A* Optimality

CS4804

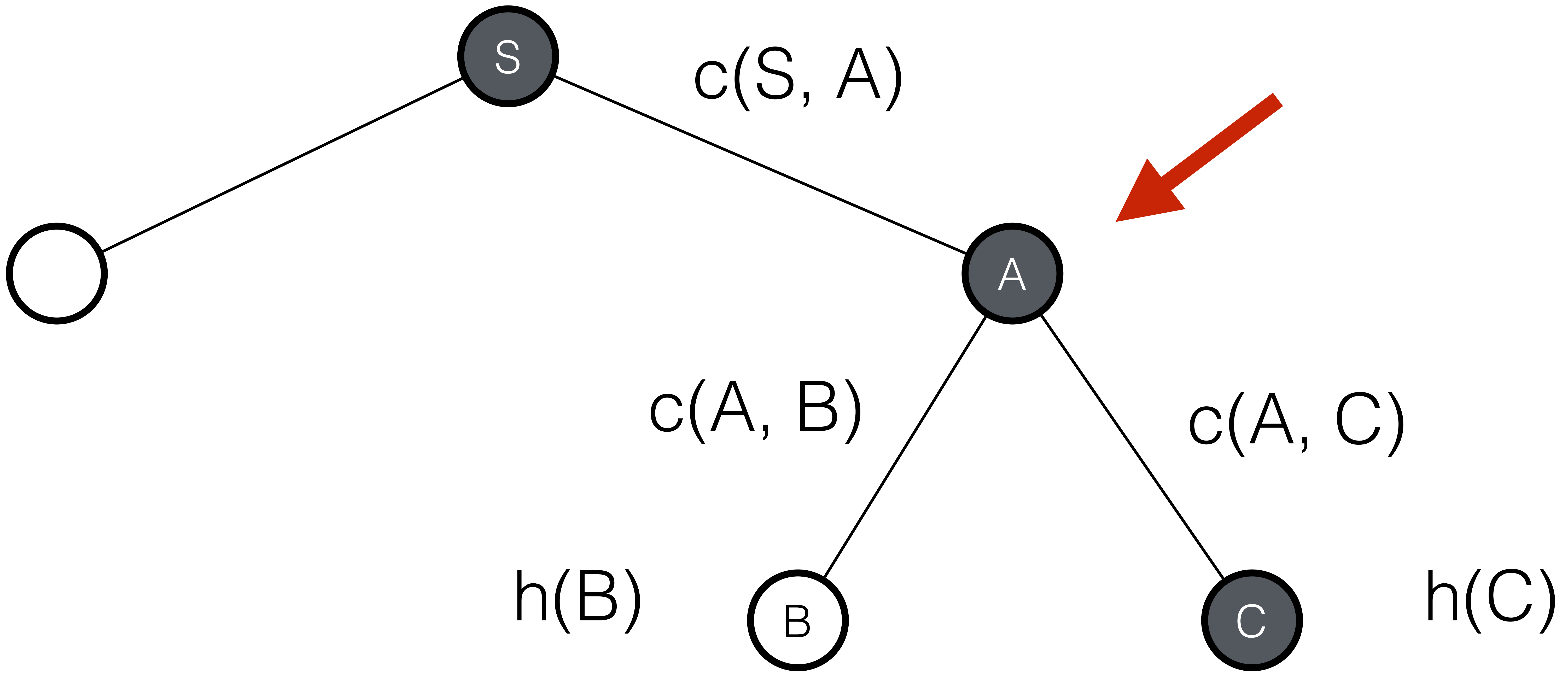
Plan

- A* in a tree
- A* in a graph
  - How to handle multiple paths to a node
  - Intuition about **consistency**
- Search space + heuristic design practice

A* Search

- Expand node in frontier with best evaluation function score **f(n)**
  - **f(n) = g(n) + h(n)**
  - **g(n) :=** cost to get from initial state to **n**
  - **h(n) :=** heuristic estimate of cost to get from **n** to goal
- Optimal in trees if **admissible** **h(n)** <= true cost to goal
- Optimal in graphs if **consistent** **h(n)** <= **c(n, n')** + **h(n')**

A* in a Tree



B: c(S, A) + c(A, B) + h(B)

C: c(S, A) +c(A, C) + h(C)

g(n) + **h(n)**

g(n) is always the exact cost of the **only** path to **n**

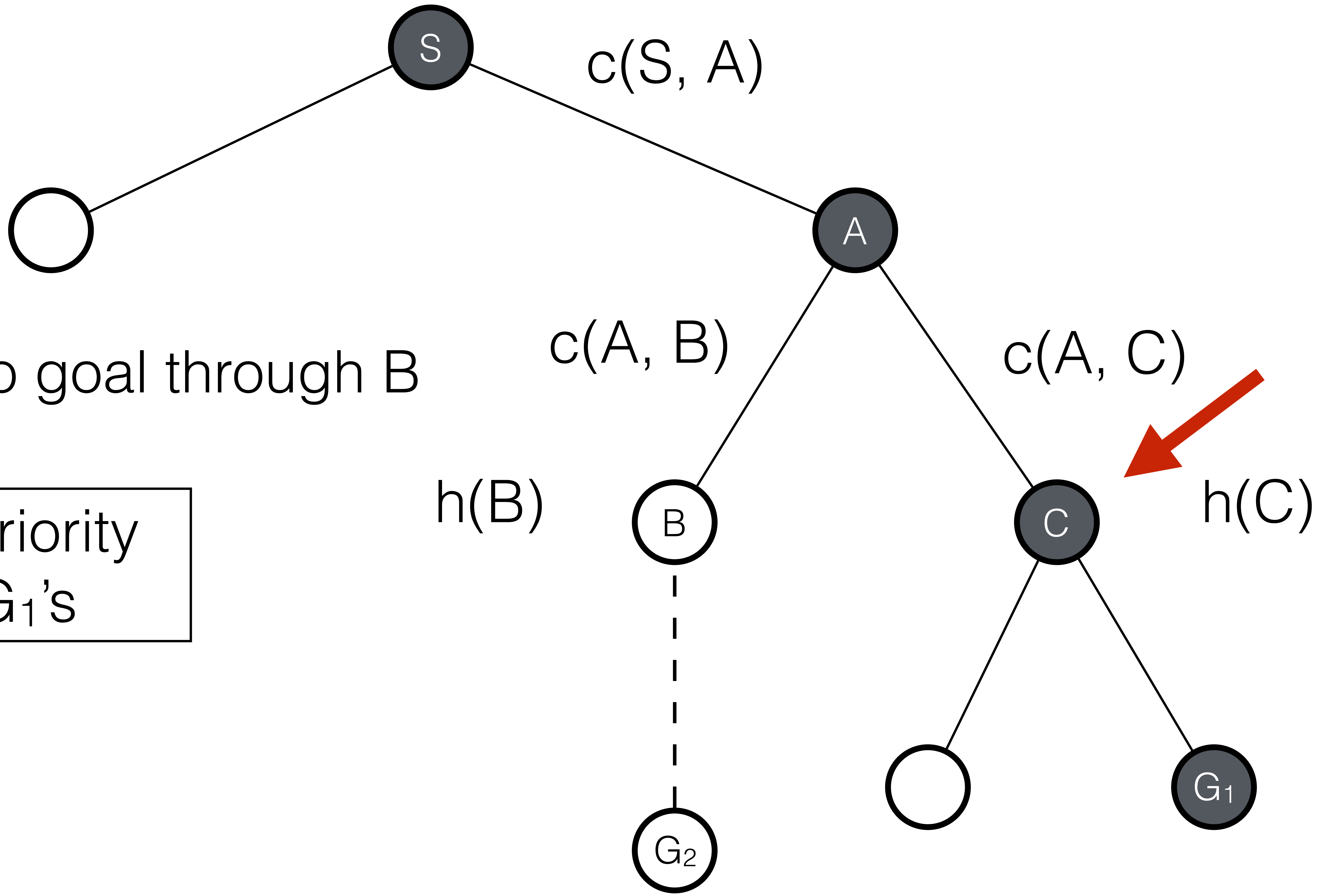
h(n) is an underestimate of cost to goal

A* in a Tree

G_1: true cost to G_1

B: underestimate of true cost to goal through B

if G_2 were cheaper, B's priority would be cheaper than G_1's



“Lemmas”

1. Priority of each node we expand is always an underestimate of true **cost to goal through node**

2. Priorities of any goal state we expand is **true cost of path to goal**

A* in a Graph

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

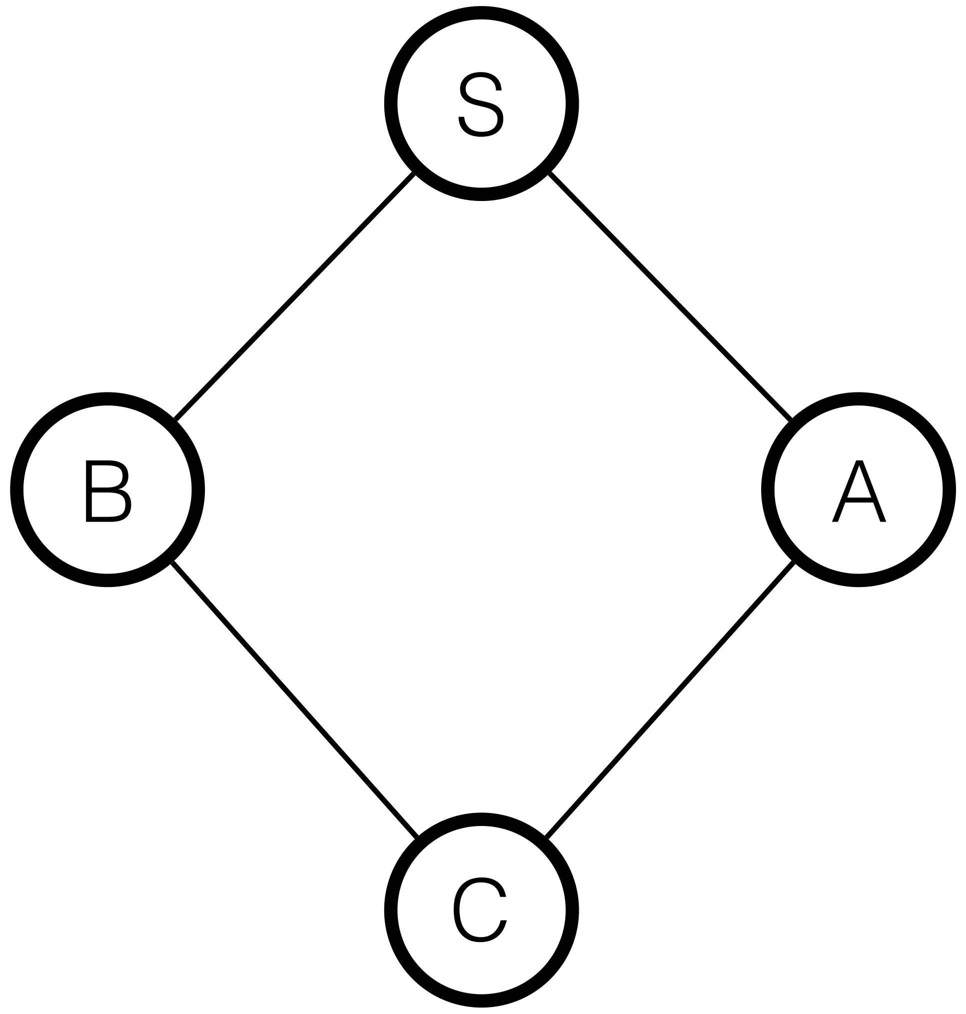
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier
only if not in the frontier or explored set



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

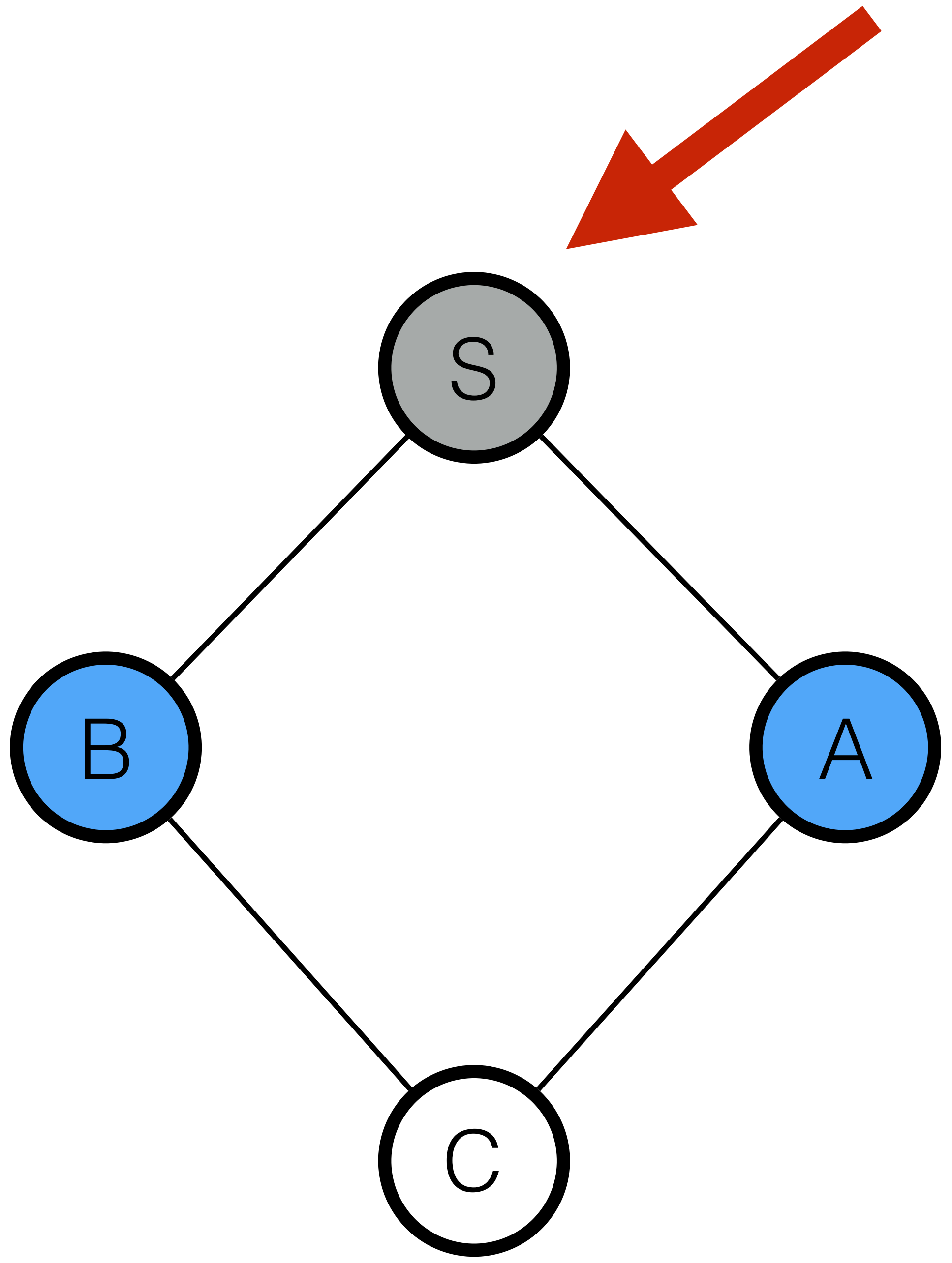
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier
only if not in the frontier or explored set



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

if the frontier is empty **then return** failure

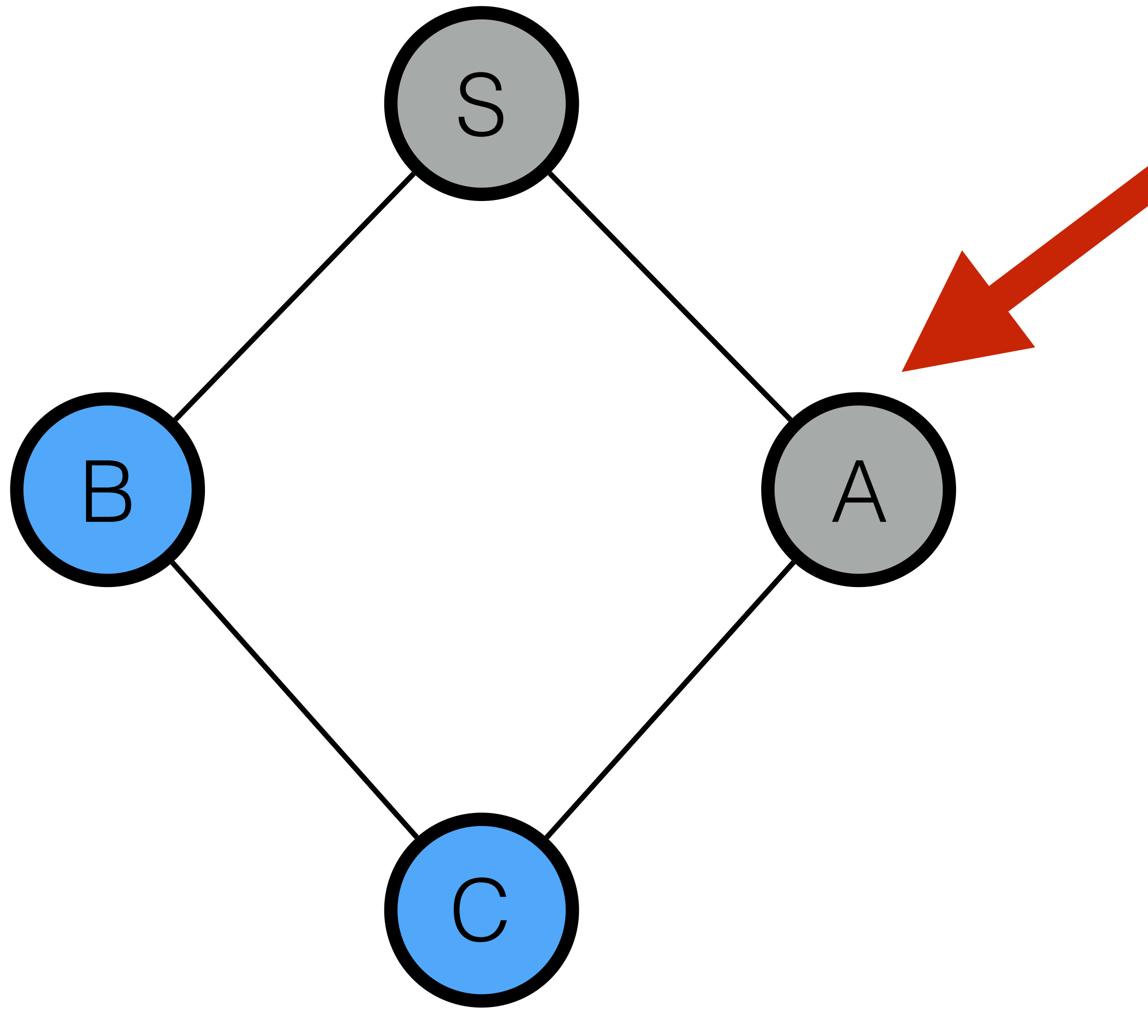
choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier

only if not in the frontier or explored set



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

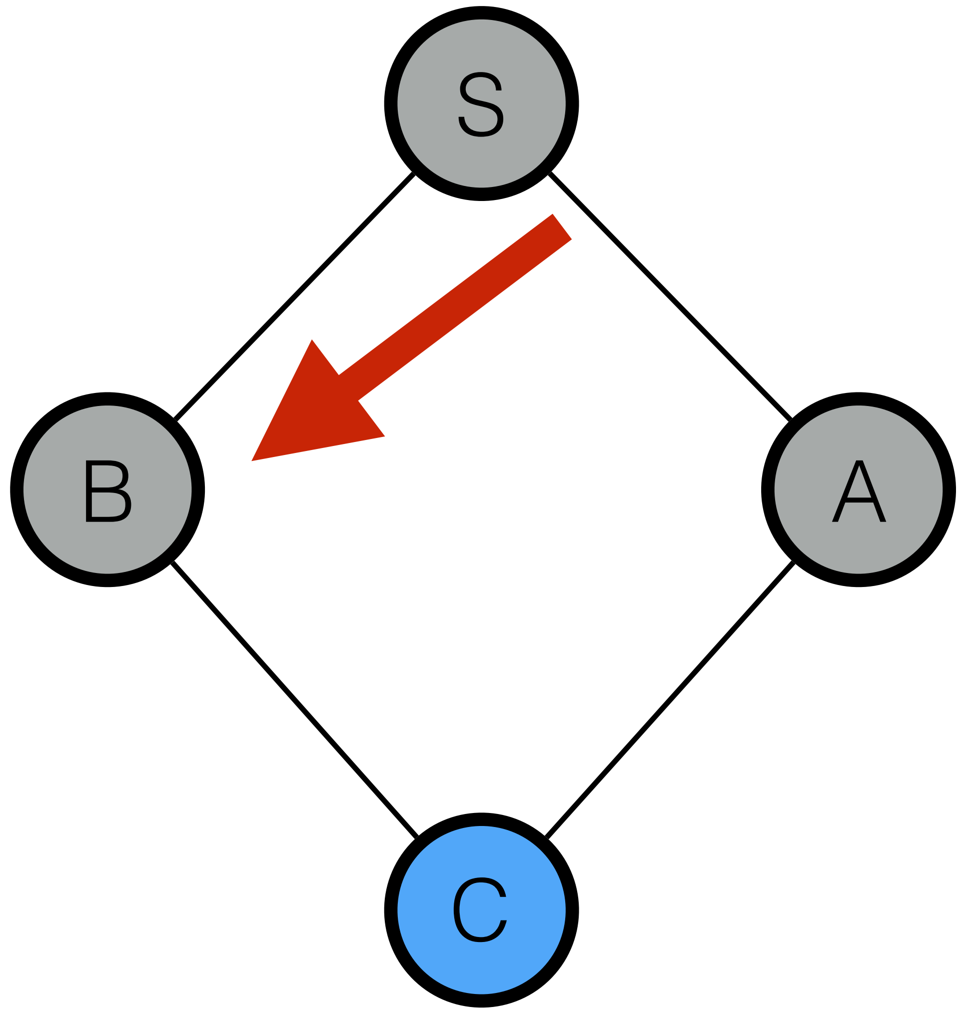
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

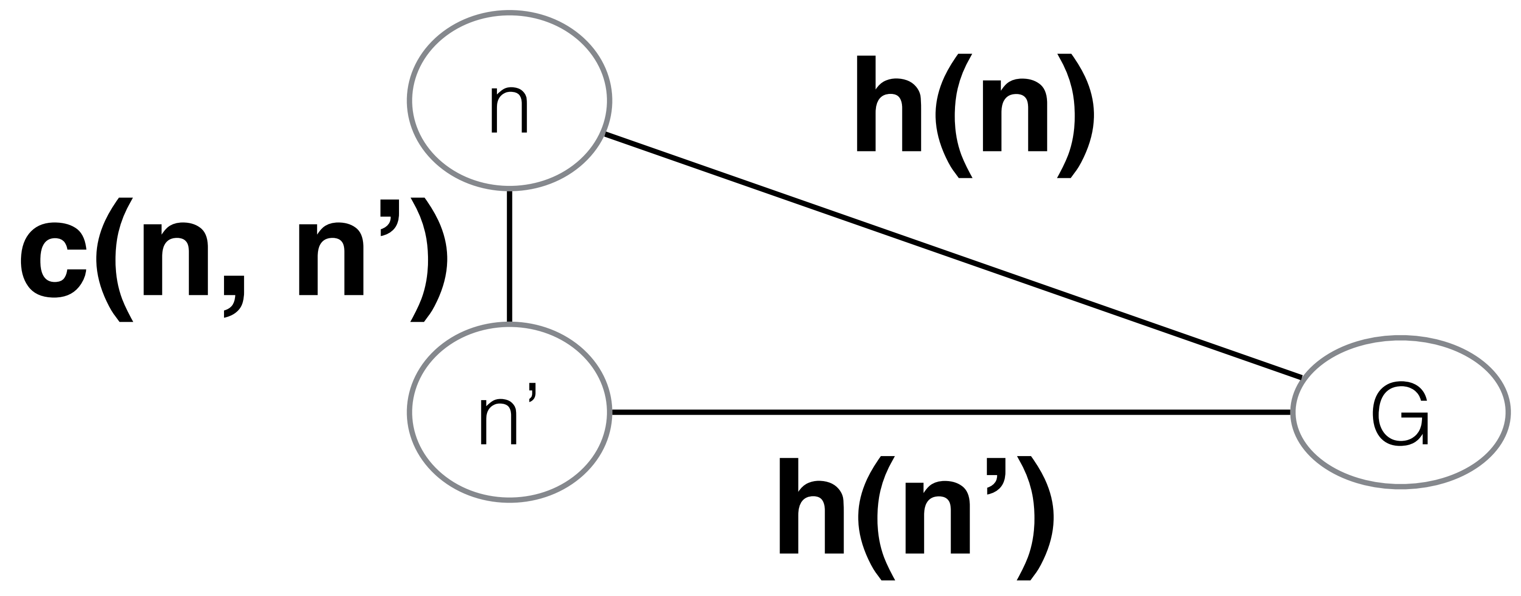
expand the chosen node, adding the resulting nodes to frontier
only if not in the frontier or explored set



Two Solutions

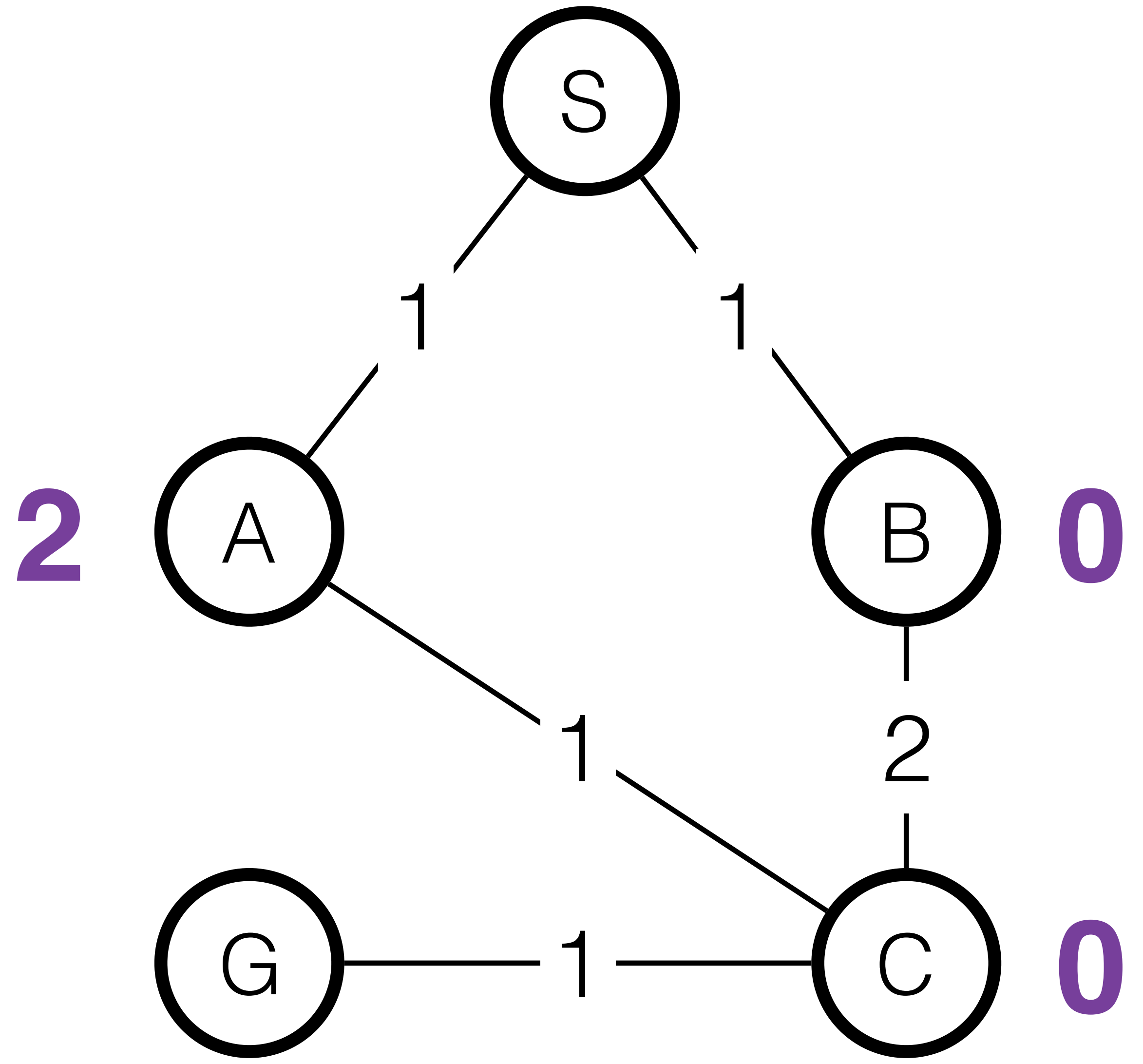
- Solution 1: If you encounter a child node already in the frontier, update the priority of the child with better score
- Solution 2: Allow multiple copies of nodes in frontier, but when selecting nodes from frontier, ignore nodes you've already expanded
- We may add nodes to frontier with overestimated costs, but every node we choose to expand will have its true shortest path cost **g(n)**

Consistency



- Definition: **h(n)** <= **c(n, n')** + **h(n')**
- Rule of thumb:
  - Design an easier search space. Set **h(n)** to cost in easier space.
  - E.g., Straight-line distance (ignoring obstacles)

Admissibility isn't Enough



Priority Queue:

S

Expand S: (A, 3), (B, 1)

Expand B: (A, 3), (C, 3)

Expand C: (A, 3), (G, 4)

Expand A: (G, 4), ~~(C, 2)~~

h(A) <= c(A, C) + h(C)

2 <= 1 + 0

Benefits from Consistency

- When expanding a node **n**, the optimal path to **n** has been found
-

Search Problems

	Graph or Tree?	States	Actions	Transition Costs	Heuristic Ideas
Car Navigation					
Rubik's Cube	graph	Config of cube	Rotating slices	Uniform	How many squares in right side (divide by 12)
16 Puzzle					
N-Queens					

Final Notes

- Store path from initial state
- Python tuples vs. lists
- Next class: Python practice
- Friday: Homework 1 coding