to 3.6 of AIMA
  - Russell and Norvig Textbook:
    Artificial Intelligence, a Modern Approach
    3rd edition

Today's Menu

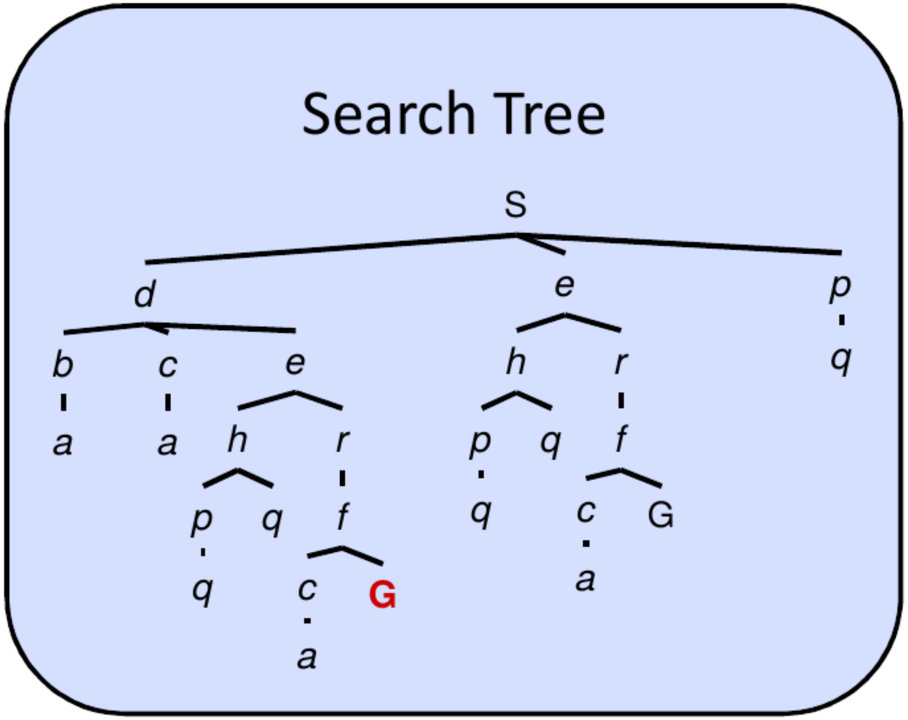
- Recap
- Informed Search Methods
  - A* Graph Search

Recap: State Graphs vs. Search Trees



Each NODE in in the search tree is an entire PATH in the problem graph.

We construct both on demand – and we construct as little as possible.

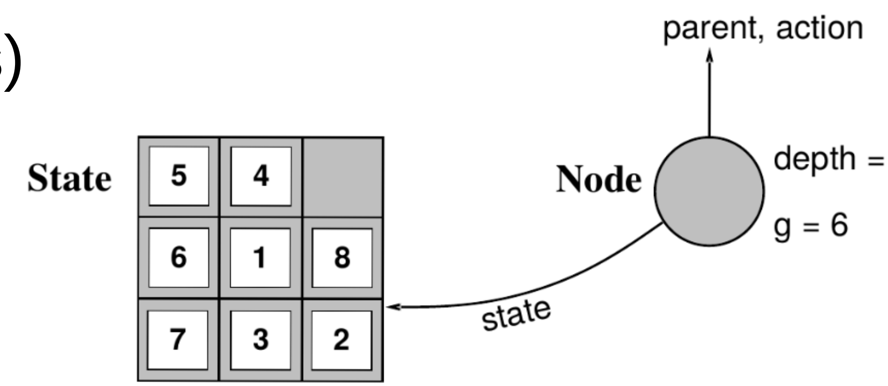


Don’t confuse these two!

Recap: sequence of actions search

- Search problem
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test
- Search tree
  - Nodes:
    represent plans for reaching states
  - Plans have costs
    (sum of action costs)
- Search algorithm
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds cheapest plans (aka sequence of actions)

Recap from Week 02



Fringe and *Frontier*
are synonyms

Graph search (Fig 3.7 in AIMA)

- We augment the tree search algorithm with a set *explored* , which remembers every state that has been goal tested negatively.

```
function Graph-Search(problem) returns a solution, or failure
    frontier ← problem.initial_state
    explored ← empty set    # initial empty set of explored states
    while frontier not empty:
        node ← frontier.pop()
        if problem.goal_test(node.state):
            return node
        explored.add(node.state)
        frontier.extend(child for child in node.expand(problem)
                        if child.state not in explored and child not in frontier)
    return None
```

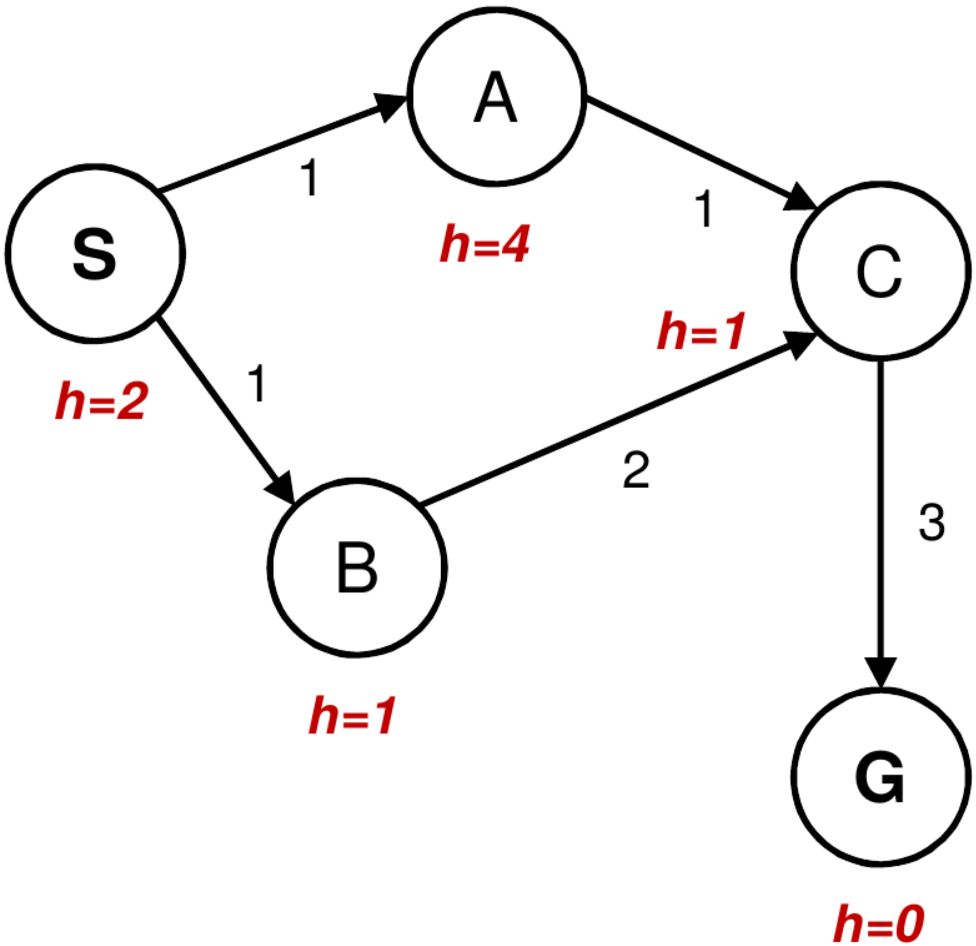
Recall from Week 3
Graph Search

```python
def best_first_graph_search(problem, f):
    """
    Search the nodes with the lowest f scores first.
    You specify the function f(node) that you want to minimize; for example,
    if f is a heuristic estimate to the goal, then we have greedy best
    first search; if f is node.depth then we have breadth-first search.
    """
    node = Node(problem.initial)
    if problem.goal_test(node.state):
        return node
    frontier = PriorityQueue(f=f)
    frontier.append(node)
    explored = set() # set of states
    while frontier:
        node = frontier.pop()
        if problem.goal_test(node.state):
            return node
        explored.add(node.state)
        for child in node.expand(problem):
            if child.state not in explored and child not in frontier:
                frontier.append(child)
            elif child in frontier:
                # frontier[child] is the f value of the
                # incumbent node that shares the same state as
                # the node child.  Read implementation of PriorityQueue
                if f(child) < frontier[child]:
                    del frontier[child] # delete the incumbent node
                    frontier.append(child) #
    return None
```

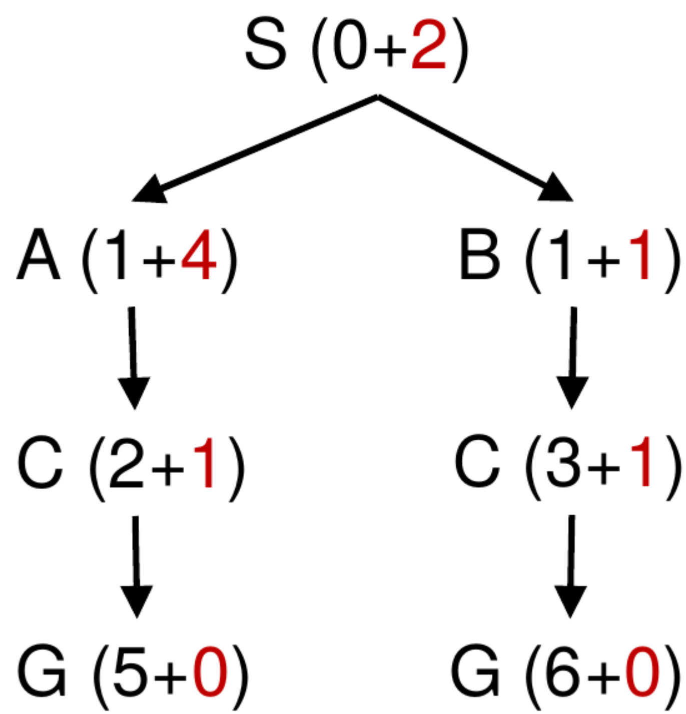
A* graph search
is ‘best first graph search’
with f=g+h

A* Graph Search Gone Wrong?

State space graph



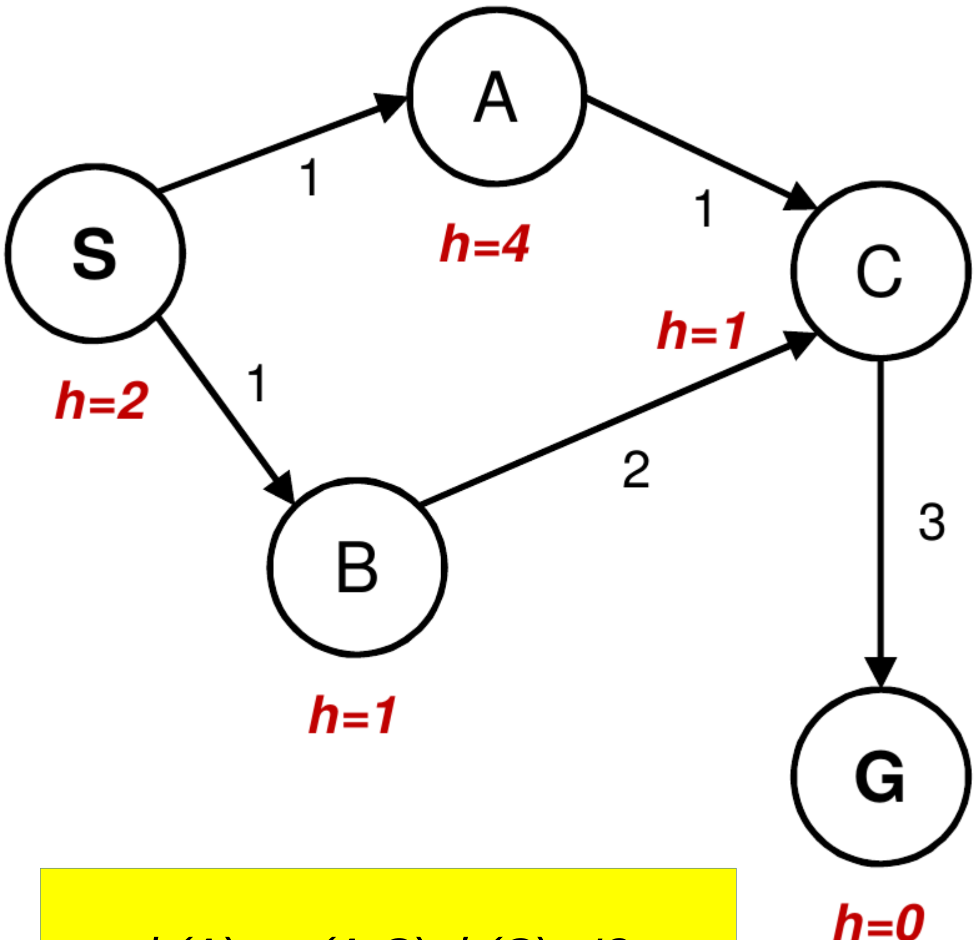
Search tree



When B(1+1) is expanded
C(3+1) is added to the frontier.
This prevents the addition of C(2+1) to
the frontier when A(1+4) when is expanded.

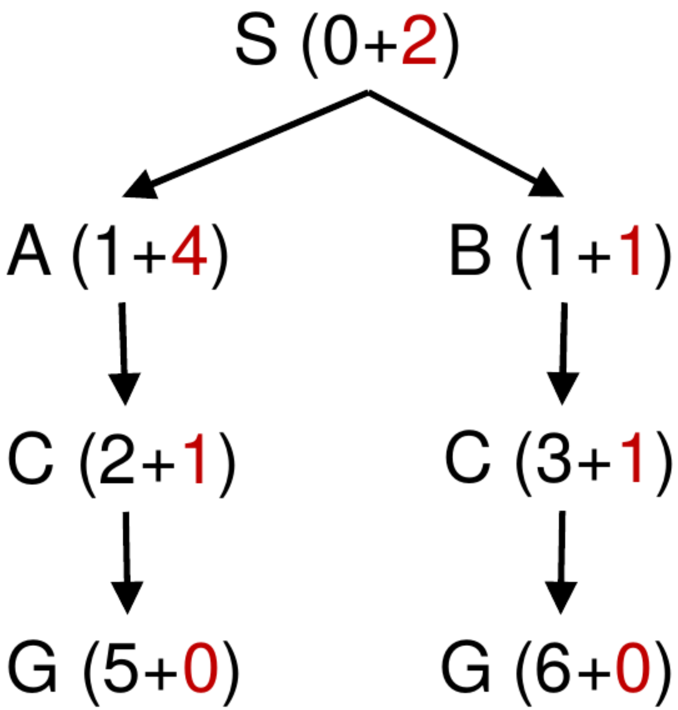
A* Graph Search Gone Wrong?

State space graph

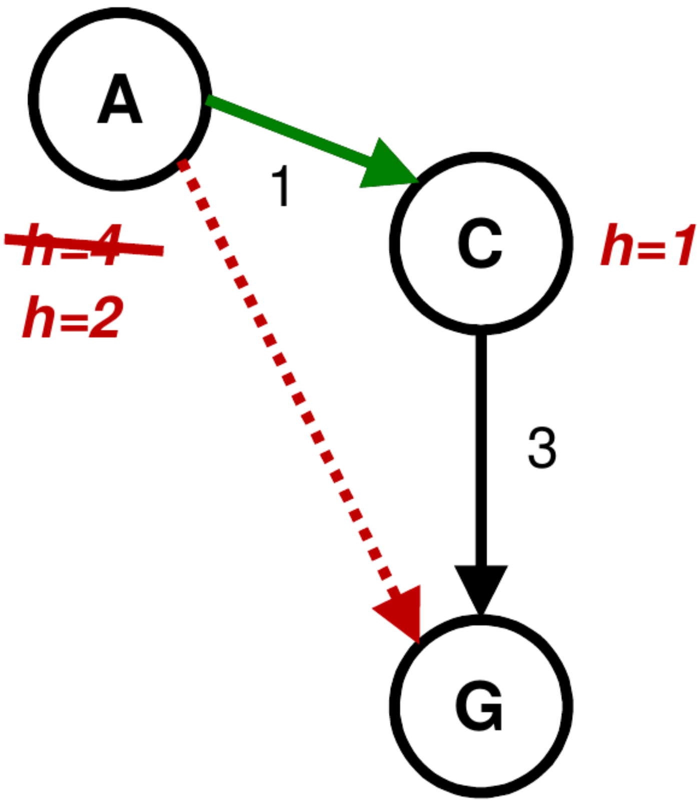


$h(A) > c(A,C)+h(C)$!?

Search tree



Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G

  - Consistency: heuristic cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

- Consequences of consistency:
  - The f value along a path never decreases

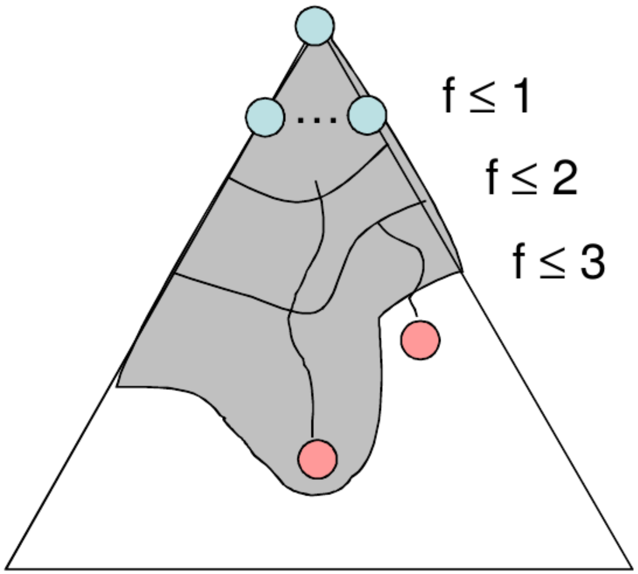
    h(A) ≤ cost(A to C) + h(C)

  - A* graph search is optimal

Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:

- Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
- Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
- Result: A* graph search is optimal



Optimality of A* Graph Search

(more details)

Let's show that

$$\boxed{\text{On a path } n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_k \quad \text{We have } f(n_1) \leq f(n_2) \leq \ldots \leq f(n_k)}$$

As *h* is consistent,

$$\underline{h(n_{i-1})} \leq \underline{c(s_{i-1}, s_i) + h(s_i)}$$

Therefore, we have

$$f(n_{i-1}) = g(n_{i-1}) + \underline{h(n_{i-1})} \leq g(n_{i-1}) + \underline{c(s_{i-1}, s_i) + h(s_i)}$$

By construction,

$$g(n_i) = g(n_{i-1}) + c(s_{i-1}, s_i)$$

We conclude that

$$f(n_{i-1}) \leq g(n_{i-1}) + c(s_{i-1}, s_i) + h(s_i) = g(n_i) + h(s_i) = f(n_i)$$

Optimality of A* Graph Search

(more details)

Recall that on a path $n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_k$ We have $f(n_1) \leq f(n_2) \leq \ldots \leq f(n_k)$

Loop invariant

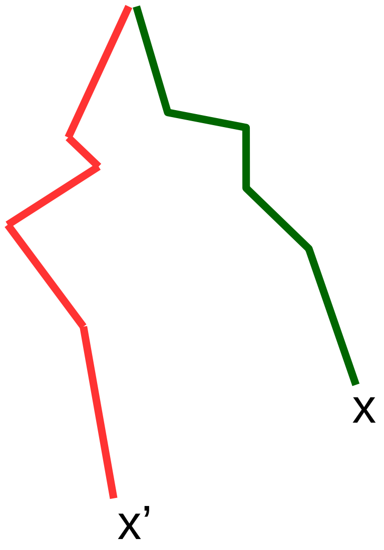
The node **x** removed from the frontier is such that **g(x)** is the cheapest cost from the **initial state** to $\mathbf{s_x}$ where $\mathbf{s_x}$ is the state associated to **x**

Proof by contradiction,
let assume that there is a cheaper path

$f(x') = \mathbf{g}(x')+h(s_{x'}) \;<\; \mathbf{g}(x)+h(s_x) = f(x)$

Because $s_{x'} = s_x$ we have $h(s_{x'}) = h(s_x)$

Therefore x' will be removed from the frontier before x



Optimality Summary

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems