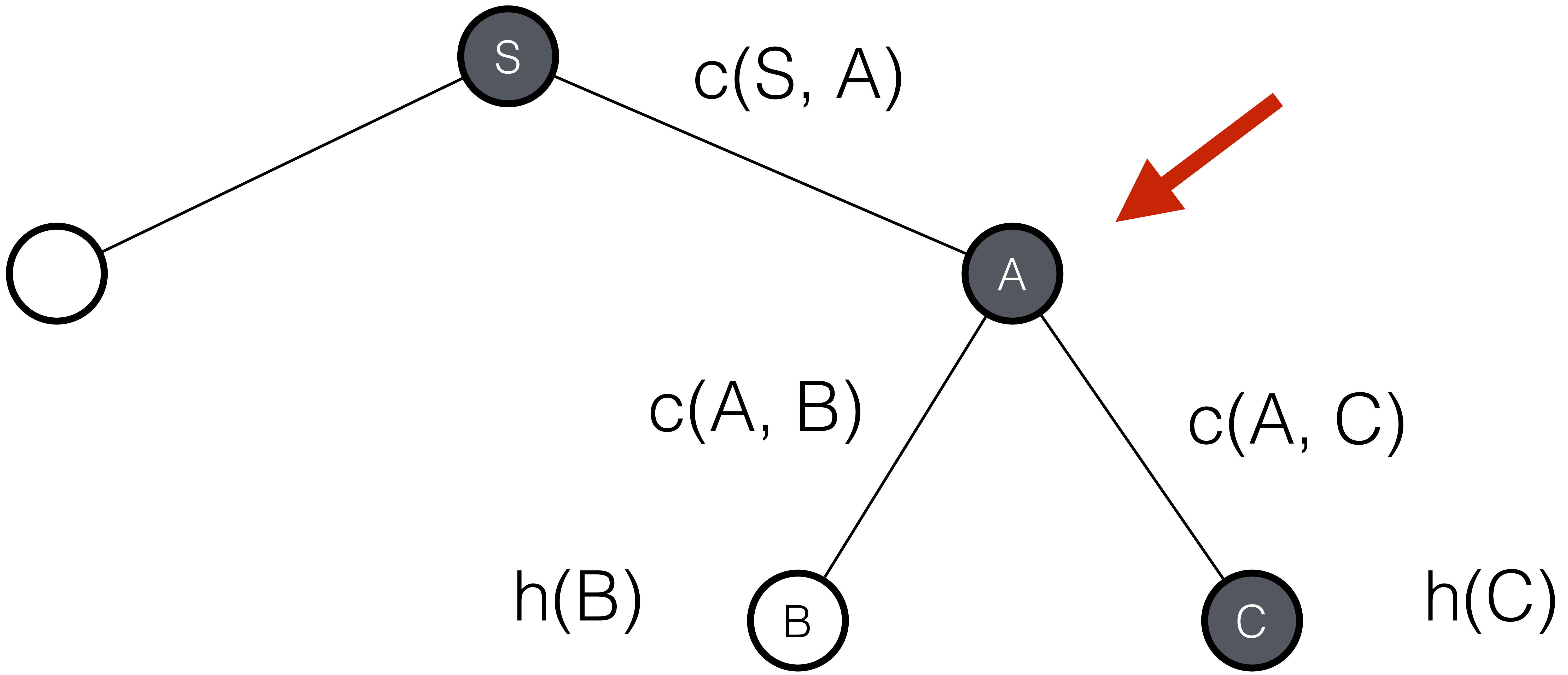
A* Optimality

Bert Huang

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**

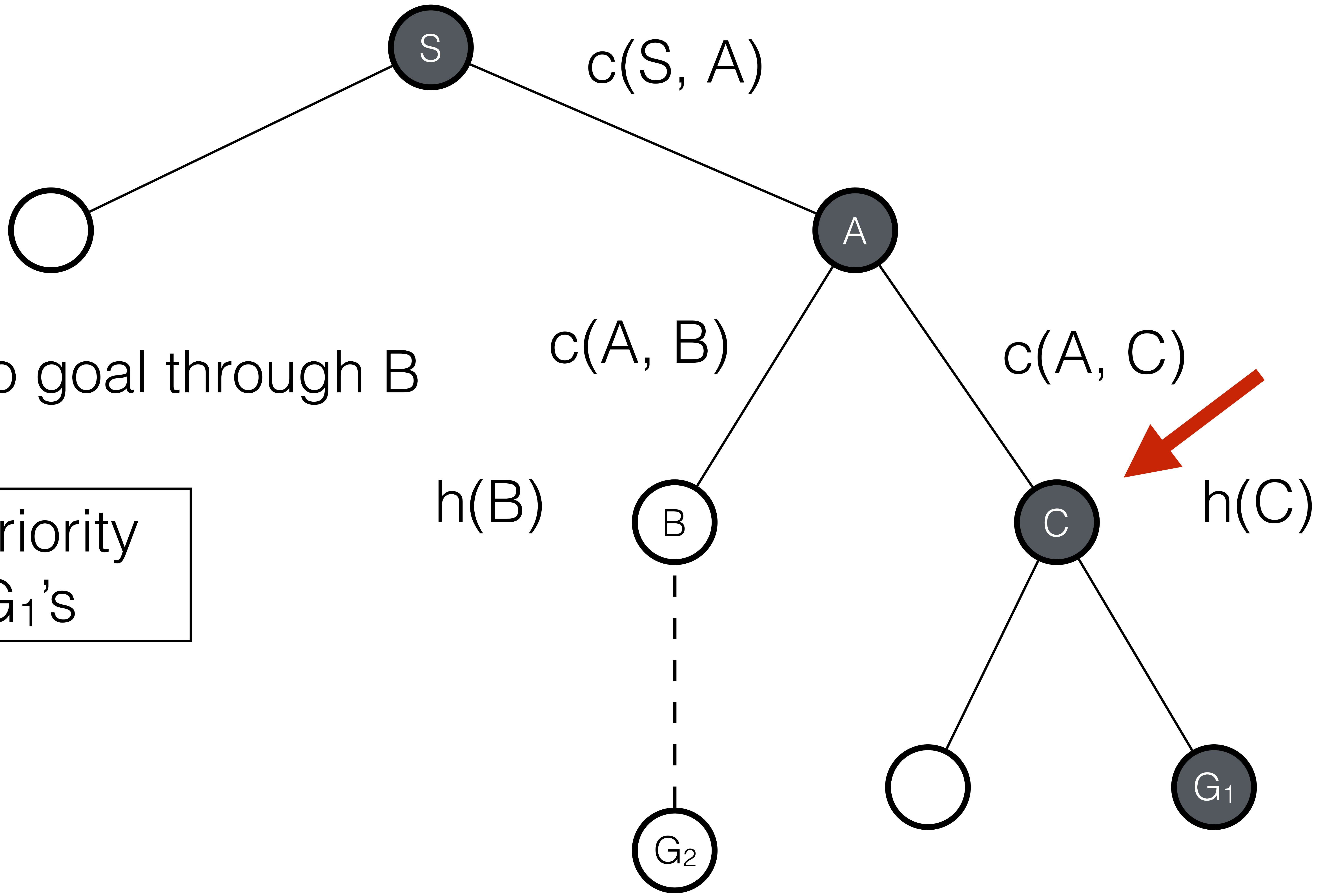
h(n) is an underestimate of cost to goal

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"

- Priority of each node we expand is always an underestimate of true **cost to goal through node**
- Priorities of any goal state we expand is **true cost of path to goal**

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:

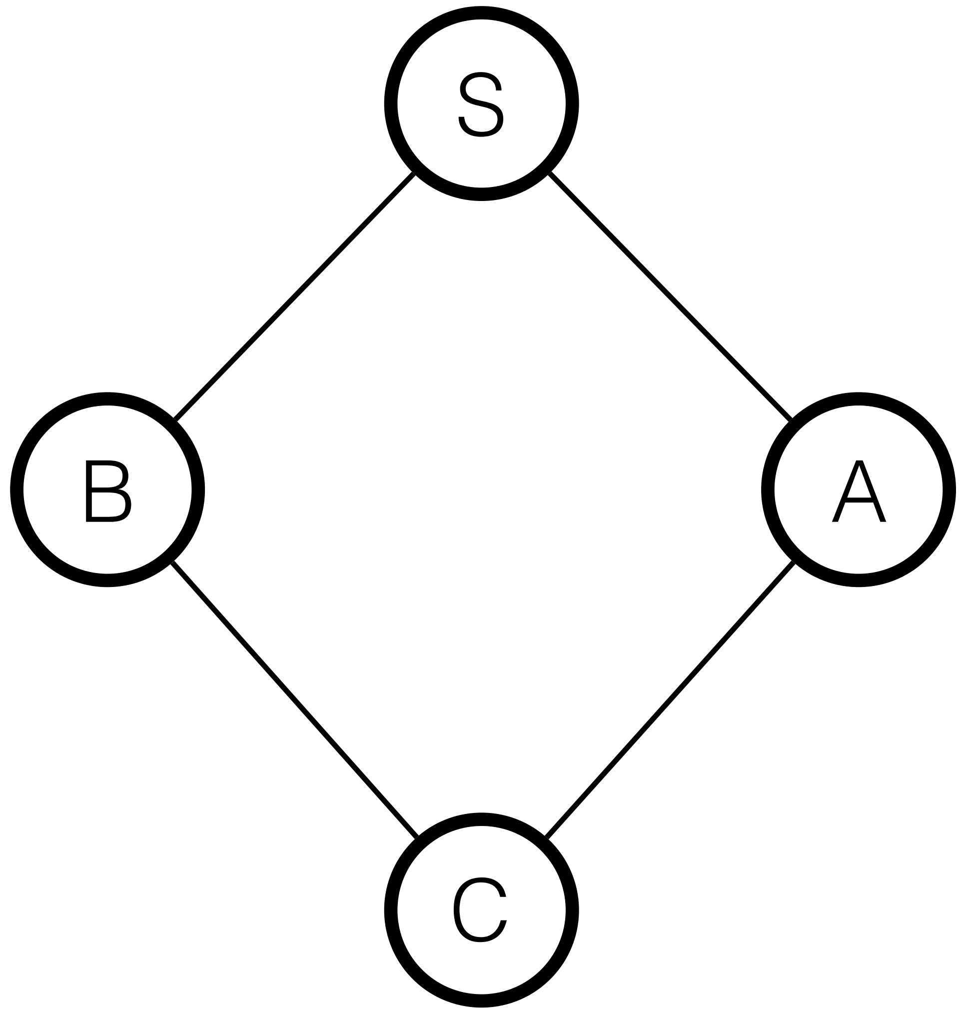
if the frontier is empty **then return** failure

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:

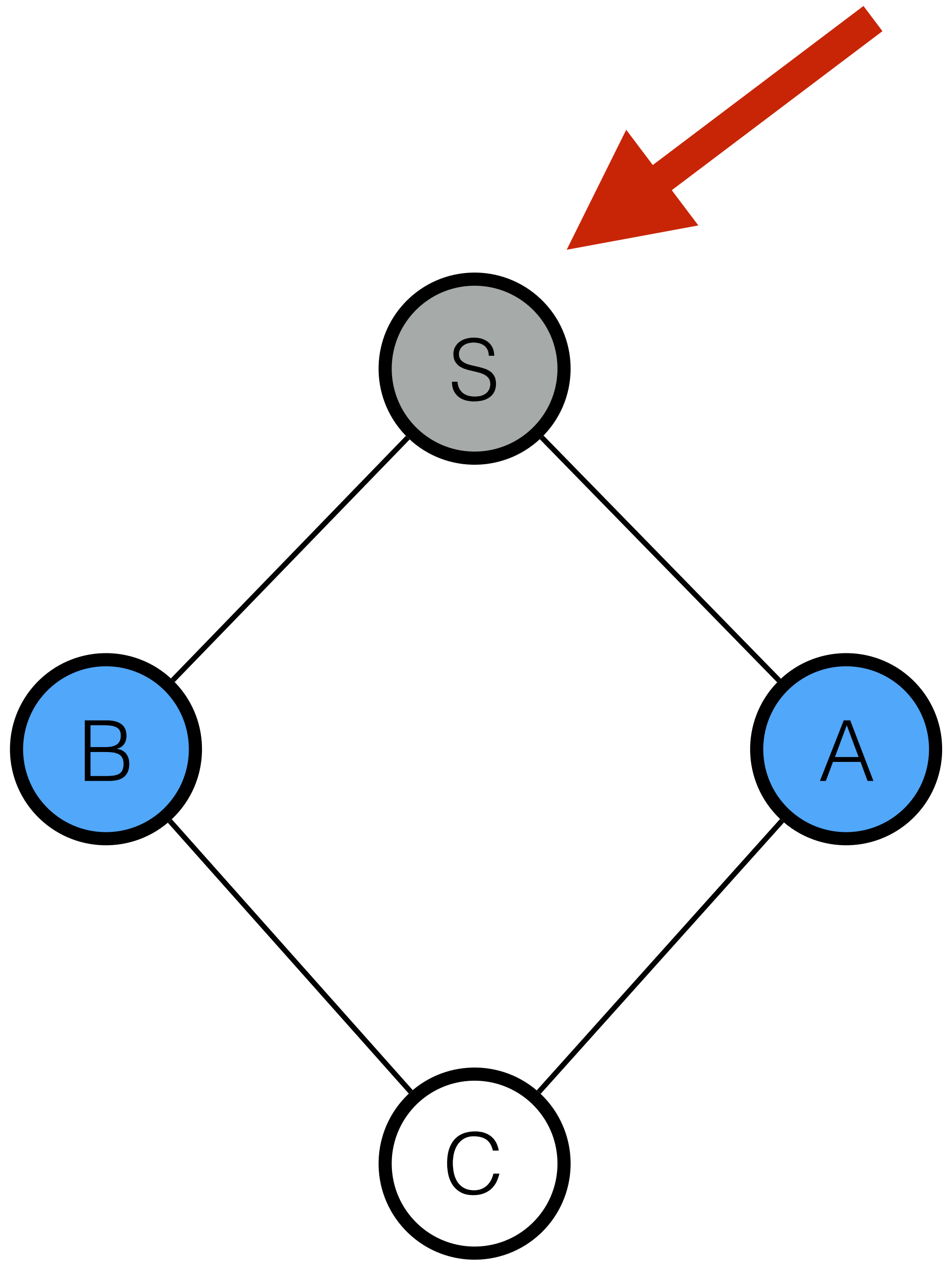
if the frontier is empty **then return** failure

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:

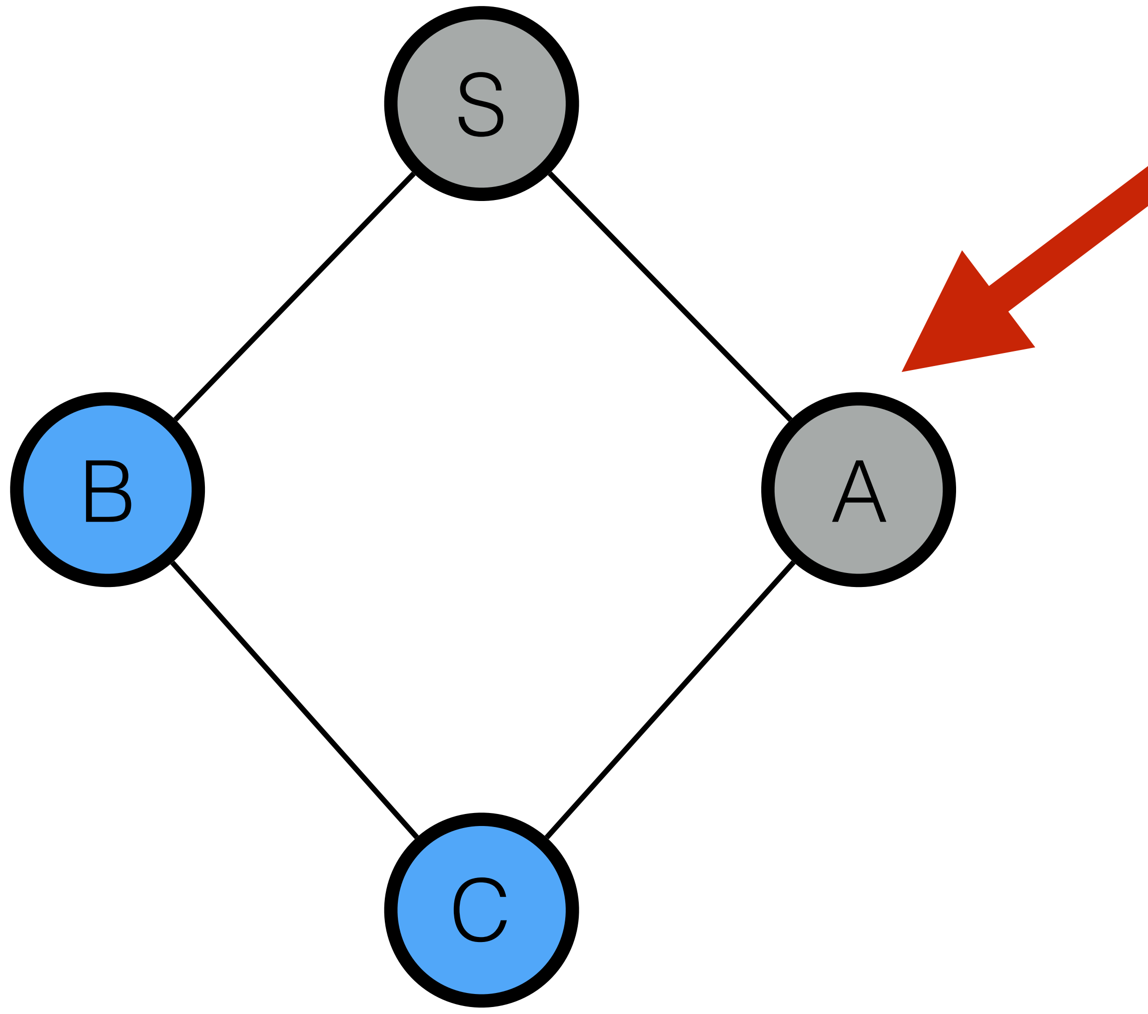
if the frontier is empty **then return** failure

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:

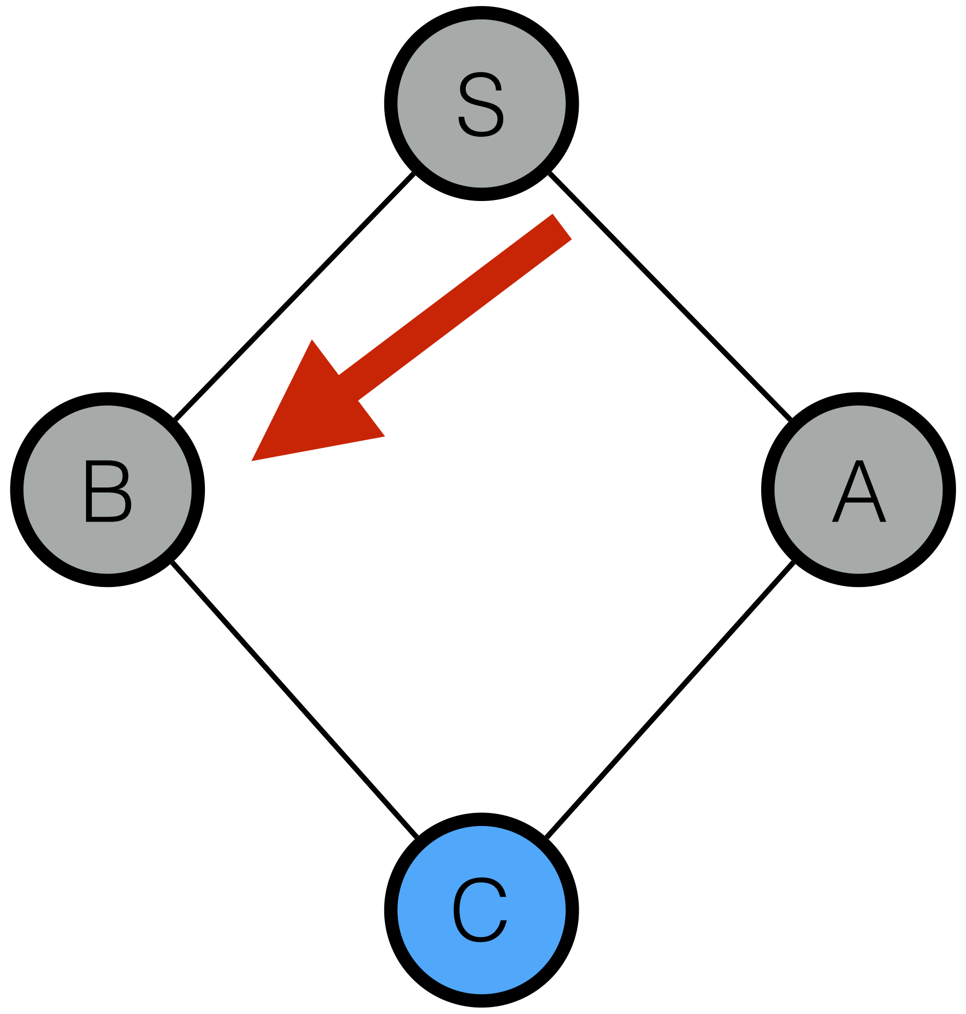
if the frontier is empty **then return** failure

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 visited

- 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)**