

CS 480

Introduction to Artificial Intelligence

January 23, 2024

Announcements / Reminders

- Please follow the Week 02 To Do List instructions (if you haven't already):
- Quiz #01: due on Sunday (01/28/24) at 11:59 PM CST
- Written Assignment #01: to be posted soon

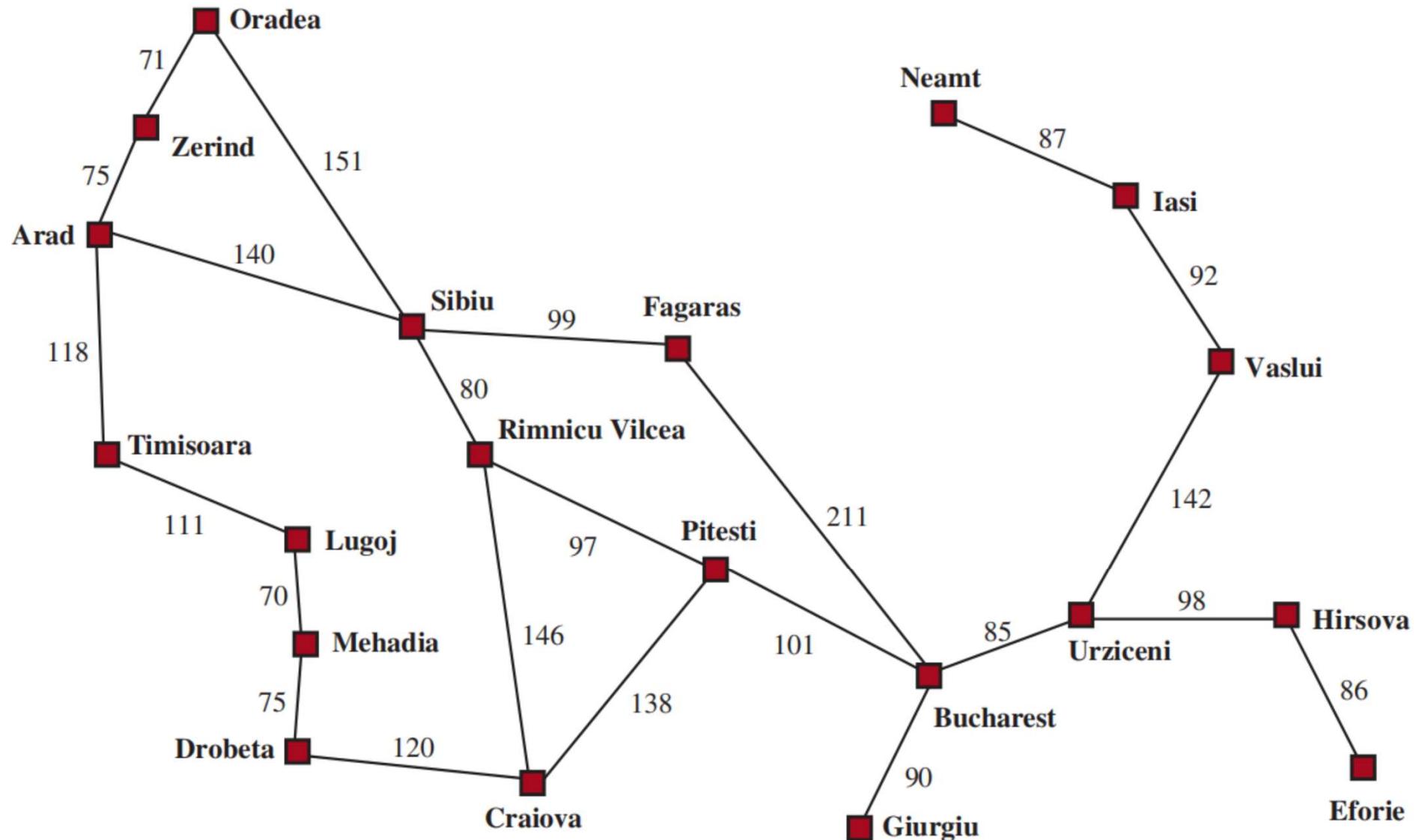
Plan for Today

- Solving problems by Searching

Defining Search Problem

- Define a set of possible states: **State Space**
- Specify **Initial State**
- Specify **Goal State(s)** (there can be multiple)
- Define a FINITE set of possible **Actions** for EACH state in the State Space
- Come up with a **Transition Model** which describes what each action does
- Specify the **Action Cost Function**: a function that gives the cost of applying action a in state s

Sample Problem: Romanian Roadtrip



Problem: Get from Arad to Bucharest efficiently (for example: quickly or cheaply).

Search Problem: Romanian Roadtrip

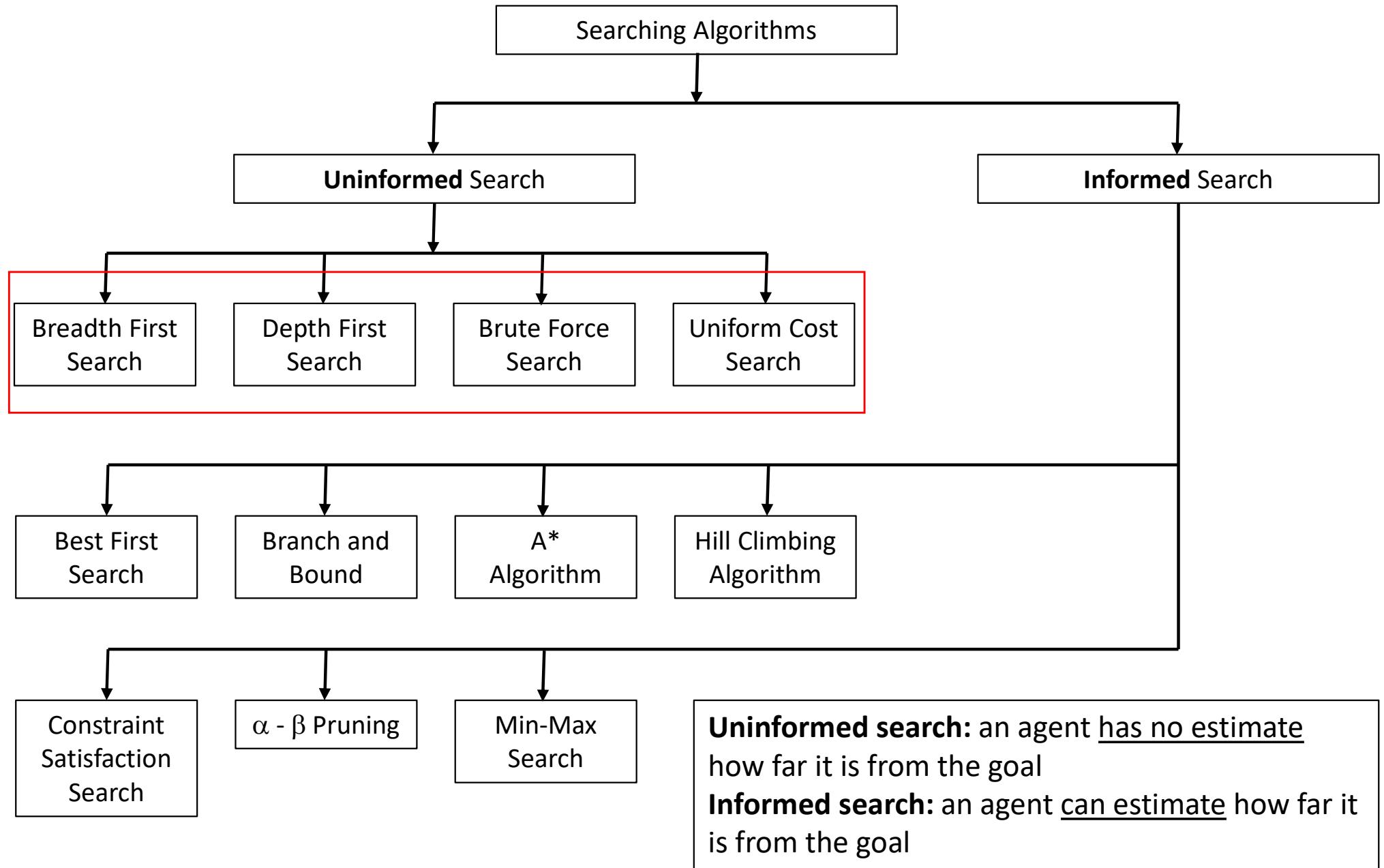
- State Space: a map of Romania
- Initial State: Arad
- Goal State: Bucharest
- Actions:
 - for example: $\text{ACTIONS}(\text{Arad}) = \{\text{ToSibiu}, \text{ToTimisoara}, \text{ToZerind}\}$
- Transition Model:
 - for example: $\text{RESULT}(\text{Arad}, \text{ToZerind}) = \text{Zerind}$
- Action Cost Function [$\text{ActionCost}(S_{\text{current}}, a, S_{\text{next}})$]
 - for example: $\text{ActionCost}(\text{Arad}, \text{ToSibiu}, \text{Sibiu}) = 140$

Measuring Searching Performance

Search algorithms can be evaluated in four ways:

- **Completeness**: Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
- **Cost optimality**: Does it find a solution with the lowest path cost of all solutions?
- **Time complexity**: How long does it take to find a solution? (in seconds, actions, states, etc.)
- **Space complexity**: How much memory is needed to perform the search?

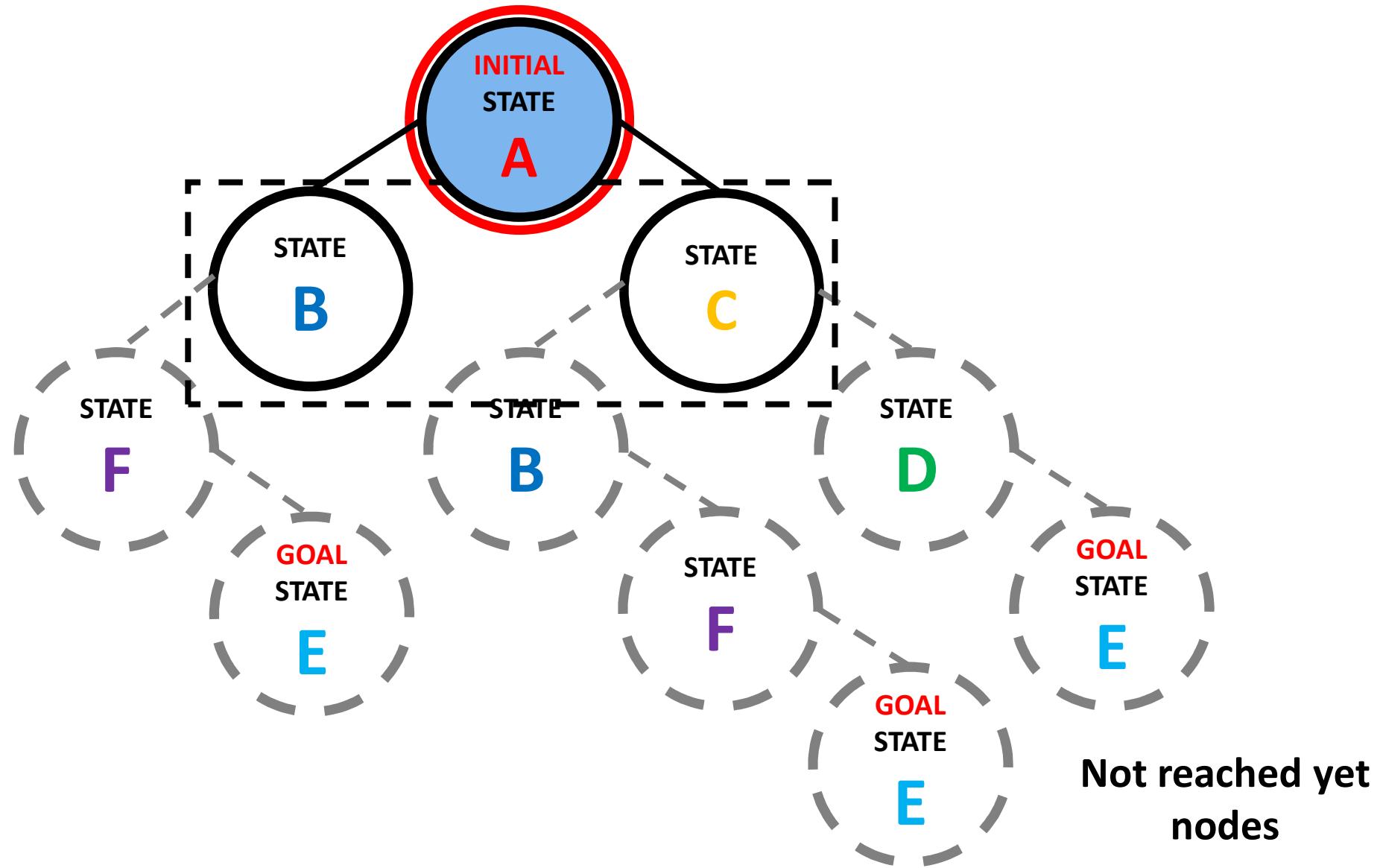
Selected Searching Algorithms



Uninformed Searching

- **Breadth First Search (BFS):**
 - Will find a solution with a minimal number of actions
 - Large memory requirement
 - Only relatively small problem instances are tractable
- **Depth First Search:**
 - May NOT find a solution with a minimal number of actions
 - Requires less memory than BFS (for tree search)
 - Backtracking (one child / successor generated at a time)
- **Brute Force Search:** depends on the approach -> bad
- **Uniform Cost Search:** minimize solution / path cost

Expansion: Which Node to Expand?



Evaluation function

Calculate / obtain:

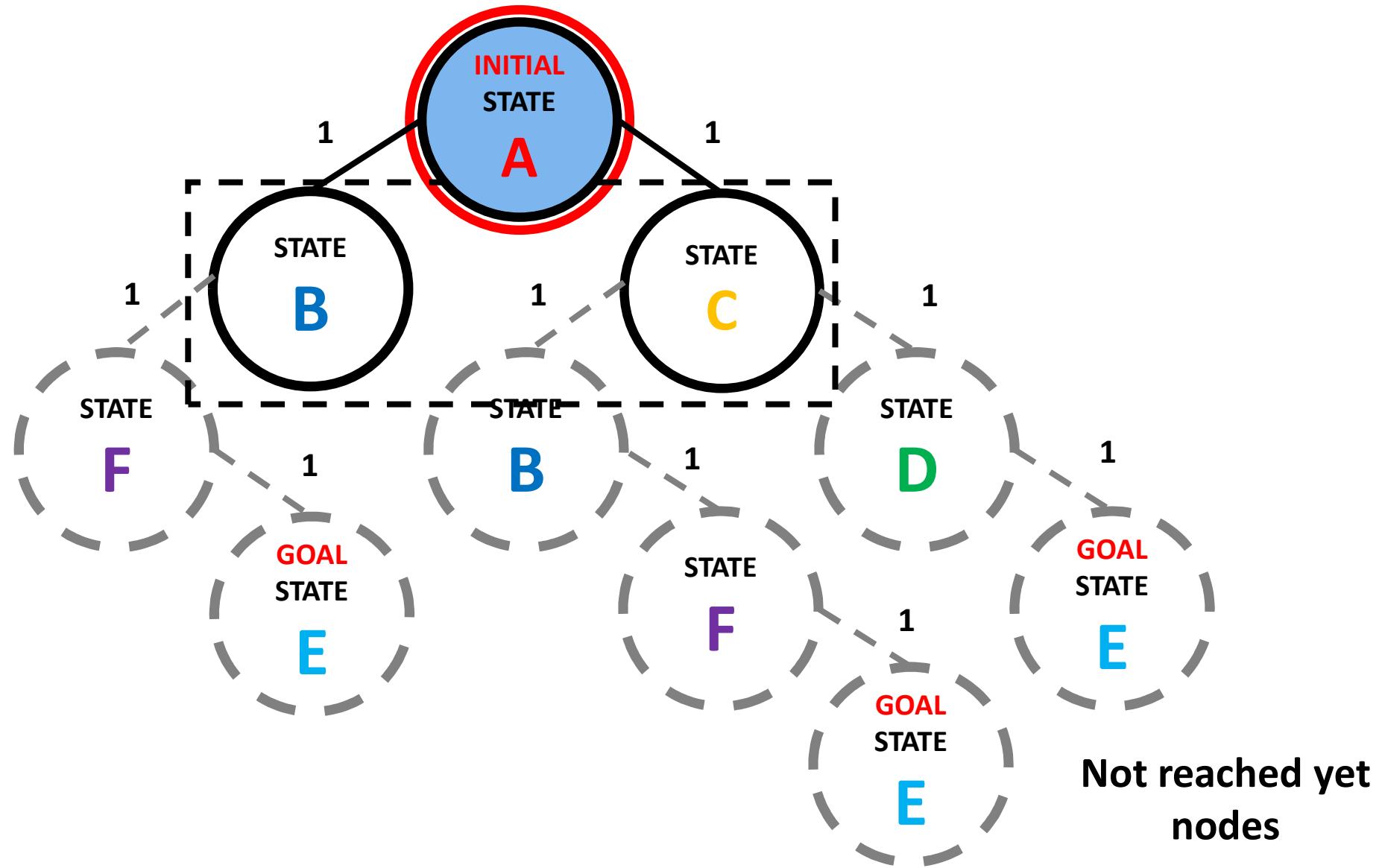
$$f(n) = f(\text{State } n)$$

$$f(n) = f(\text{relevant information about State } n)$$

A state n with minimum $f(n)$ should be chosen for expansion

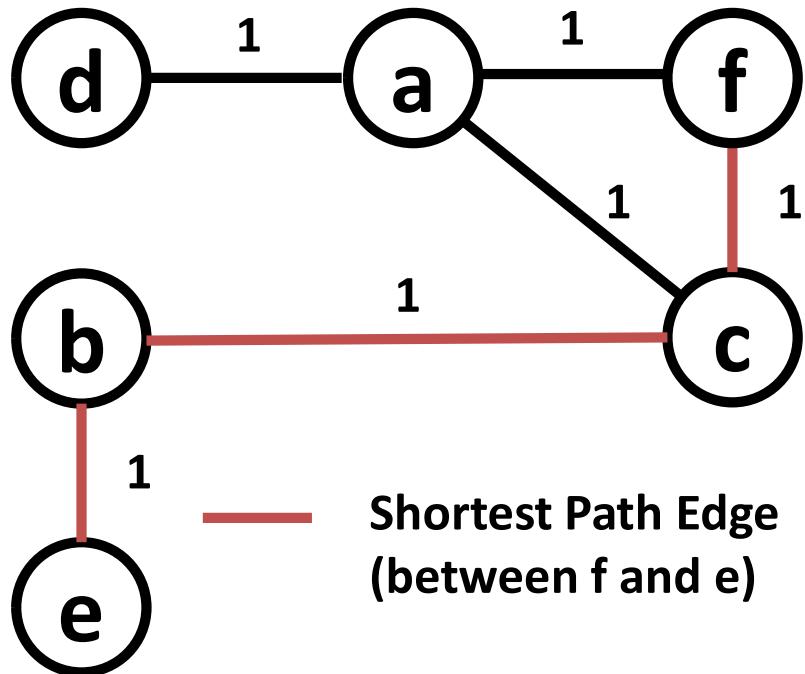
What about ties?

Search Tree: Uniform Action Cost



Uniform Cost Search | Dijkstra's Algo

Weighted Graph G



Popular algorithms:

- Dijkstra's algorithm

Shortest Path Problem

Shortest path problem:

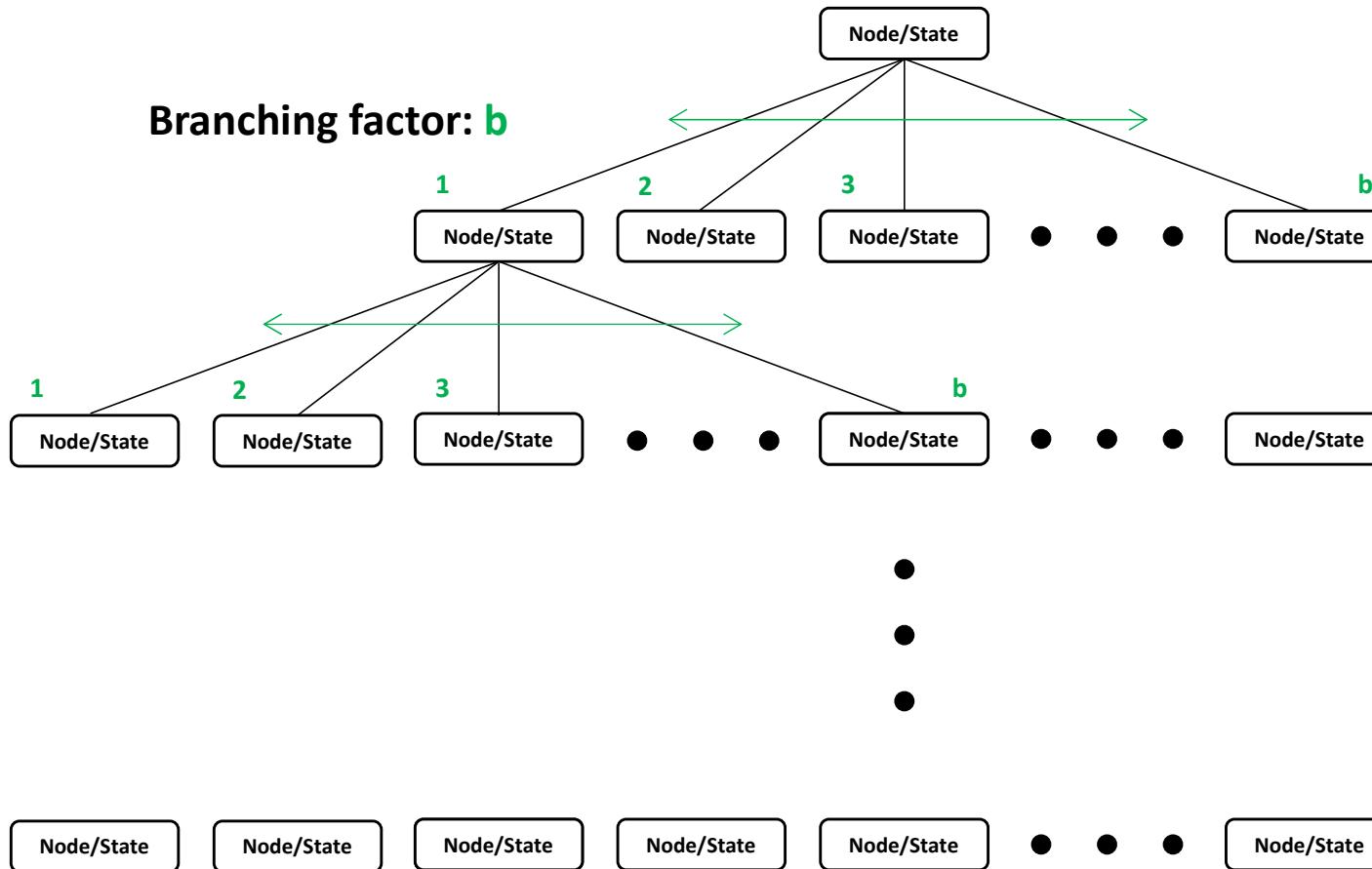
Given a weighted graph $G(V, E, w)$ and two vertices a, b in V , find the shortest path between vertices a and b (**all edge weights are equal**).

BFS and UCS: Pseudocode

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node  $\leftarrow$  NODE(problem.INITIAL)
  if problem.Is-GOAL(node.STATE) then return node
  frontier  $\leftarrow$  a FIFO queue, with node as an element
  reached  $\leftarrow$  {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node  $\leftarrow$  POP(frontier)
    for each child in EXPAND(problem, node) do
      s  $\leftarrow$  child.STATE
      if problem.Is-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

```
function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

Let's Go Back to Depth First Search



Depth: 0 | $N_0 = 1$

Depth: 1 | $N_1 = b$

Depth: 2 | $N_2 = b^2$

•
•
•

Depth: d | $N_d = b^d$

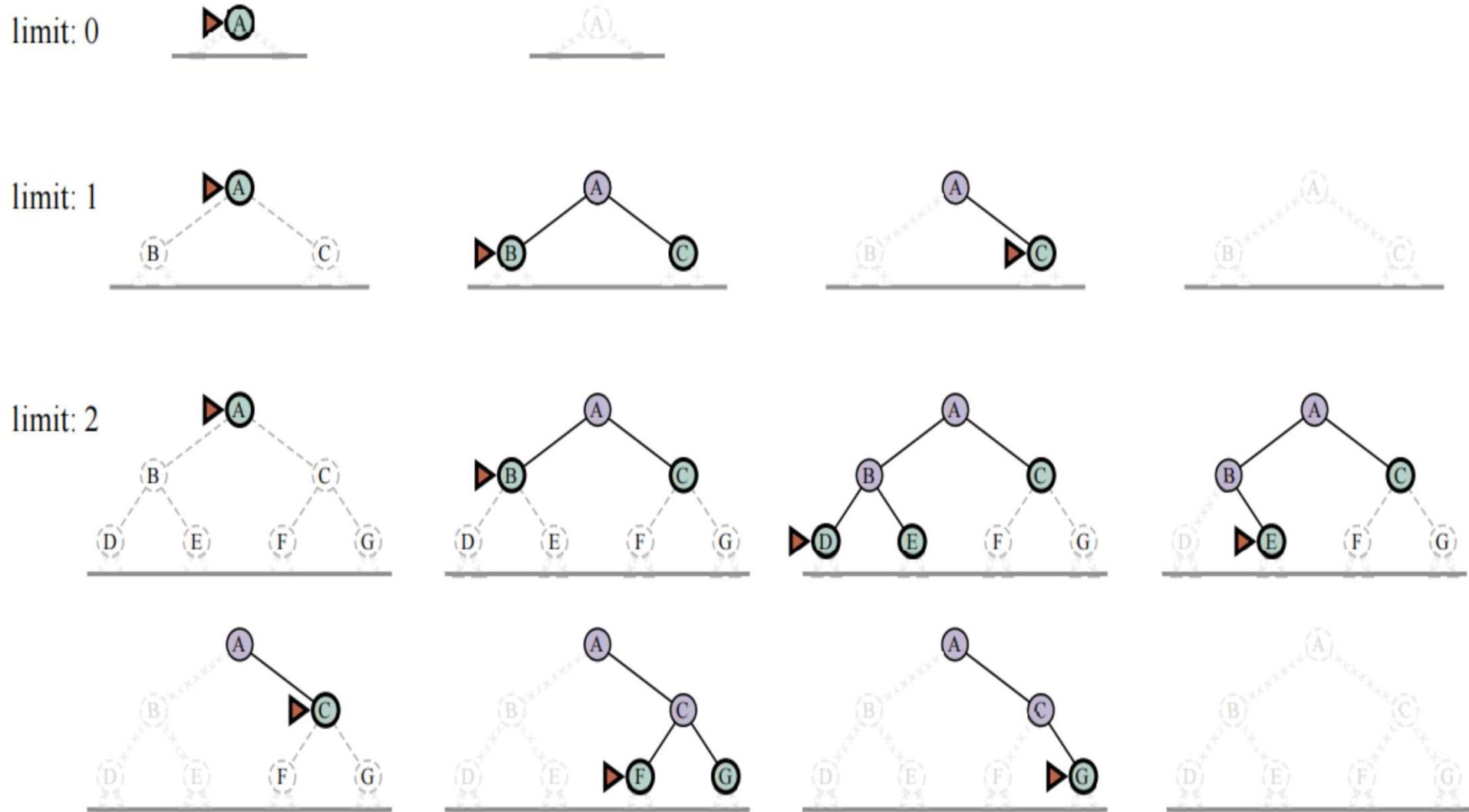
Tree depth is an issue!

“Controlled” DFS: Pseudocode

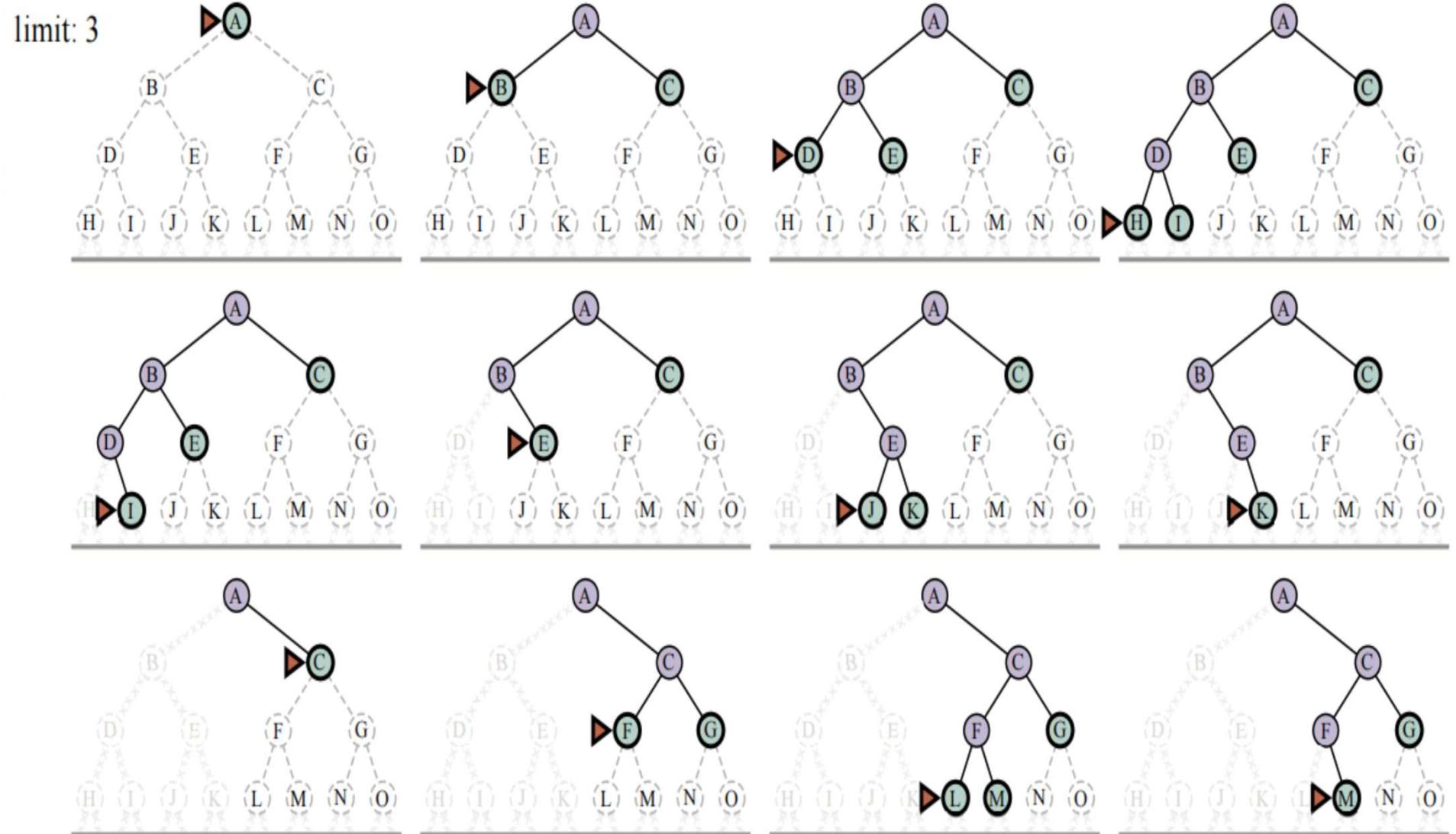
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result

function DEPTH-LIMITED-SEARCH(problem,  $\ell$ ) returns a node or failure or cutoff
frontier  $\leftarrow$  a LIFO queue (stack) with NODE(problem.INITIAL) as an element
result  $\leftarrow$  failure
while not Is-EMPTY(frontier) do
    node  $\leftarrow$  POP(frontier)
    if problem.Is-GOAL(node.STATE) then return node
    if DEPTH(node)  $>$   $\ell$  then
        result  $\leftarrow$  cutoff
    else if not Is-CYCLE(node) do
        for each child in EXPAND(problem, node) do
            add child to frontier
    return result
```

Iterative Deepening DFS: Illustration



Iterative Deepening DFS: Illustration

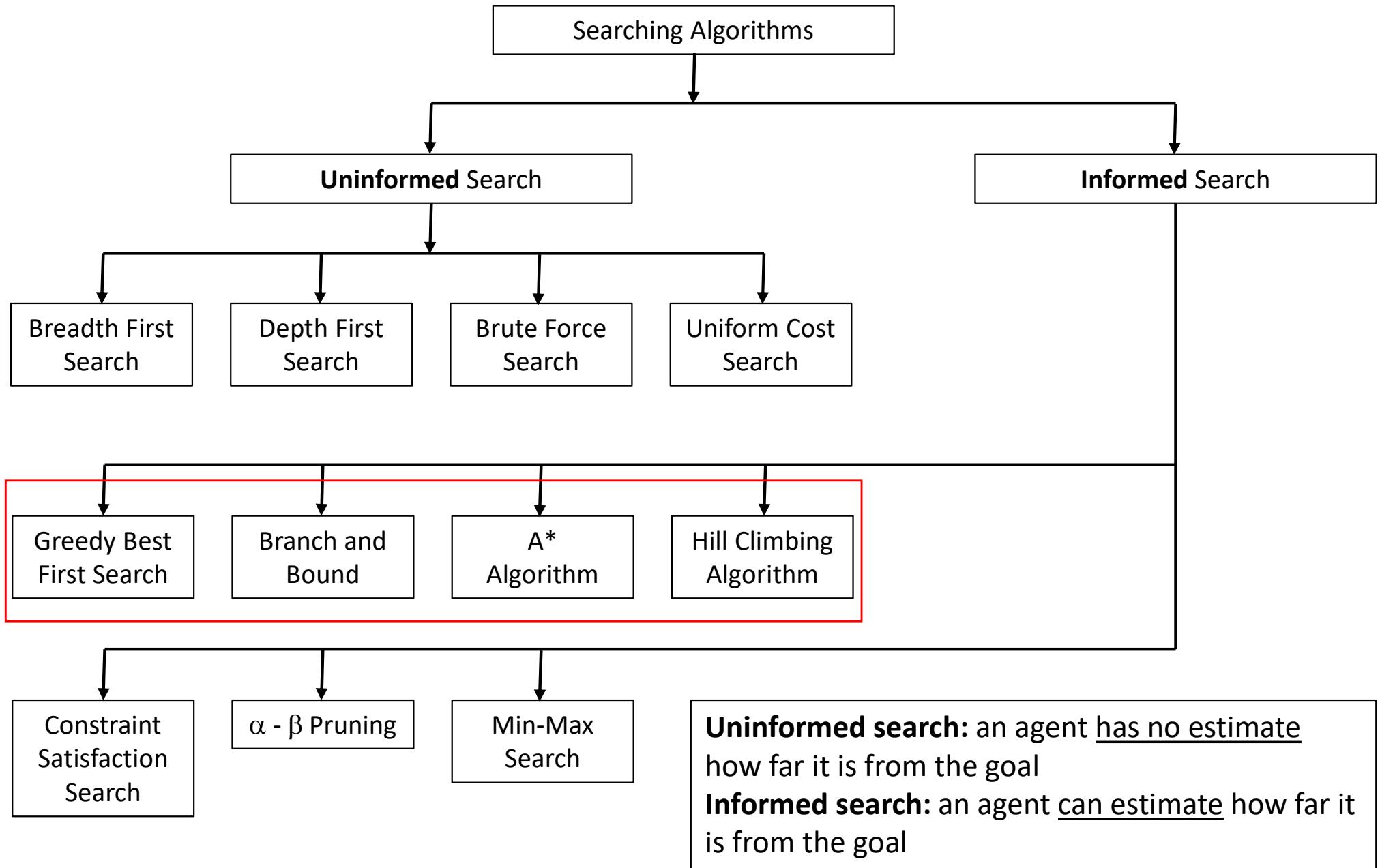


Uninformed Search Algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ¹	Yes ^{1,2}	No	No	Yes ¹	Yes ^{1,4}
Optimal cost?	Yes ³	Yes	No	No	Yes ³	Yes ^{3,4}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: ¹ complete if b is finite, and the state space either has a solution or is finite. ² complete if all action costs are $\geq \epsilon > 0$; ³ cost-optimal if action costs are all identical; ⁴ if both directions are breadth-first or uniform-cost.

Selected Searching Algorithms



Informed Search and Heuristics

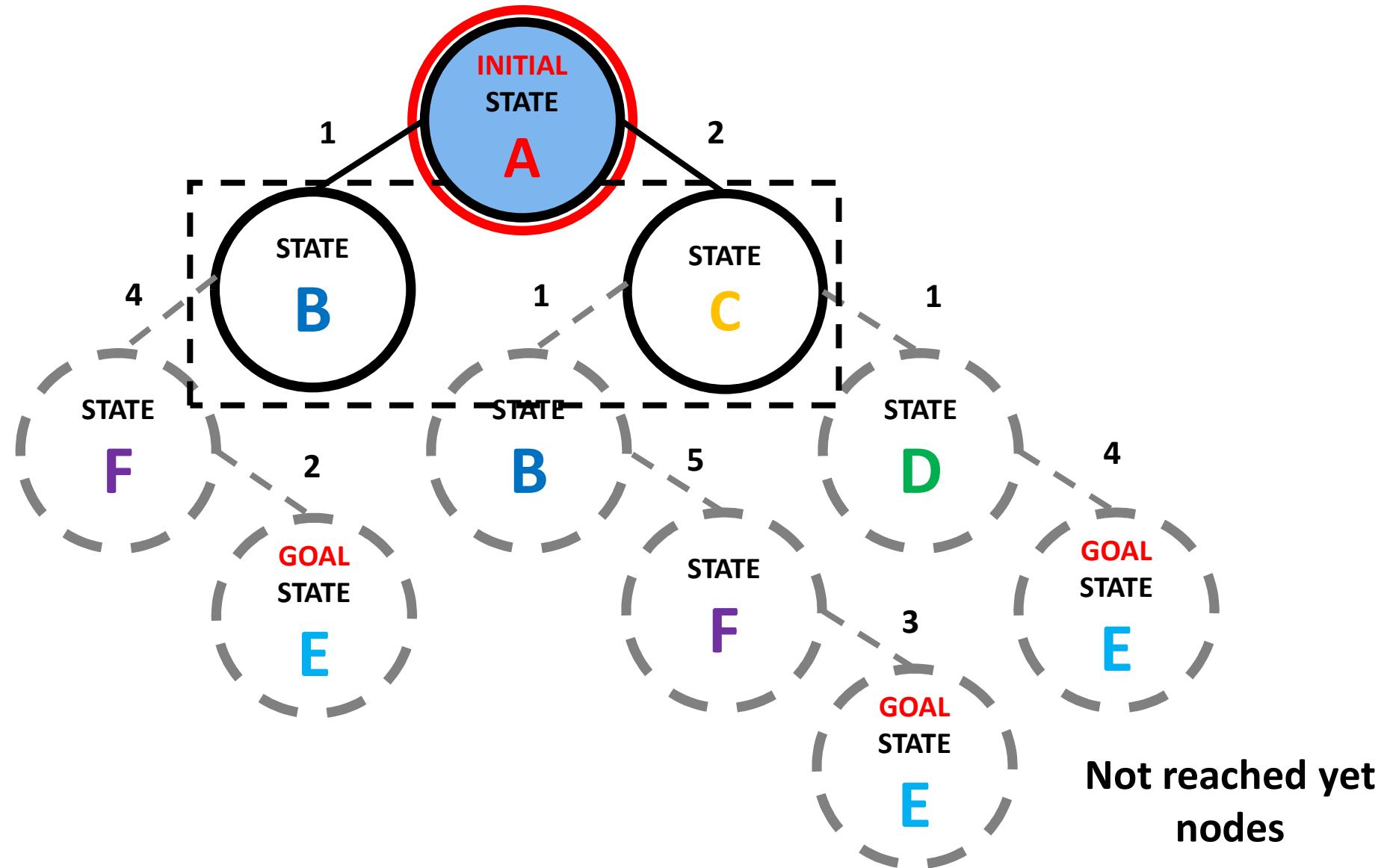
Informed search relies on **domain-specific knowledge / hints** that help locate the goal state.

$$h(n) = h(\text{State } n)$$

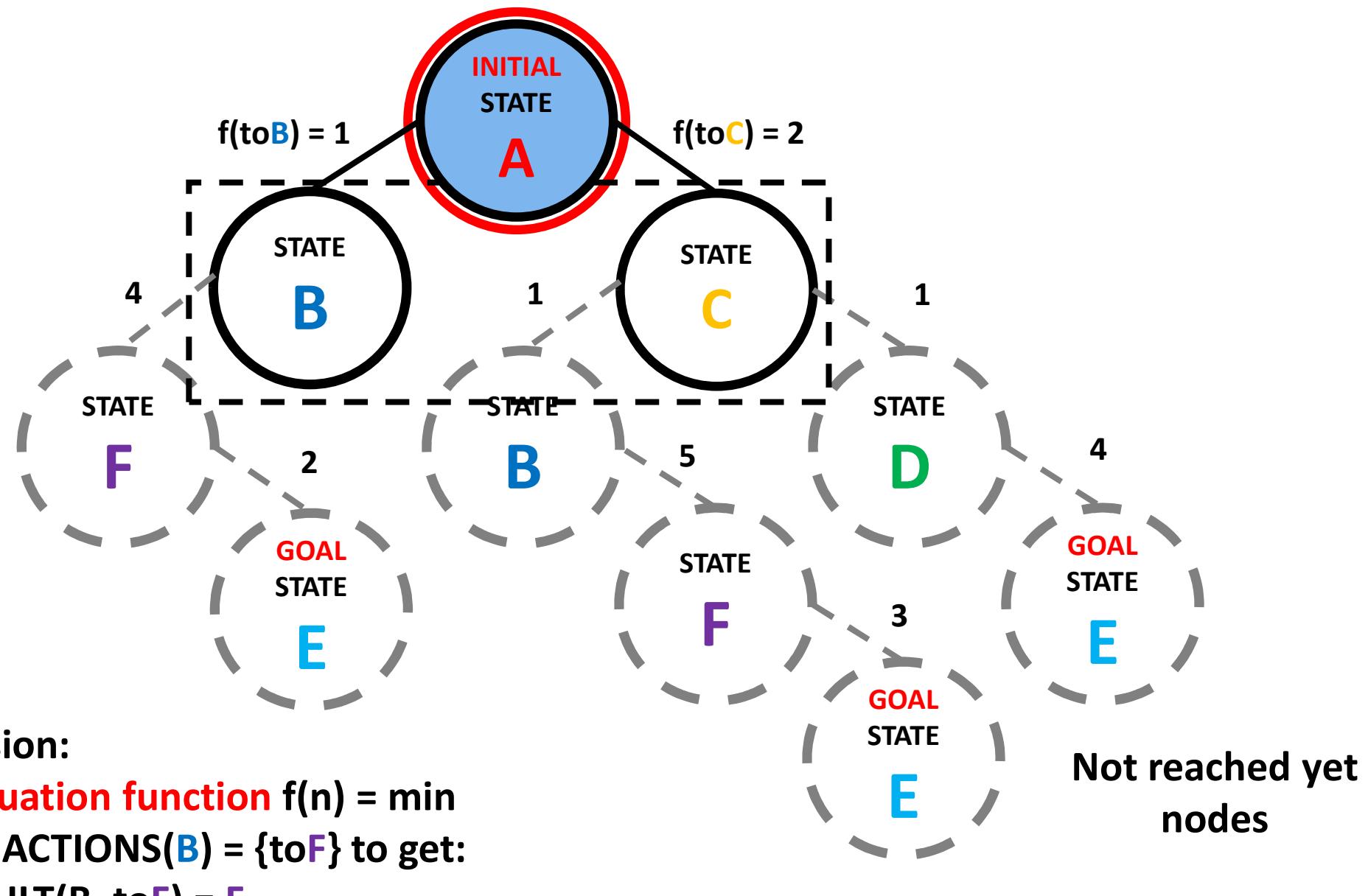
$$h(n) = n(\text{relevant information about State } n)$$

$h(n)$: heuristic function - estimated cost of the cheapest path from State n to the goal state

Search Tree: Variable Action Cost

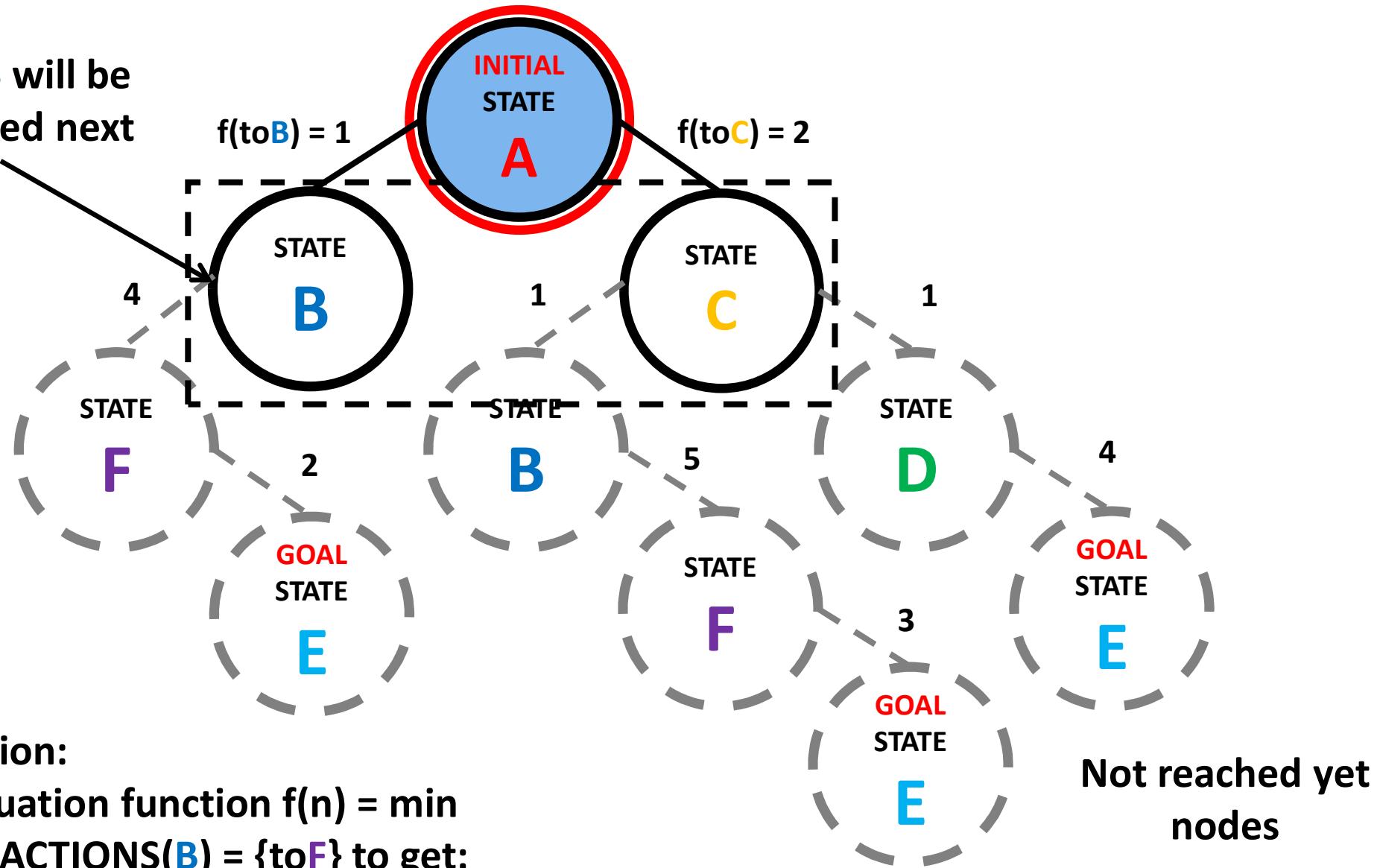


Search Tree: Variable Action Cost



Search Tree Intuition: “Take Best First”

Node B will be expanded next

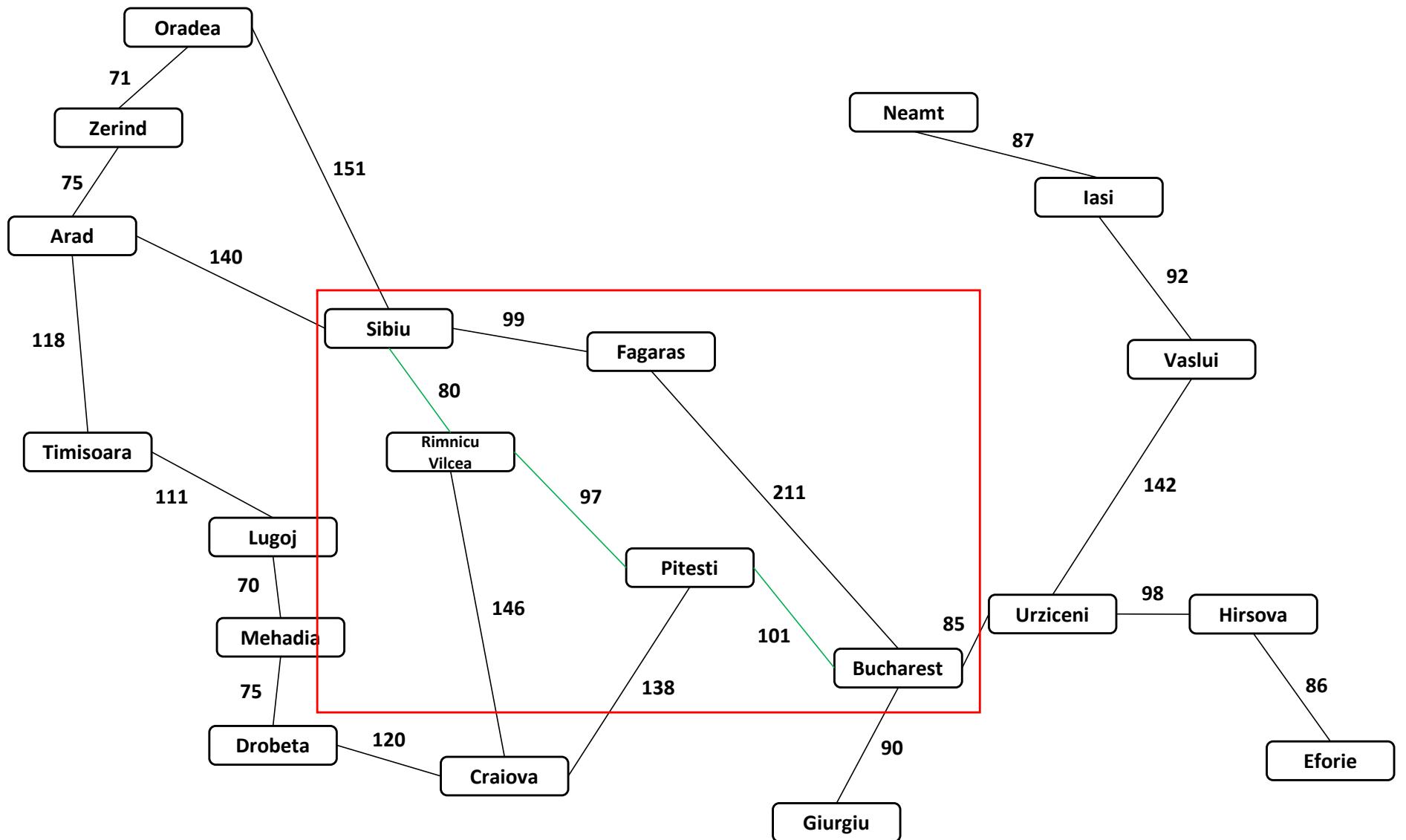


Expansion:

- Evaluation function $f(n) = \min$
- Use $\text{ACTIONS}(B) = \{\text{toF}\}$ to get:
- $\text{RESULT}(B, \text{toF}) = F$

Not reached yet
nodes

“Take Best First”: Issue



Best-First (CLASS) Search: Pseudocode

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.Is-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure

function EXPAND(problem, node) yields nodes
    s  $\leftarrow$  node.STATE
    for each action in problem.ACTIONS(s) do
        s'  $\leftarrow$  problem.RESULT(s, action)
        cost  $\leftarrow$  node.PATH-COST + problem.ACTION-COST(s, action, s')
        yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Informed Search and Heuristics

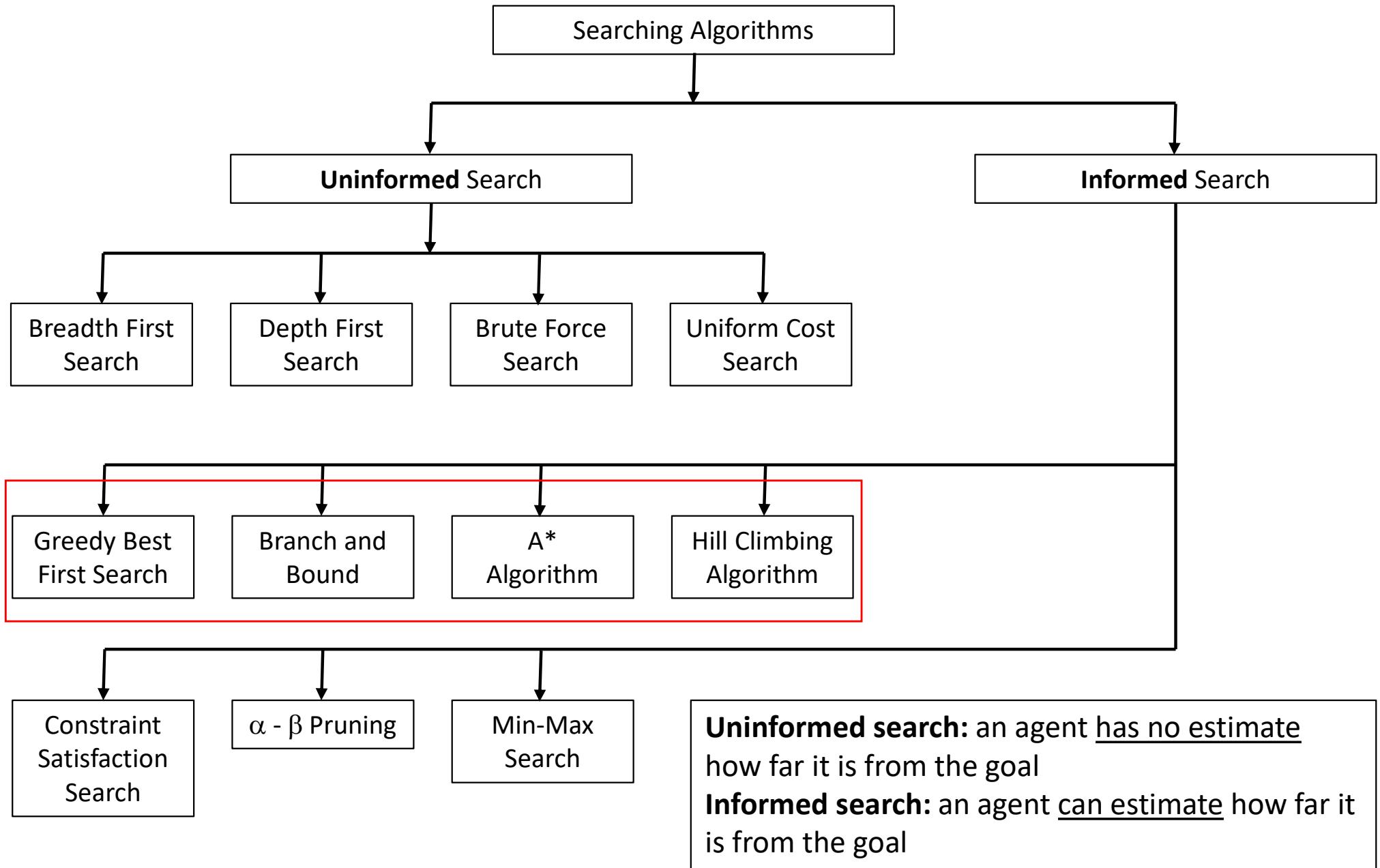
Informed search relies on **domain-specific knowledge / hints** that help locate the goal state.

$$h(n) = h(\text{State } n)$$

$$h(n) = n(\text{relevant information about State } n)$$

$h(n)$: heuristic function - estimated cost of the cheapest path from State n to the goal state

Selected Searching Algorithms



Informed Search: the Idea

When traversing the search tree **use domain knowledge / heuristics to avoid search paths that are likely to be fruitless**

Informed Search and Heuristics

Informed search relies on **domain-specific knowledge / hints** that help locate the goal state.

$$h(n) = h(\text{State } n)$$

$$h(n) = n(\text{relevant information about State } n)$$

$h(n)$: heuristic function - estimated cost of the cheapest path from State n to the goal state

Evaluation function

Calculate / obtain:

$$f(n) = f(\text{State } n)$$

$$f(n) = f(\text{relevant information about State } n)$$

A state n with minimum (or maximum) $f(n)$ should be chosen for expansion

What about ties?

Informed Search and Heuristics

Informed search relies on **domain-specific knowledge / hints** that help locate the goal state.

$$h(n) = h(\text{State } n)$$

$h(n)$ = n (relevant information about State n)

$h(n)$: heuristic function - estimated cost of the cheapest path from State n to the goal state

Hill Climbing Search

- The most primitive informed search approach
 - a naive greedy algorithm
 - evaluation function: the cost of next move
 - does not care about the “bigger picture” (for example: total search path cost)
- Practicalities:
 - usually does not keep track of search history:
 - not tracking visited nodes → loops!
 - not tracking frontier nodes → does not look at alternatives

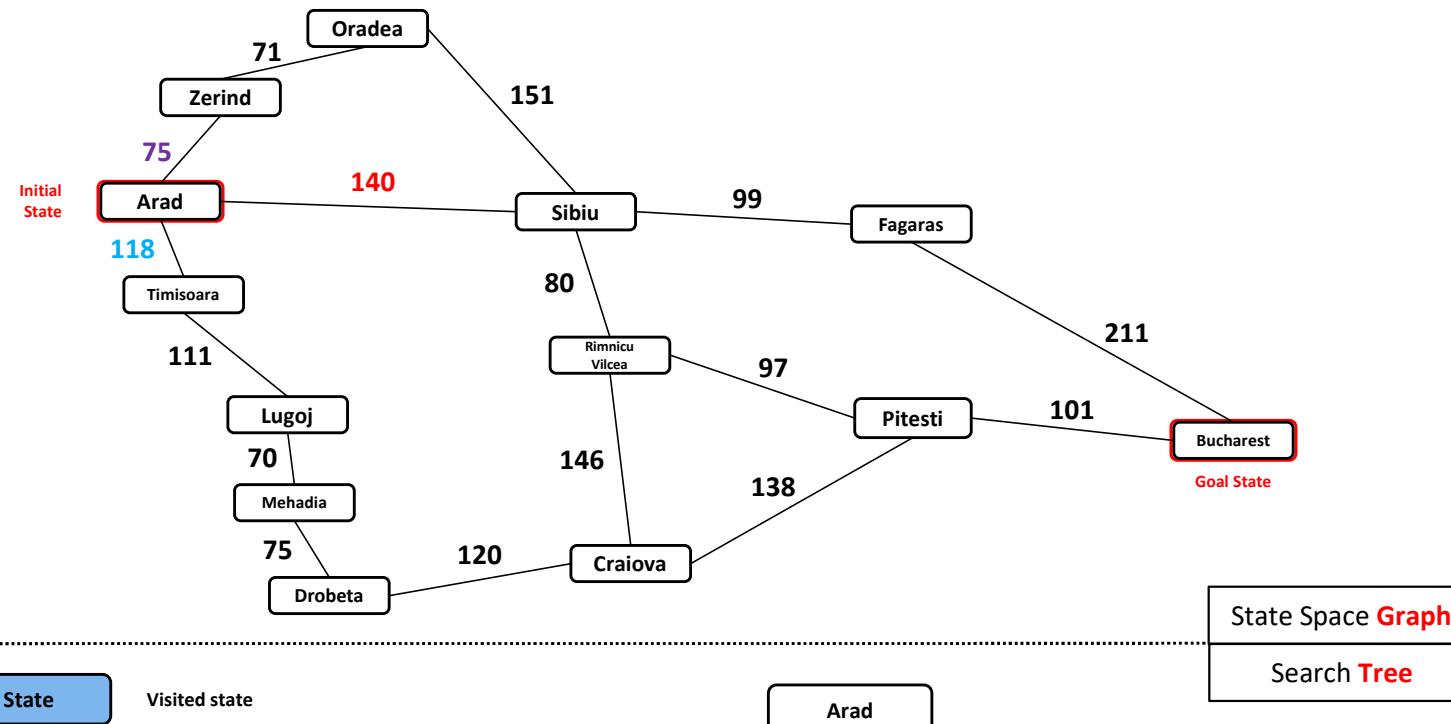
Hill Climbing: Evaluation function

Calculate / obtain:

$$f(n) = \text{ACTION-COST}(\text{State}_a, \text{toState}_n, \text{State}_n)$$

A state n with minimum (or maximum) $f(n)$ should be chosen for expansion

Romanian Roadtrip: Hill Climbing

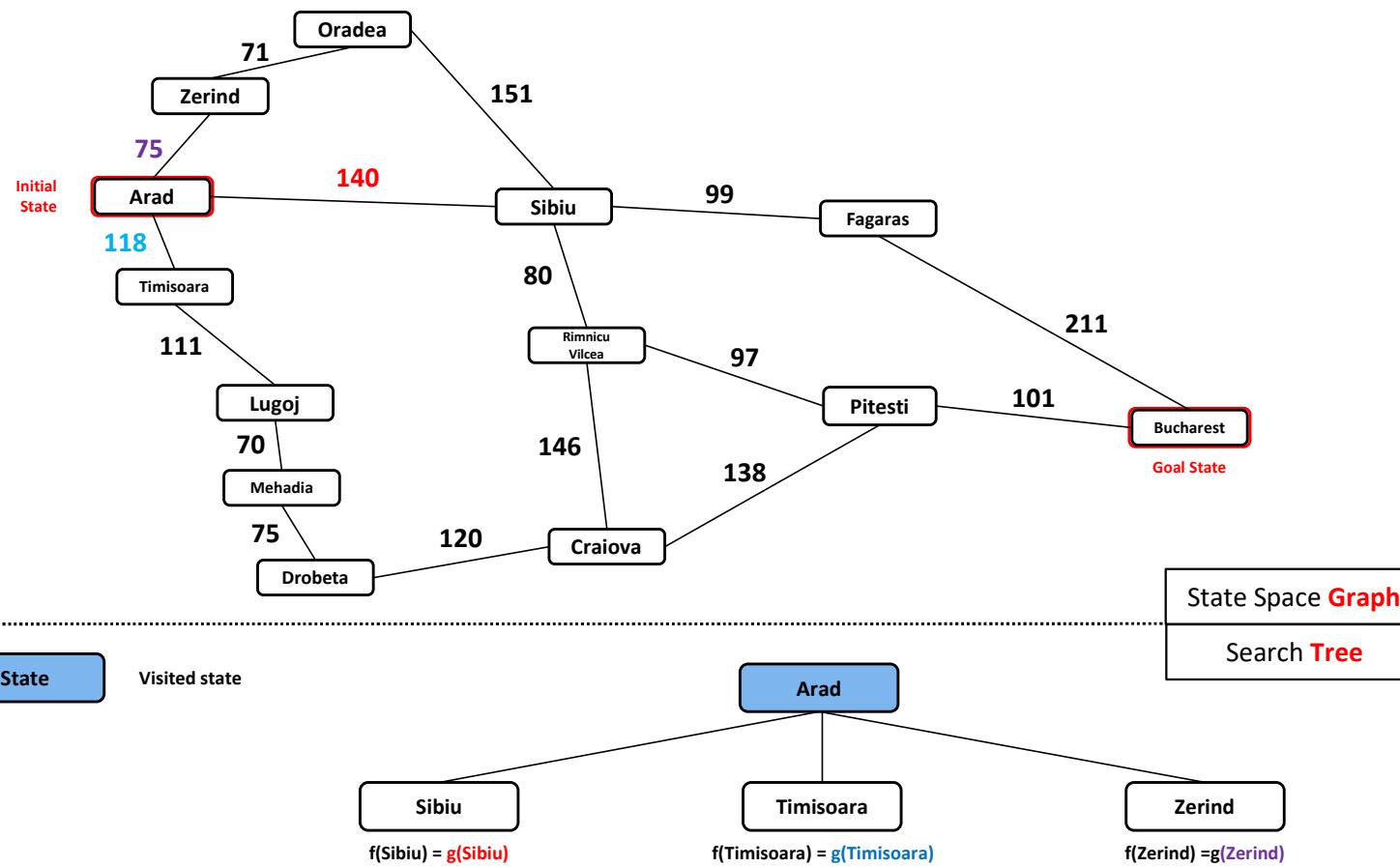


Assumption:
We don't "go" to a
repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

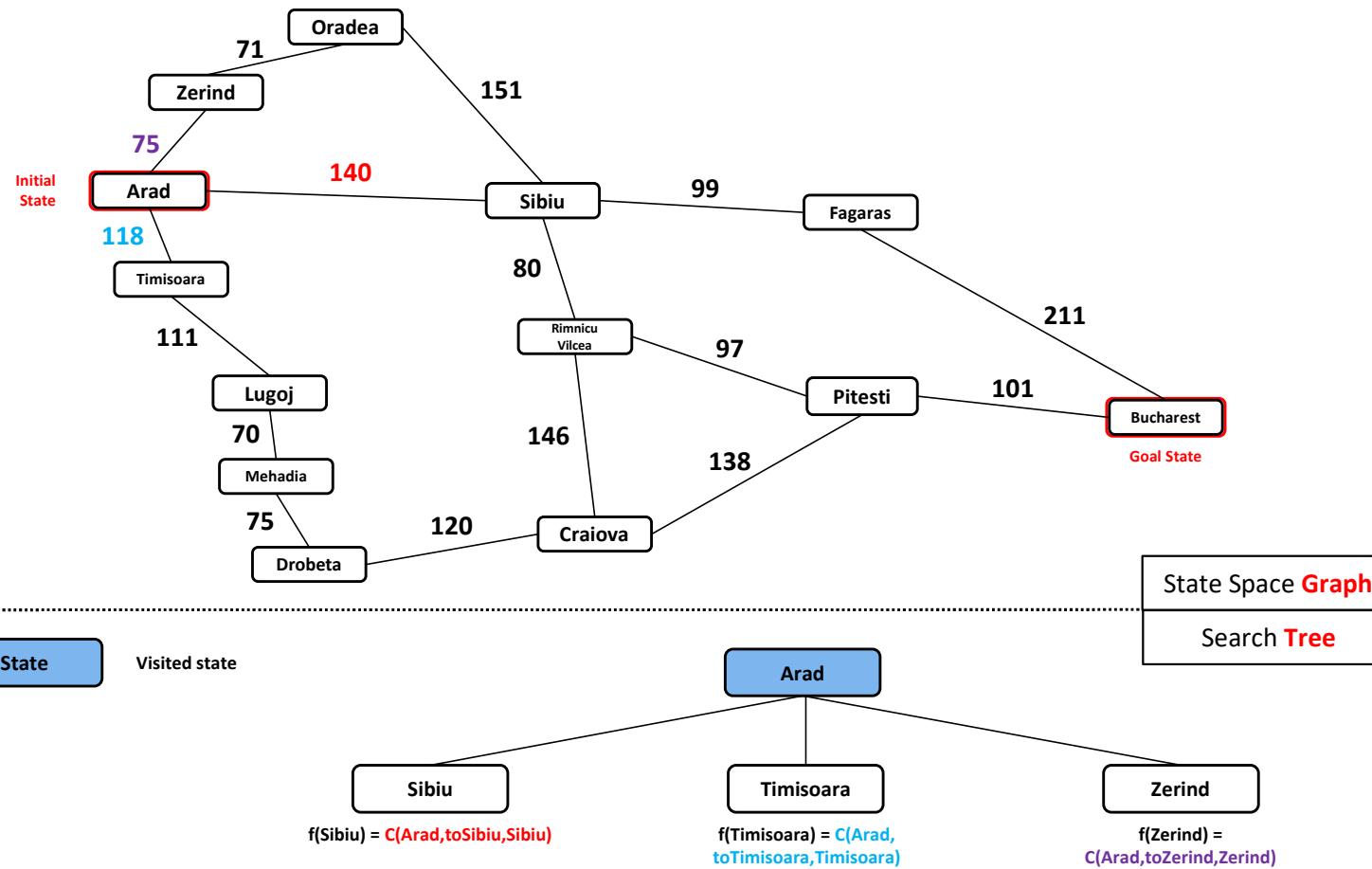


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

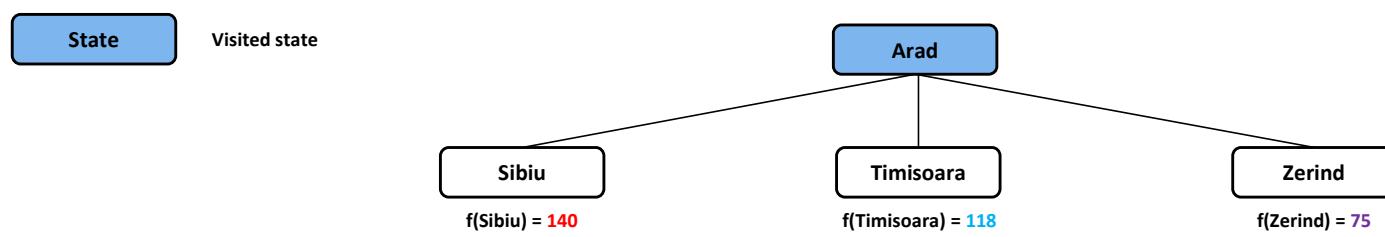
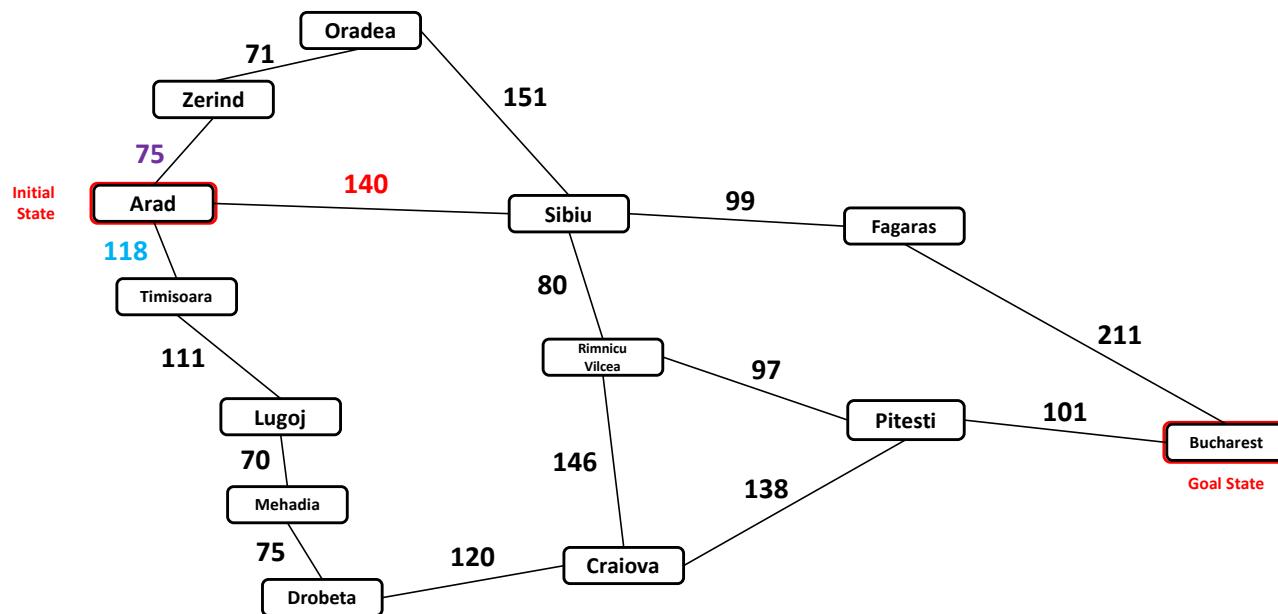


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state and
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

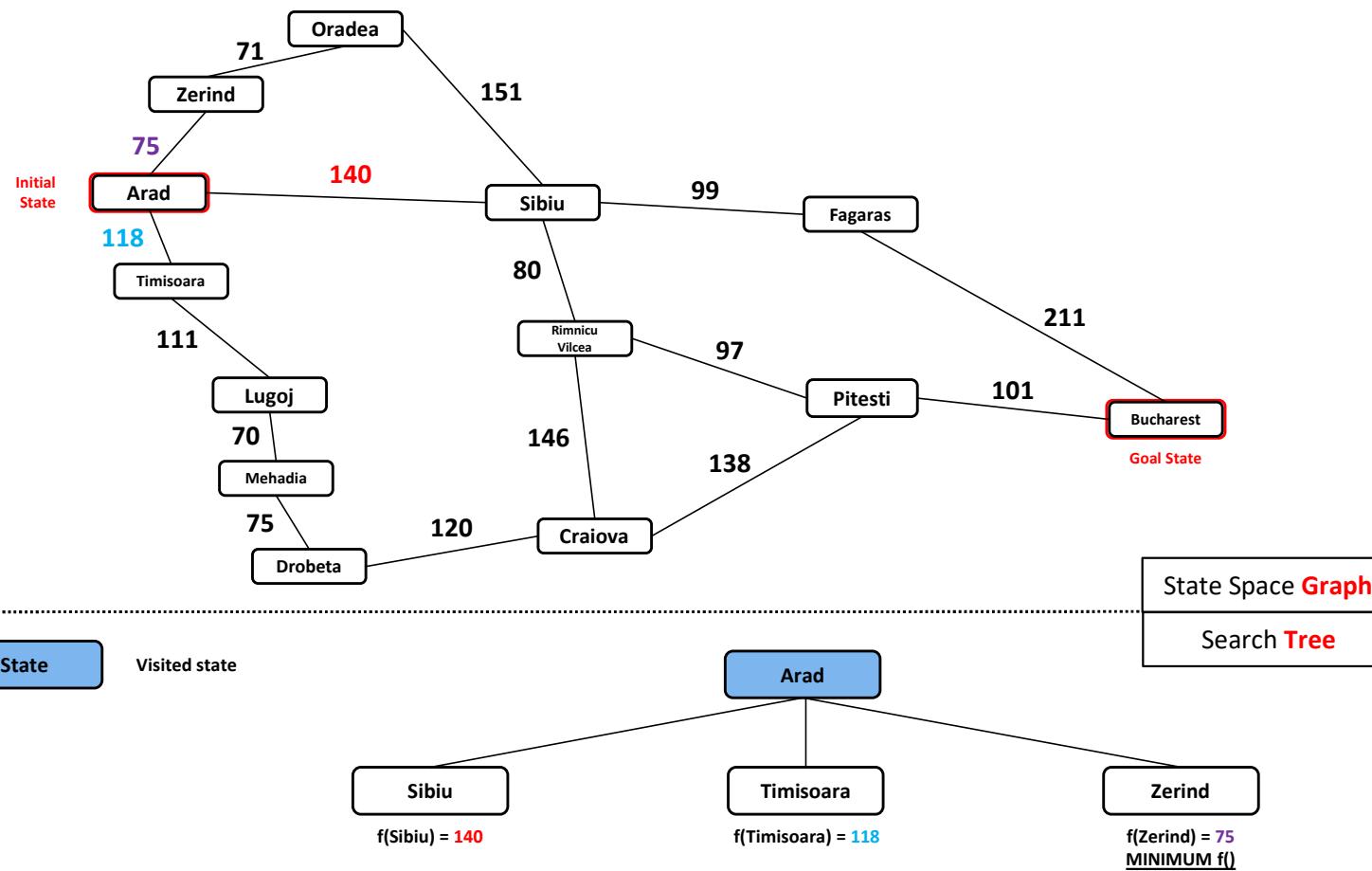


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state and get stuck there or - Infinite loop

Romanian Roadtrip: Hill Climbing

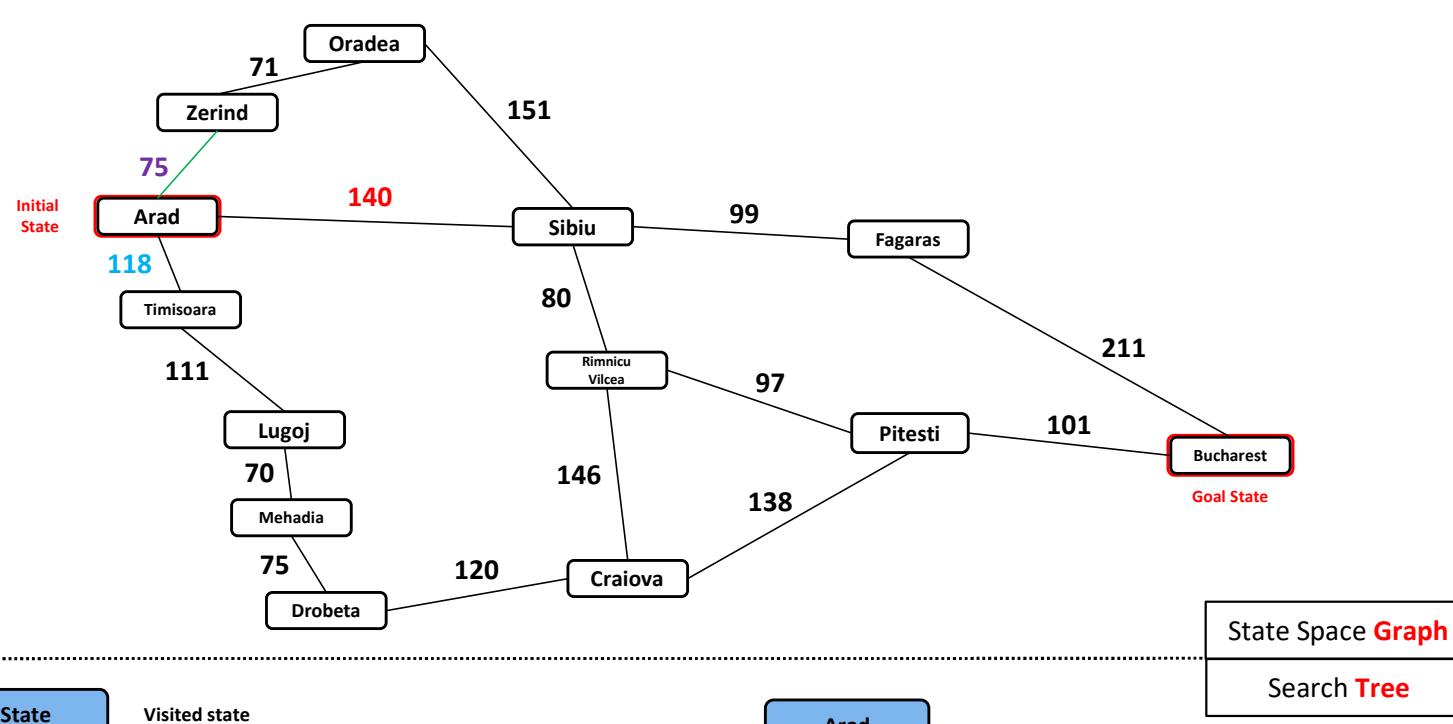


Assumption:
We don't "go" to a repeated state

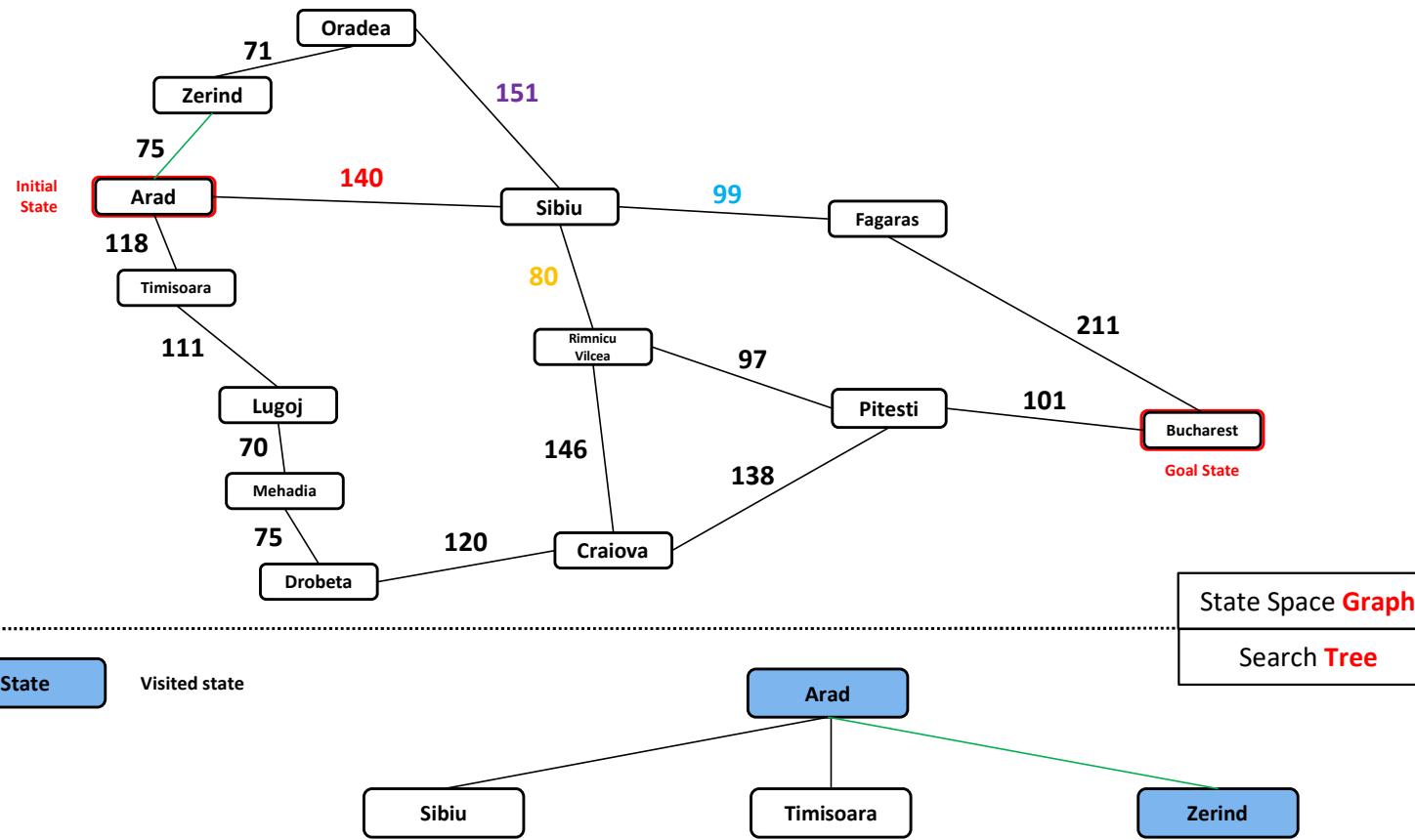
Alternatively:

- We could go to a repeated state end
 - get stuck there or
 - Infinite loop

Romanian Roadtrip: Hill Climbing



Romanian Roadtrip: Hill Climbing

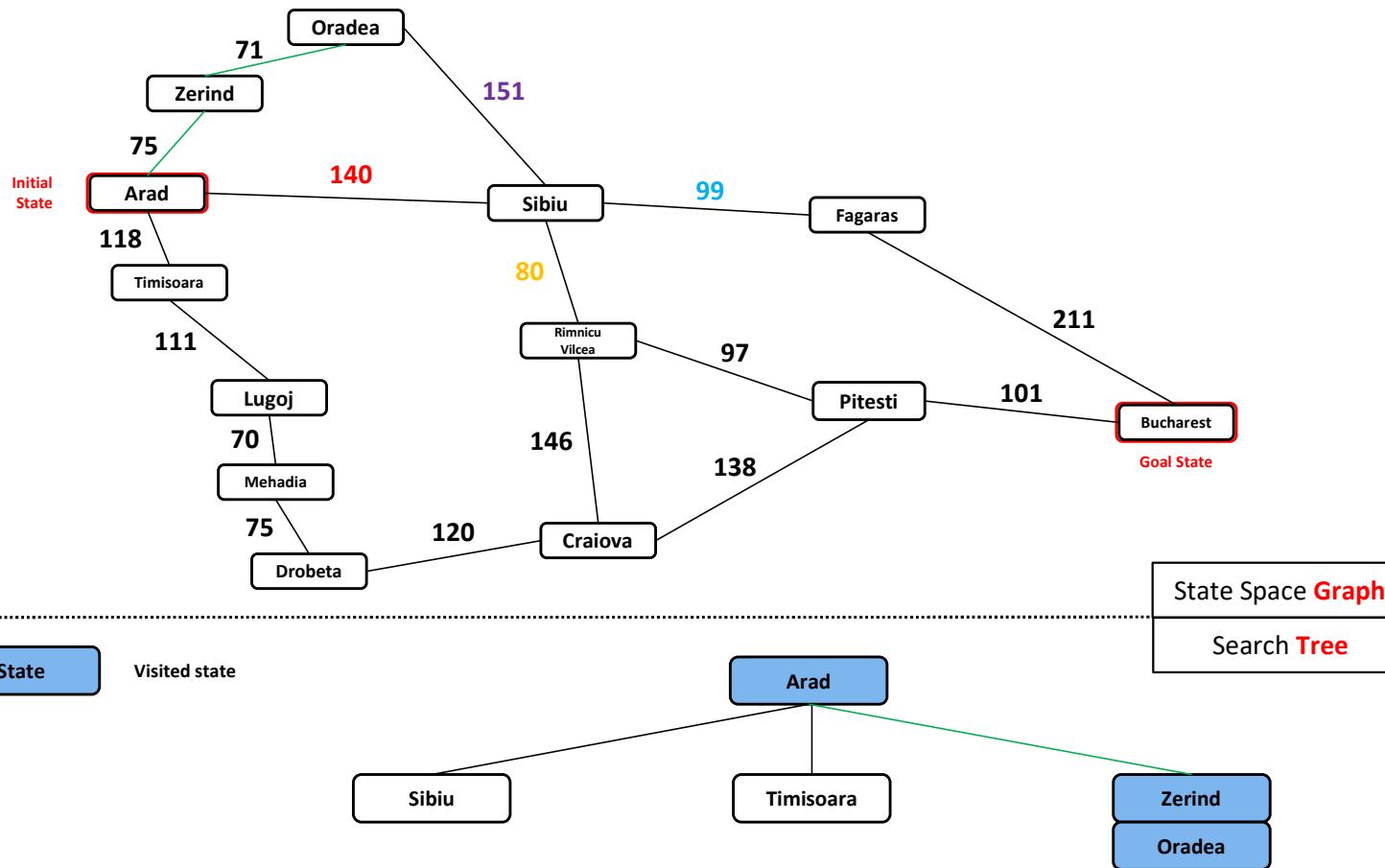


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

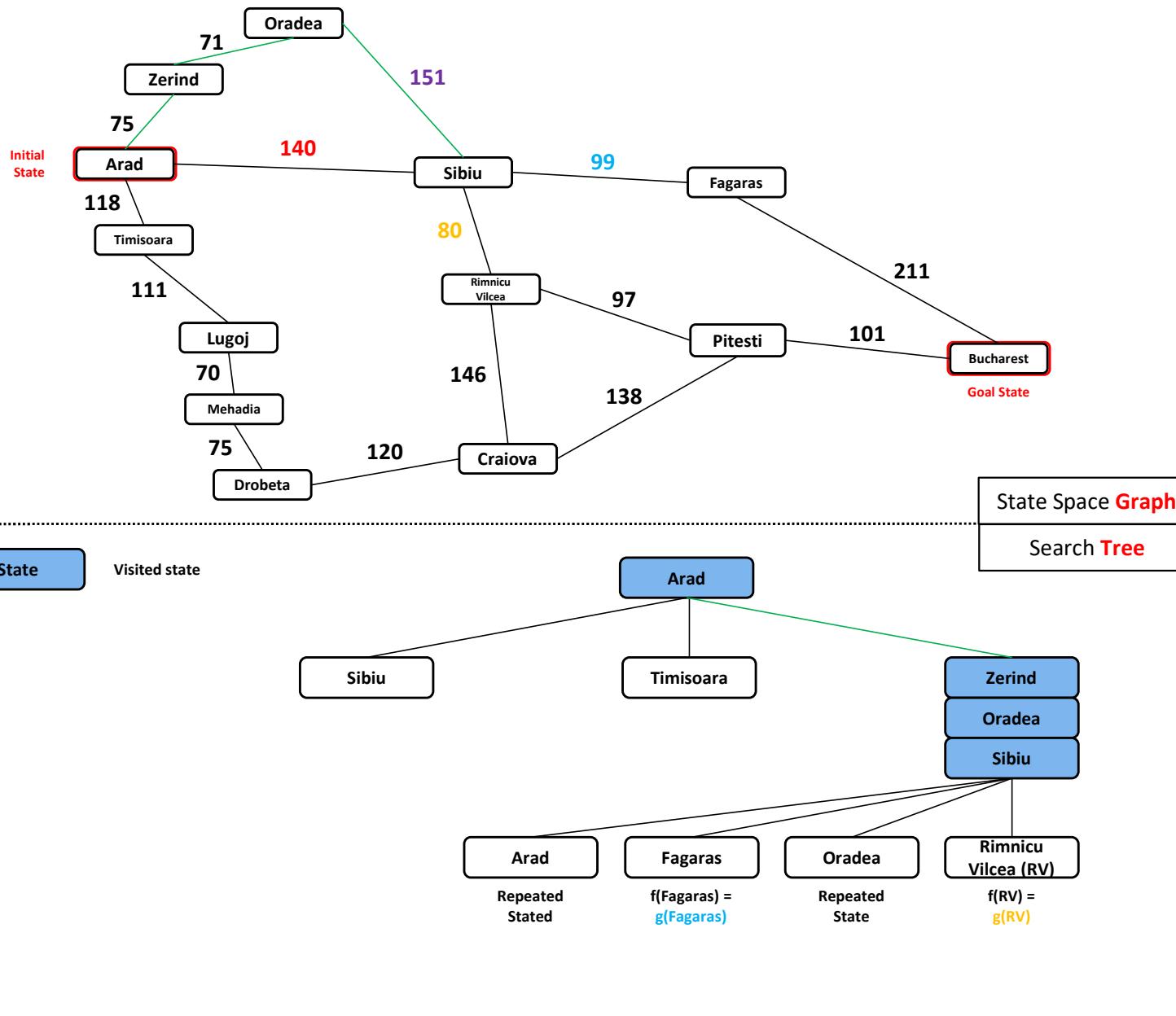


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state and
- get stuck there
or
- Infinite loop

Romanian Roadtrip: Hill Climbing

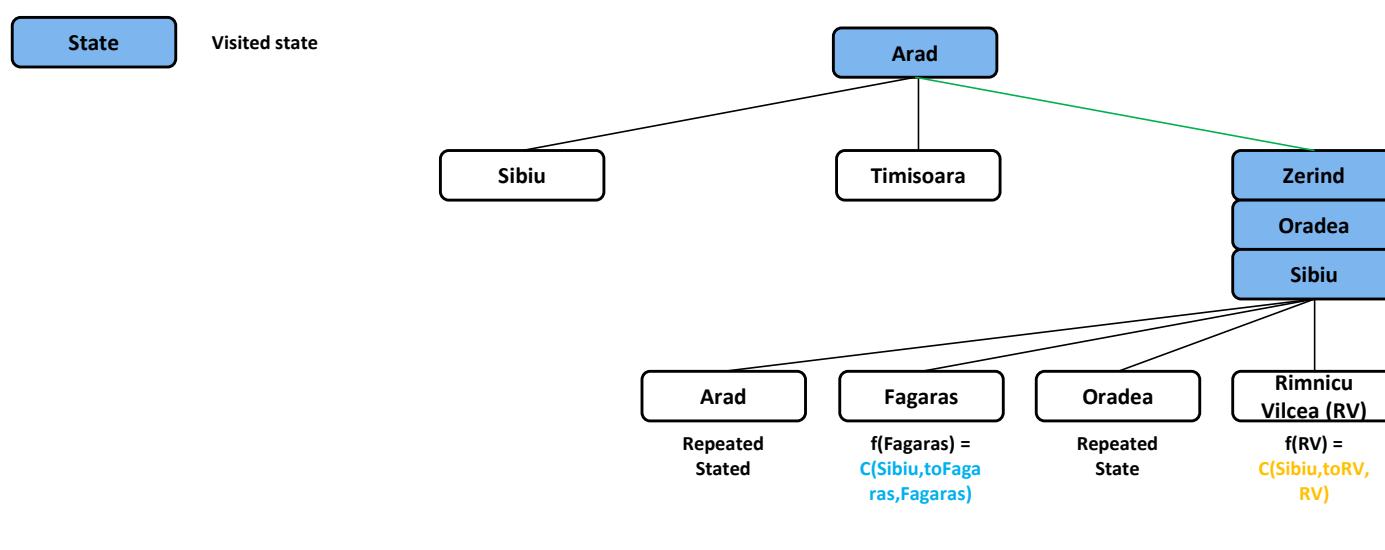
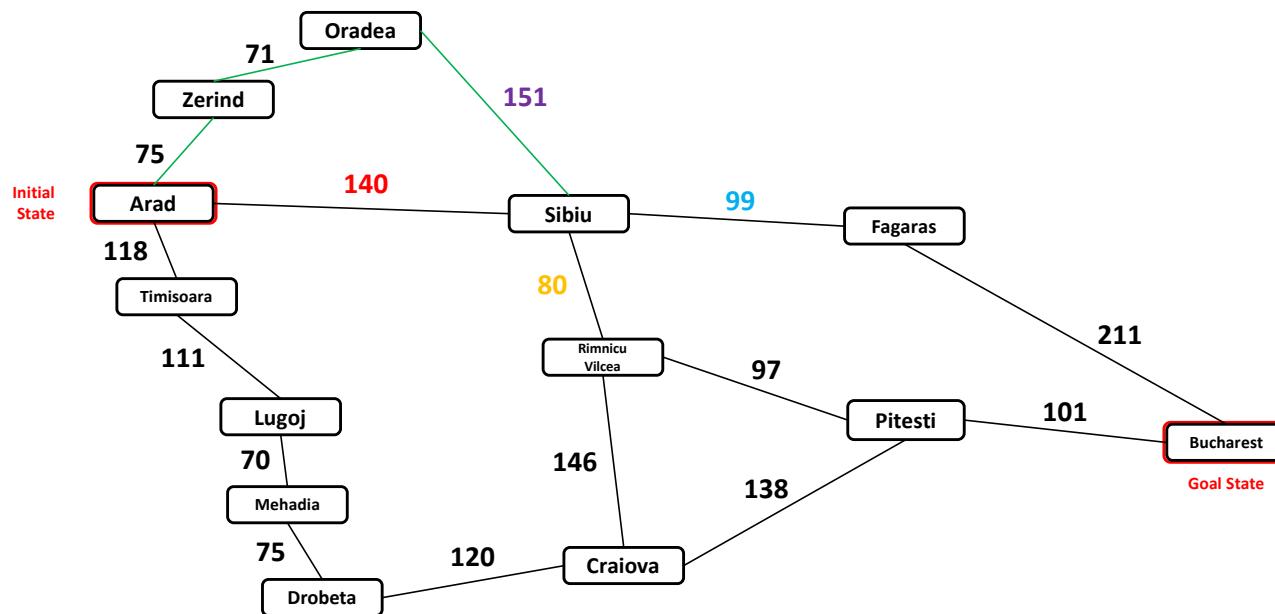


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

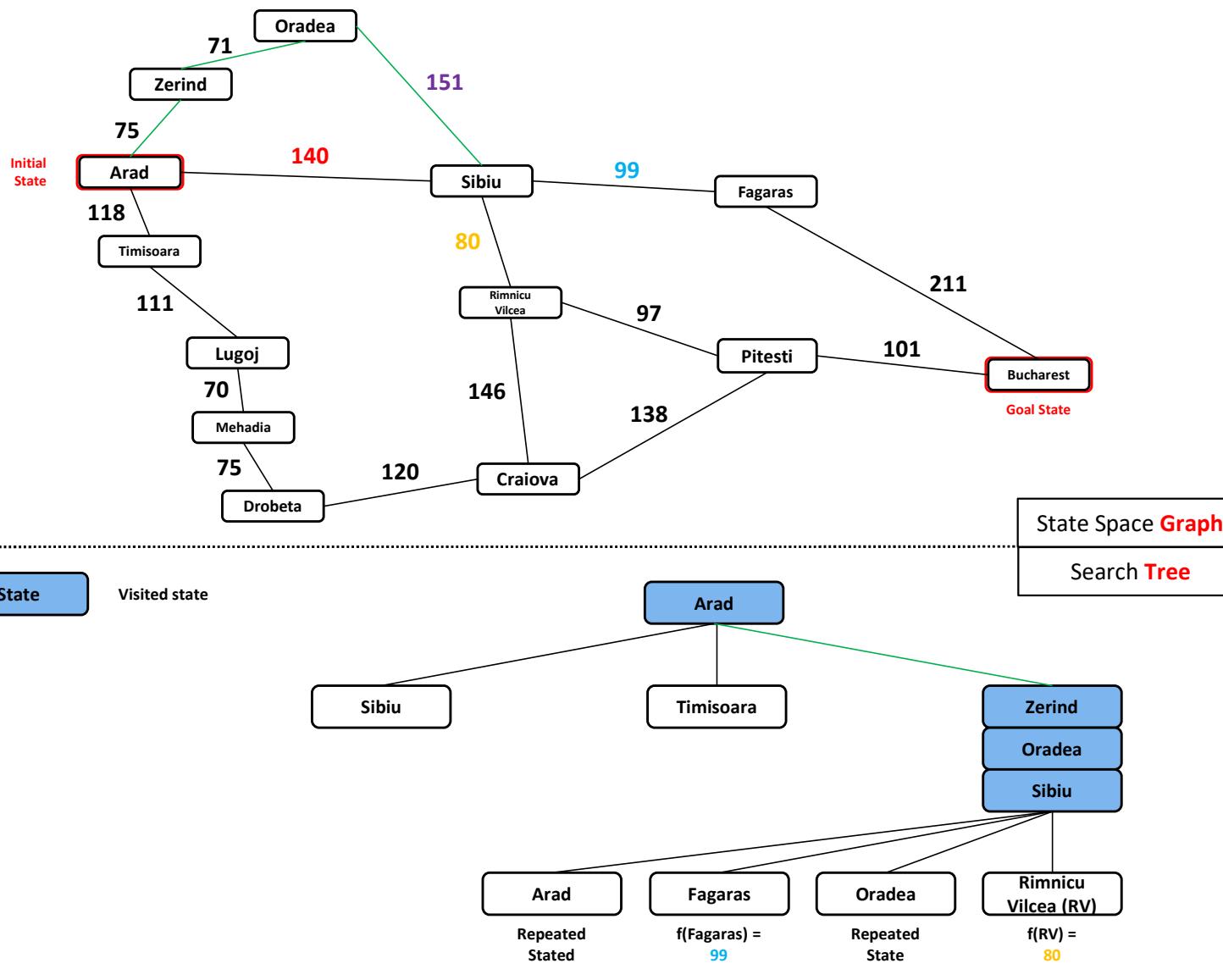


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

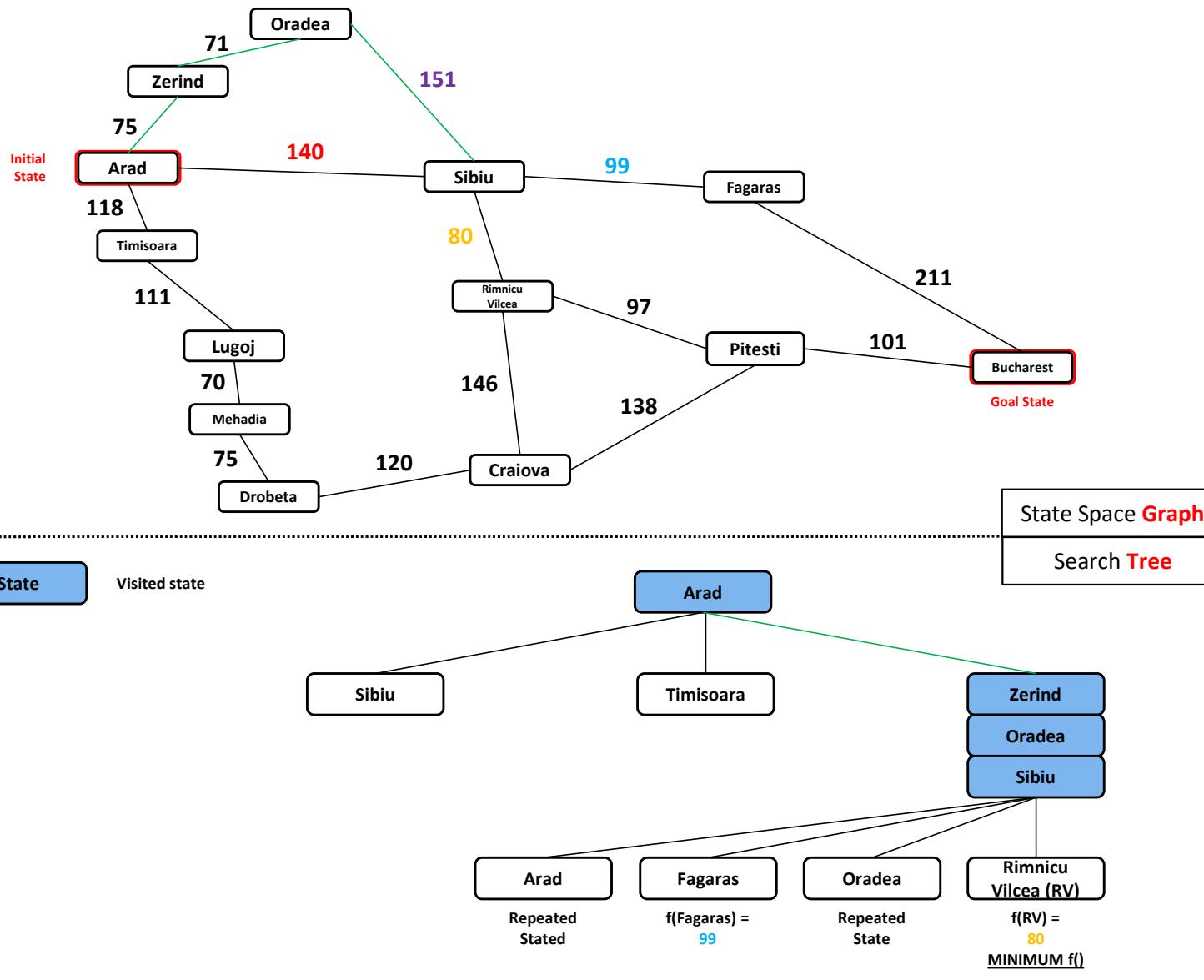


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

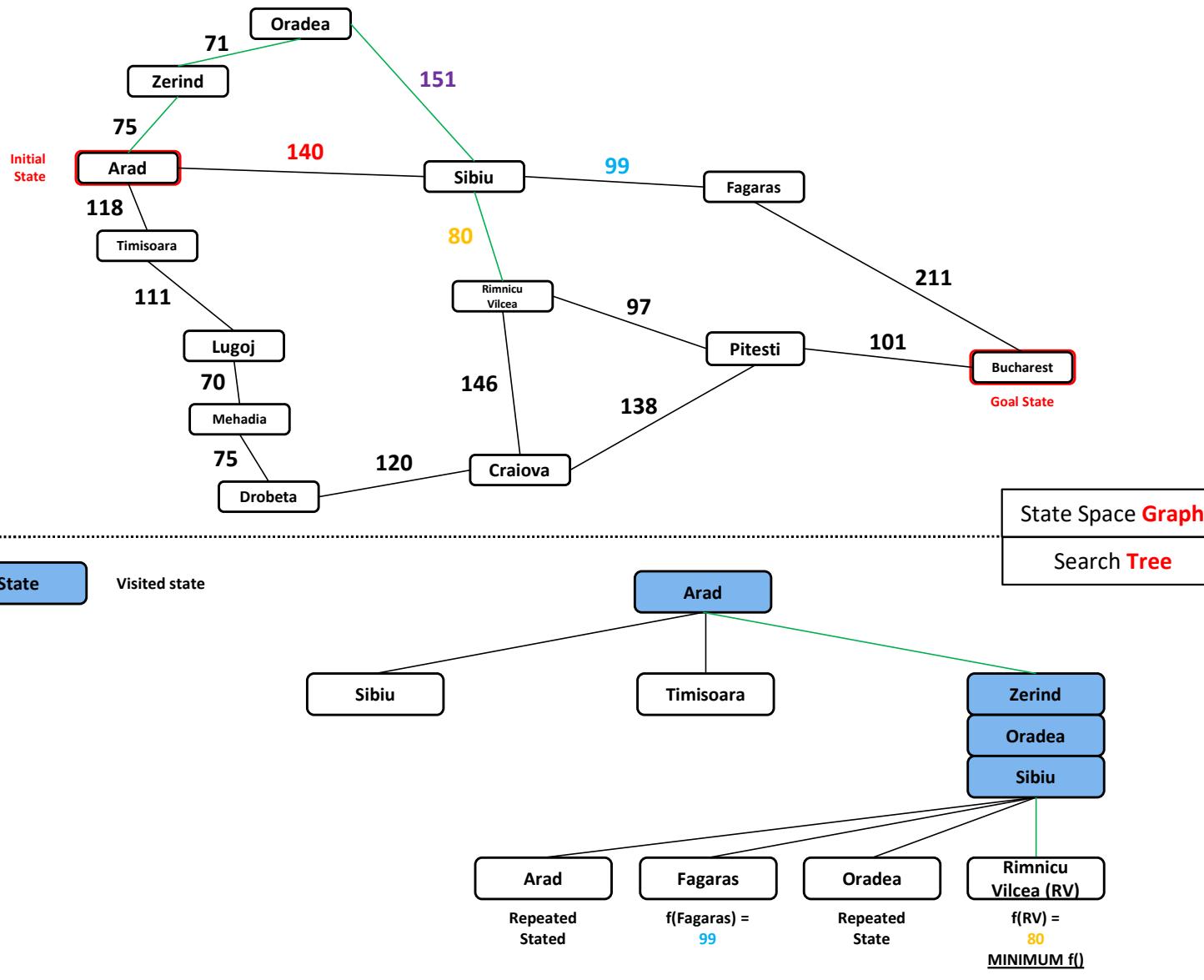


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

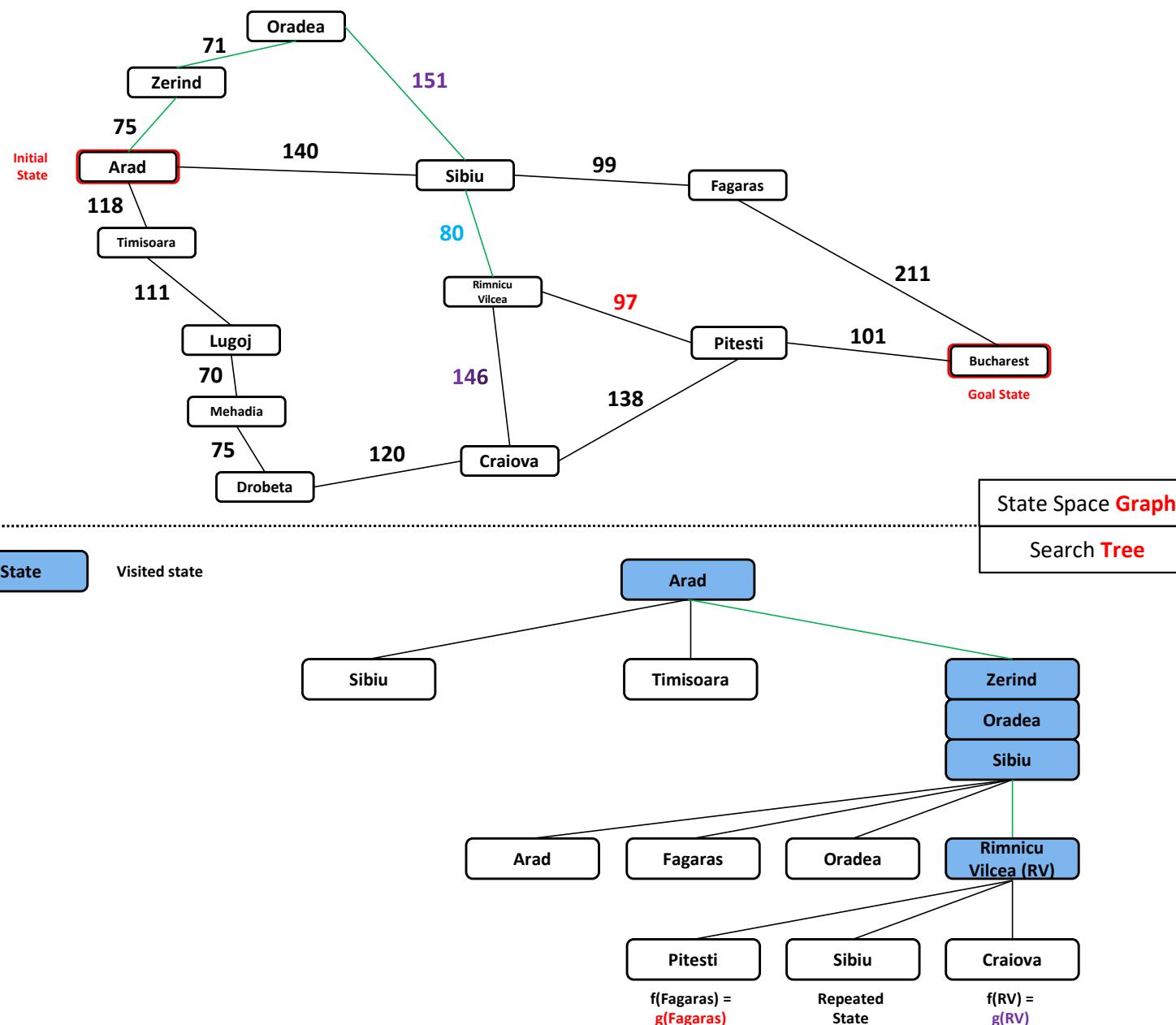


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
 - get stuck there
 - or
 - Infinite loop

Romanian Roadtrip: Hill Climbing

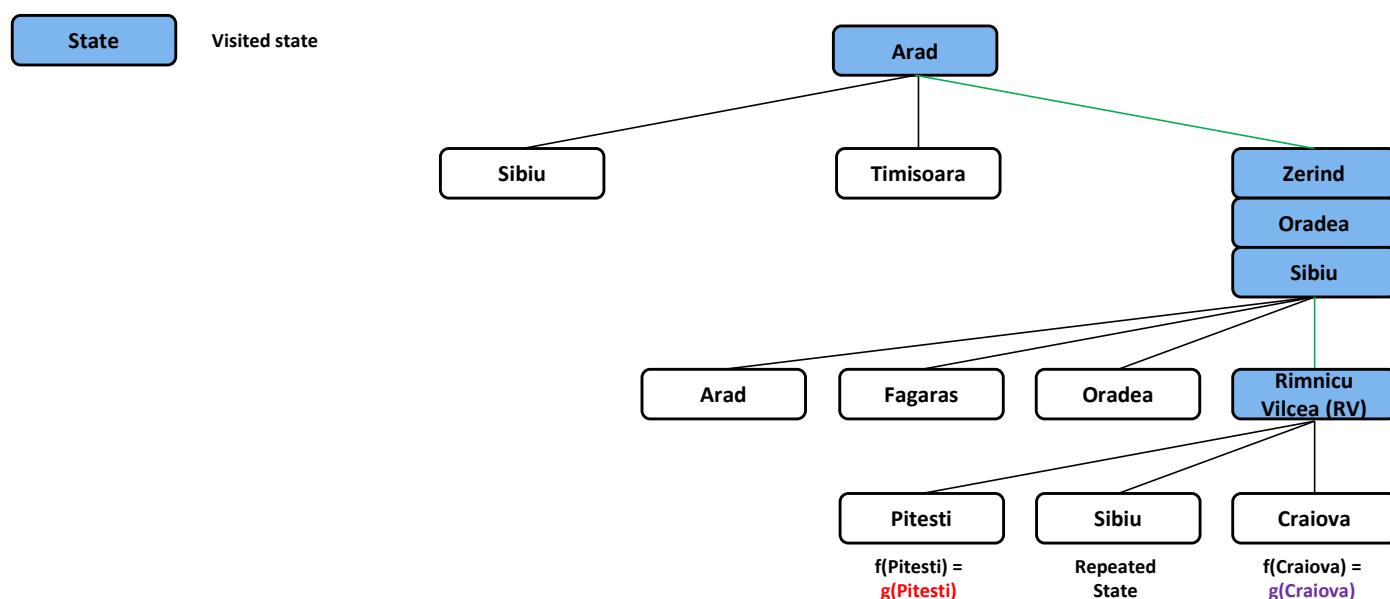
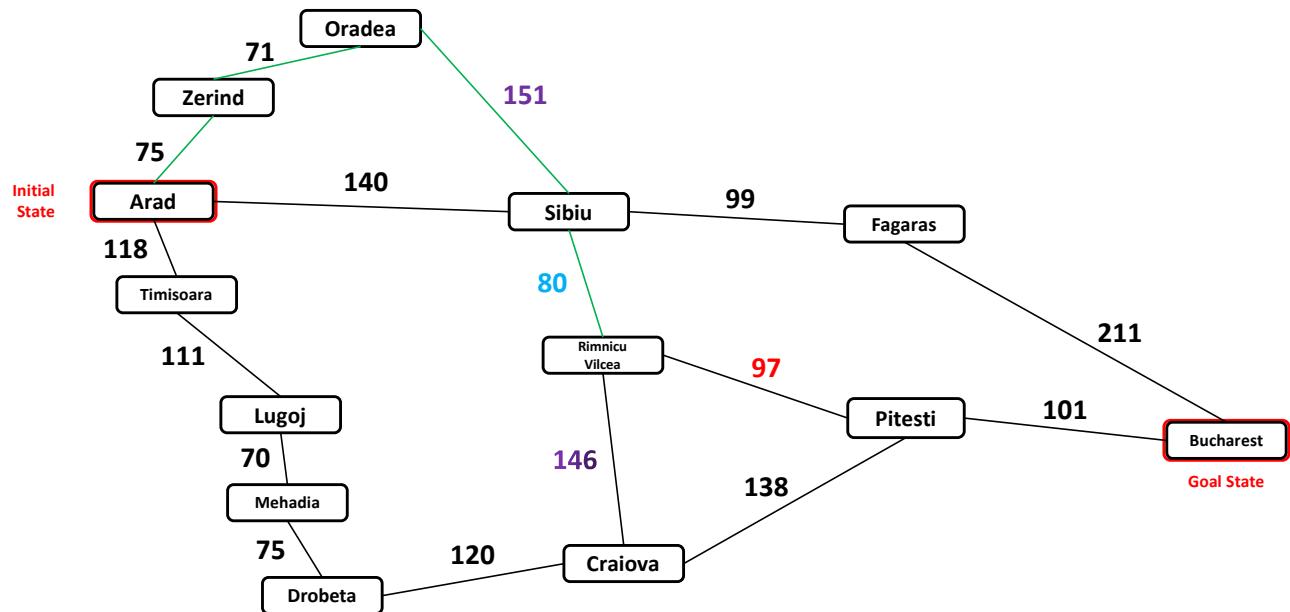


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

Romanian Roadtrip: Hill Climbing

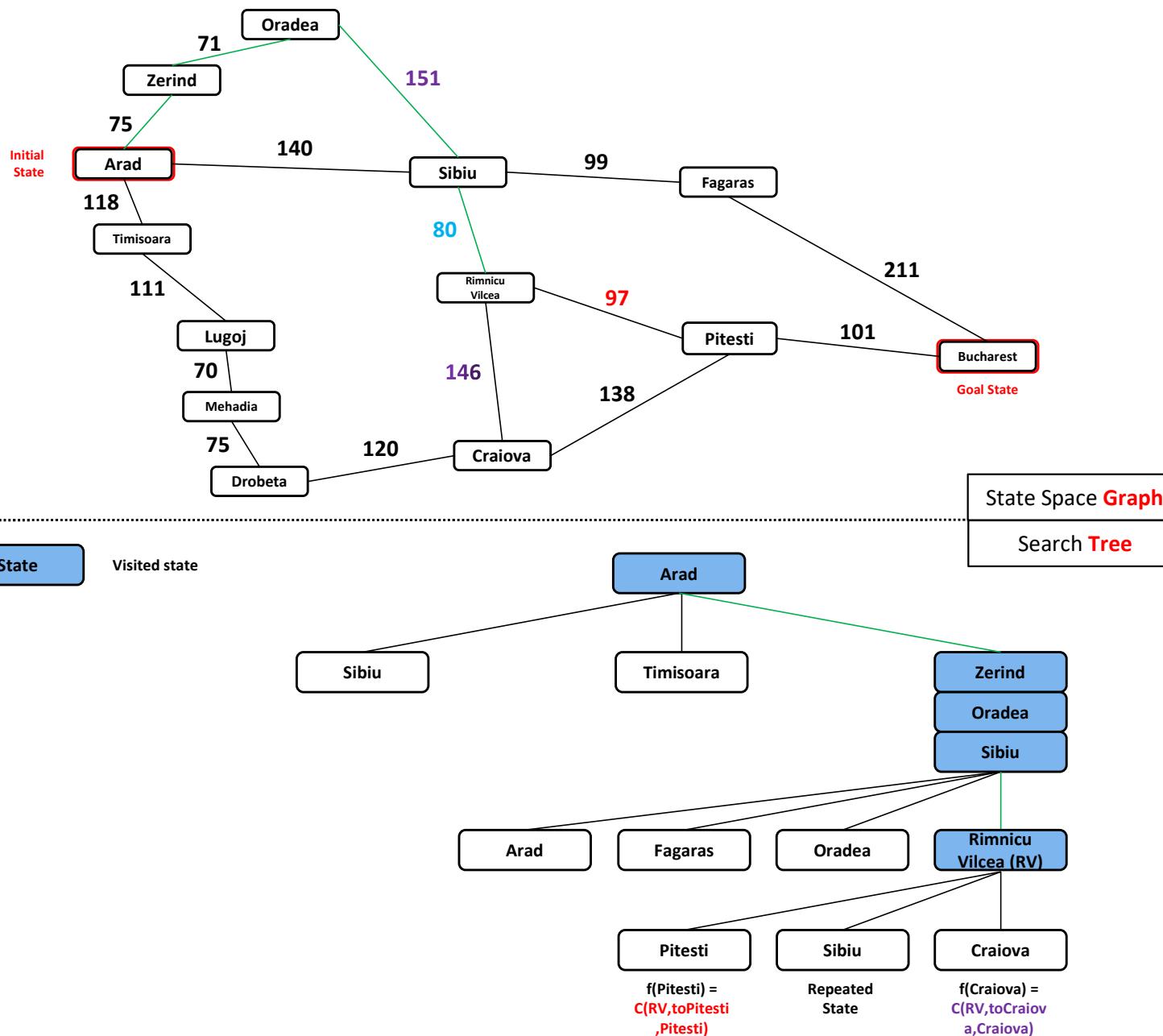


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

Romanian Roadtrip: Hill Climbing

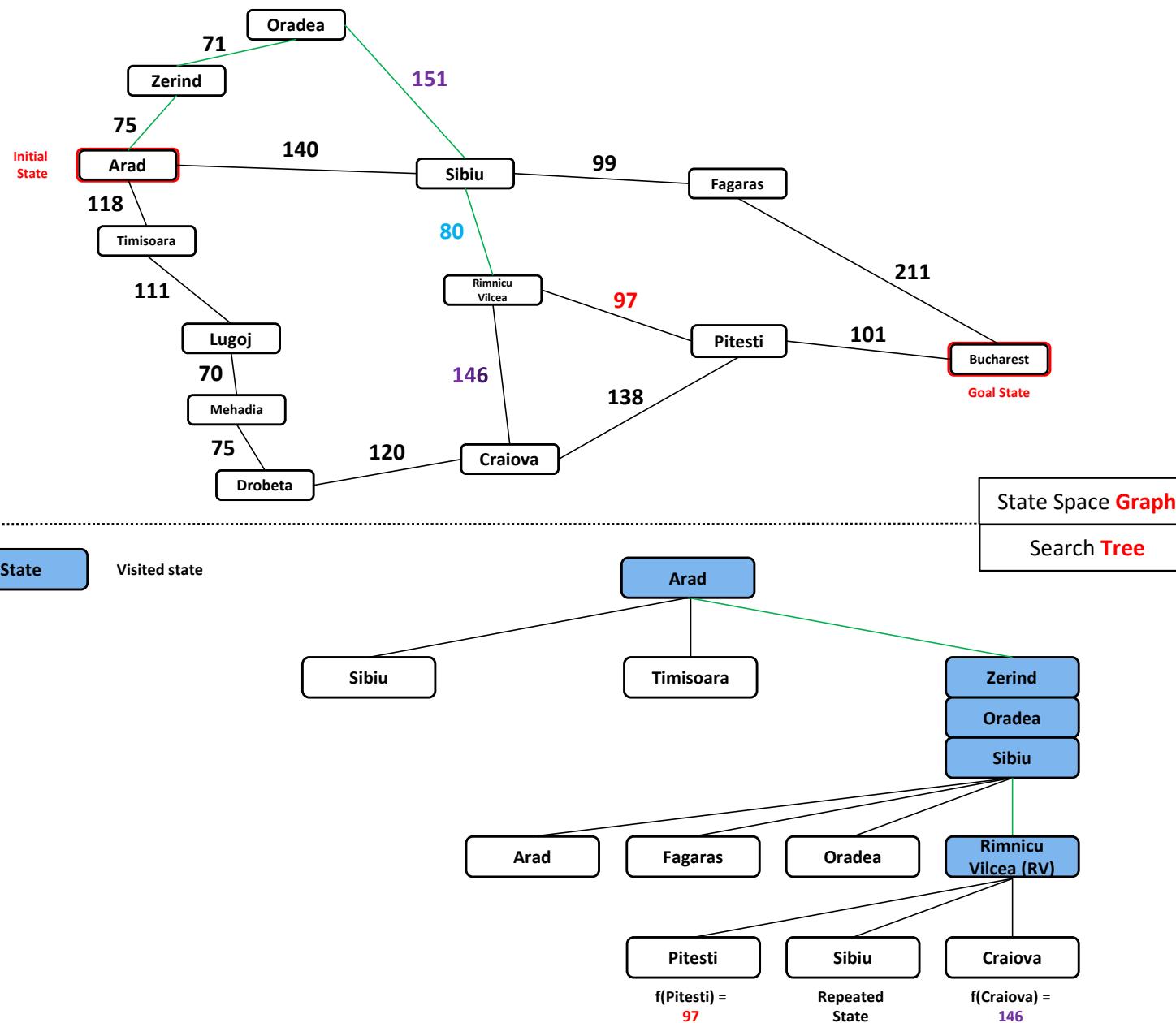


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

Romanian Roadtrip: Hill Climbing

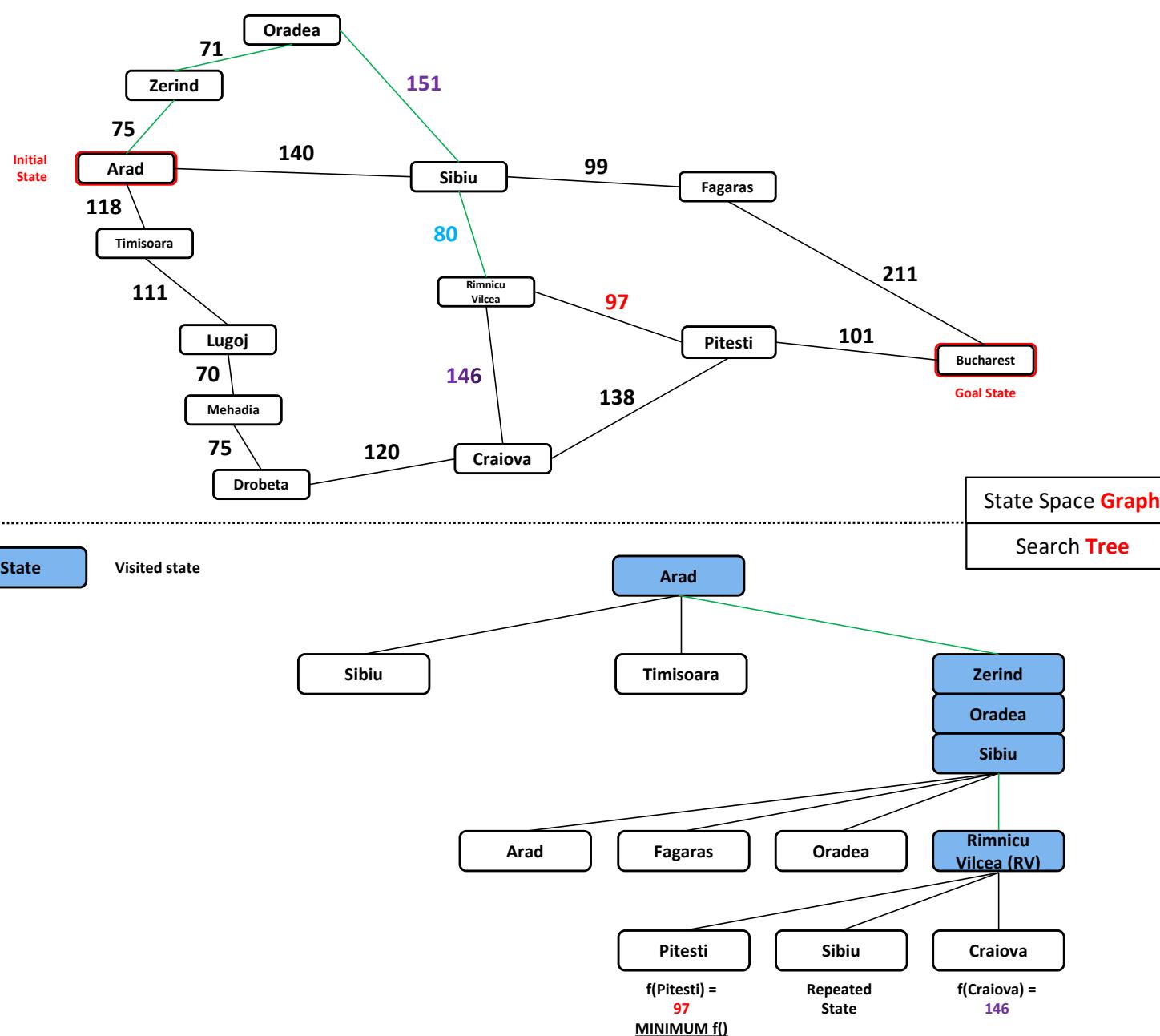


Assumption:
We don't "go" to a repeated state

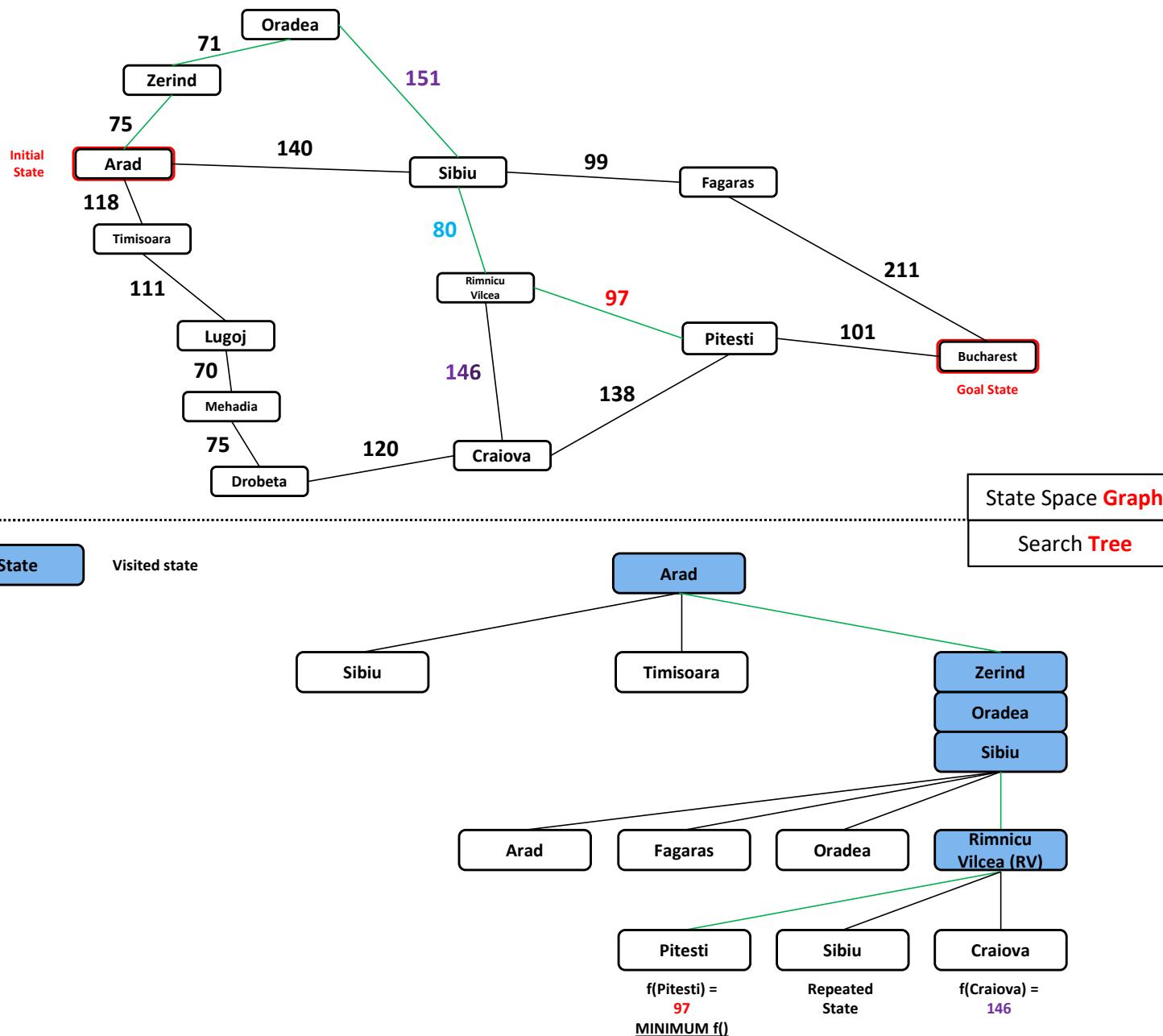
Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

Romanian Roadtrip: Hill Climbing



Romanian Roadtrip: Hill Climbing

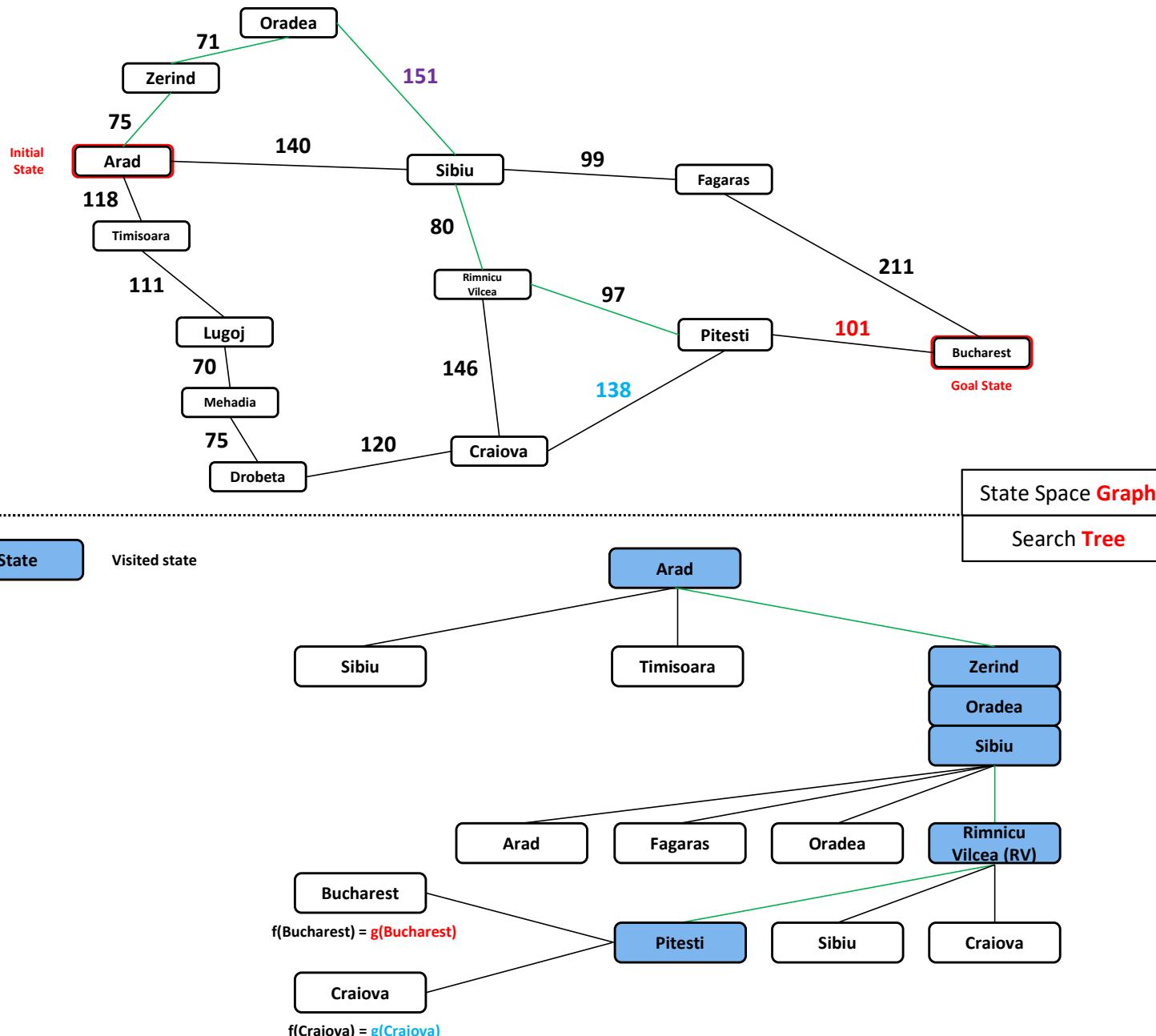


Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

Romanian Roadtrip: Hill Climbing

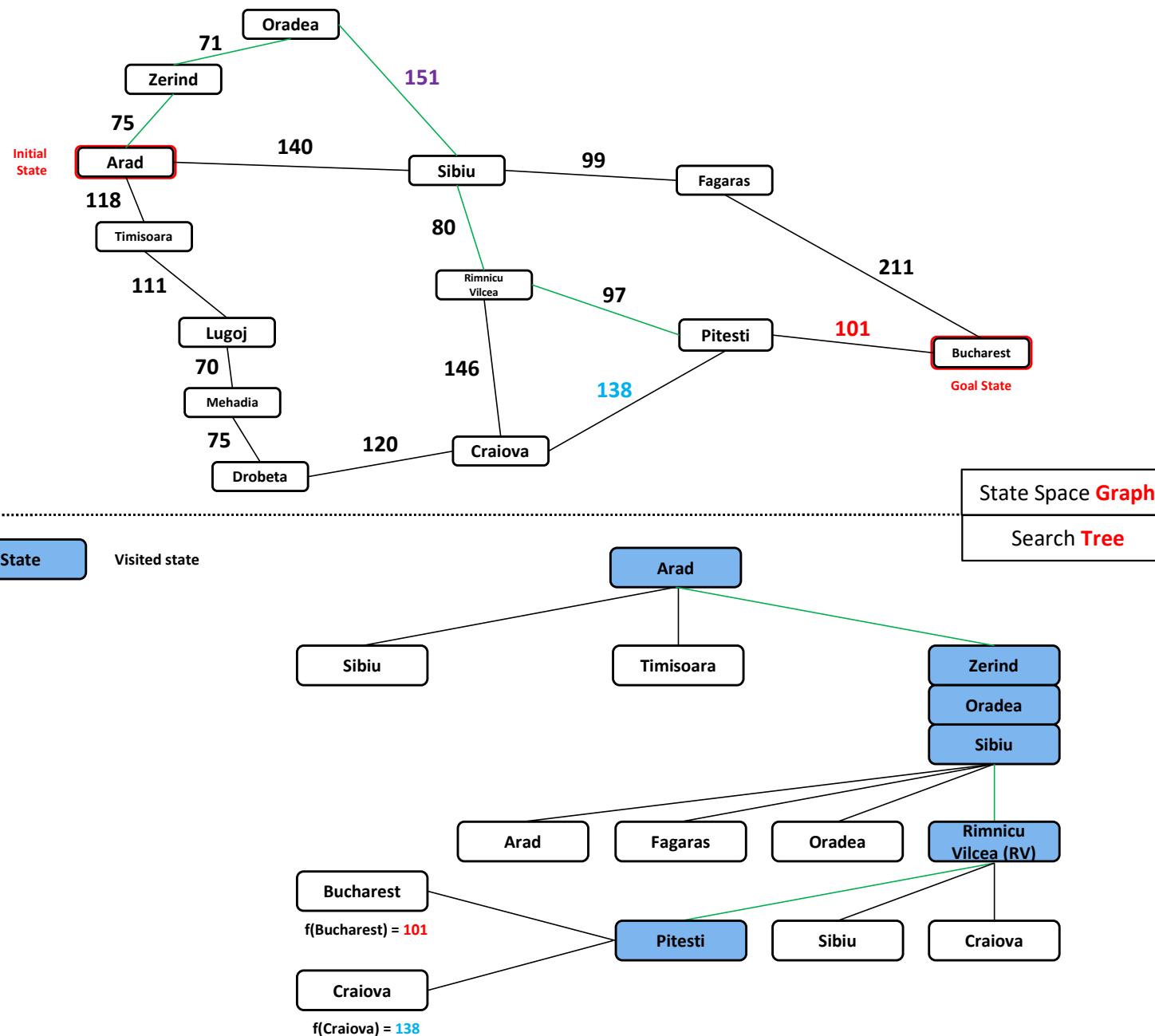


Assumption:
We don't "go" to a repeated state

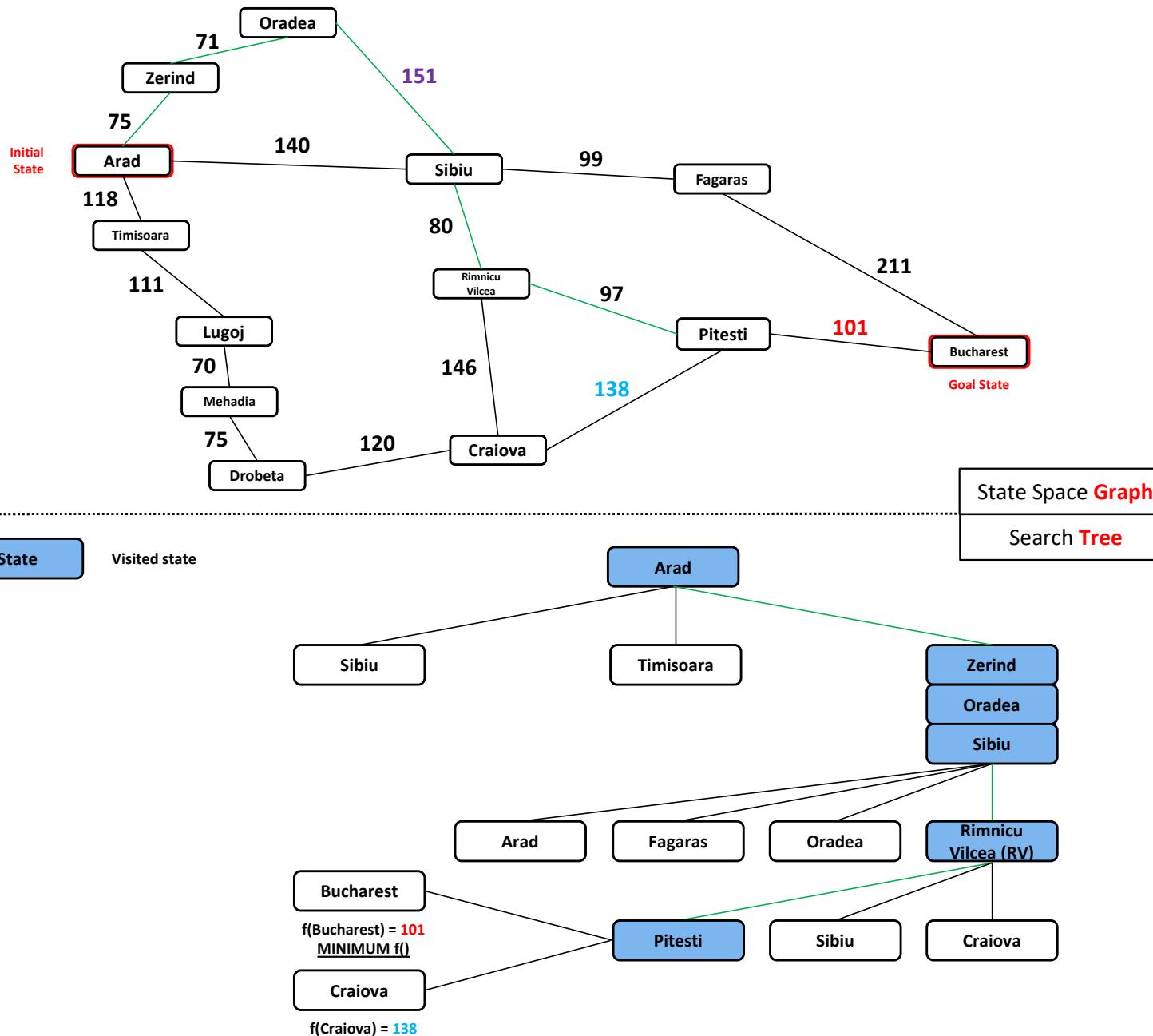
Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

Romanian Roadtrip: Hill Climbing



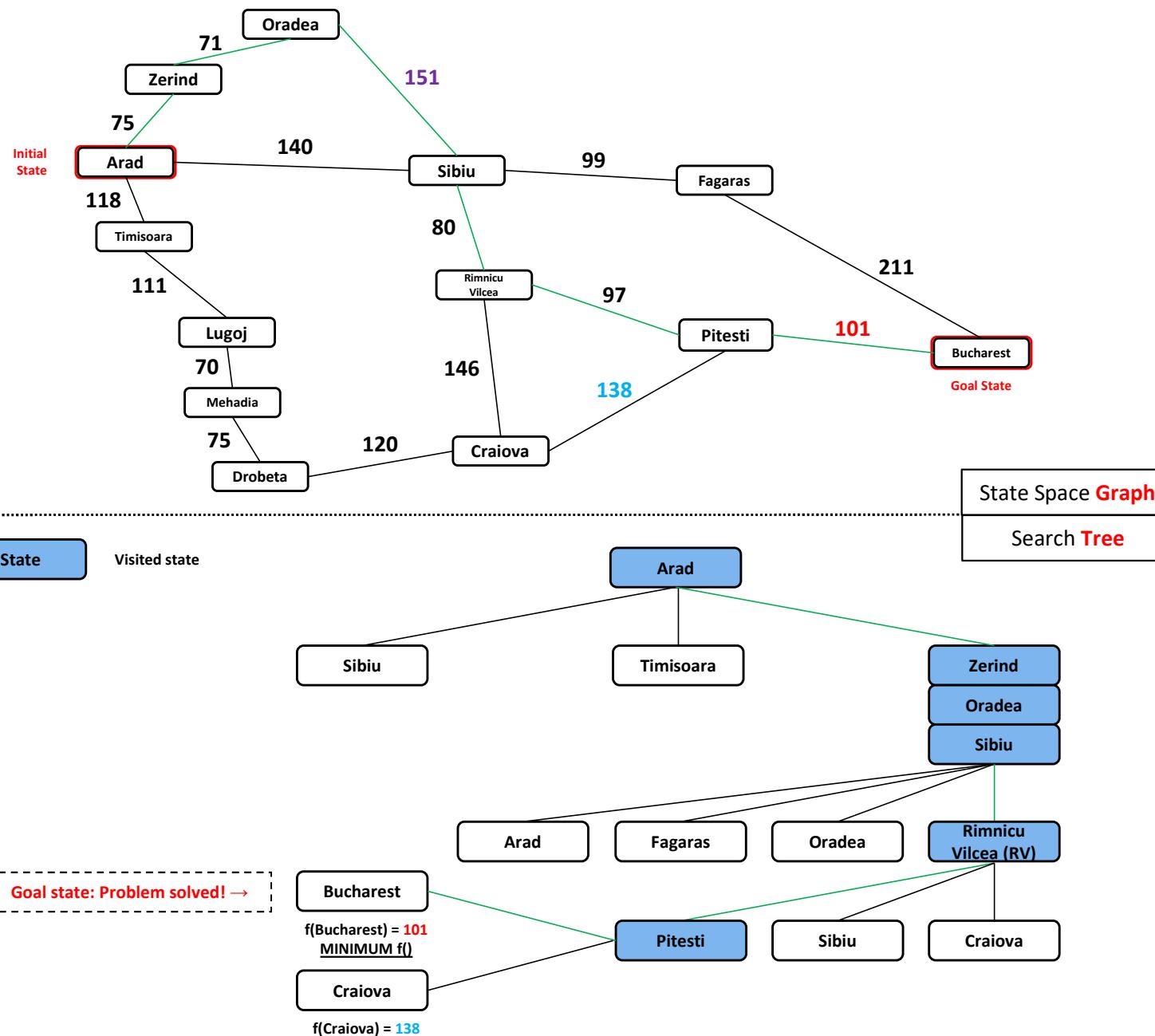
Romanian Roadtrip: Hill Climbing



Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

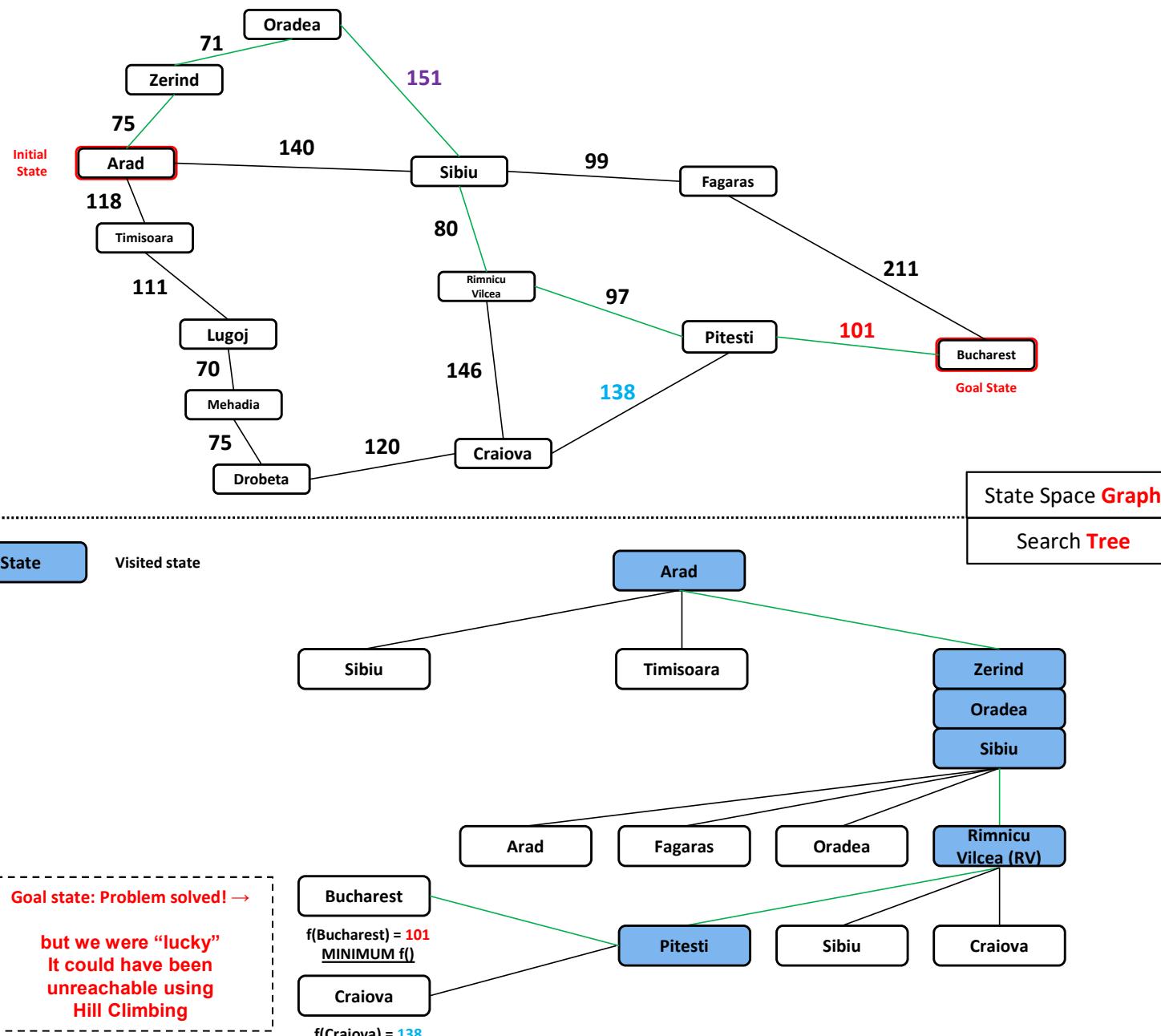
Romanian Roadtrip: Hill Climbing



Alternatively:

- We could go to a repeated state end
- get stuck there
or
- Infinite loop

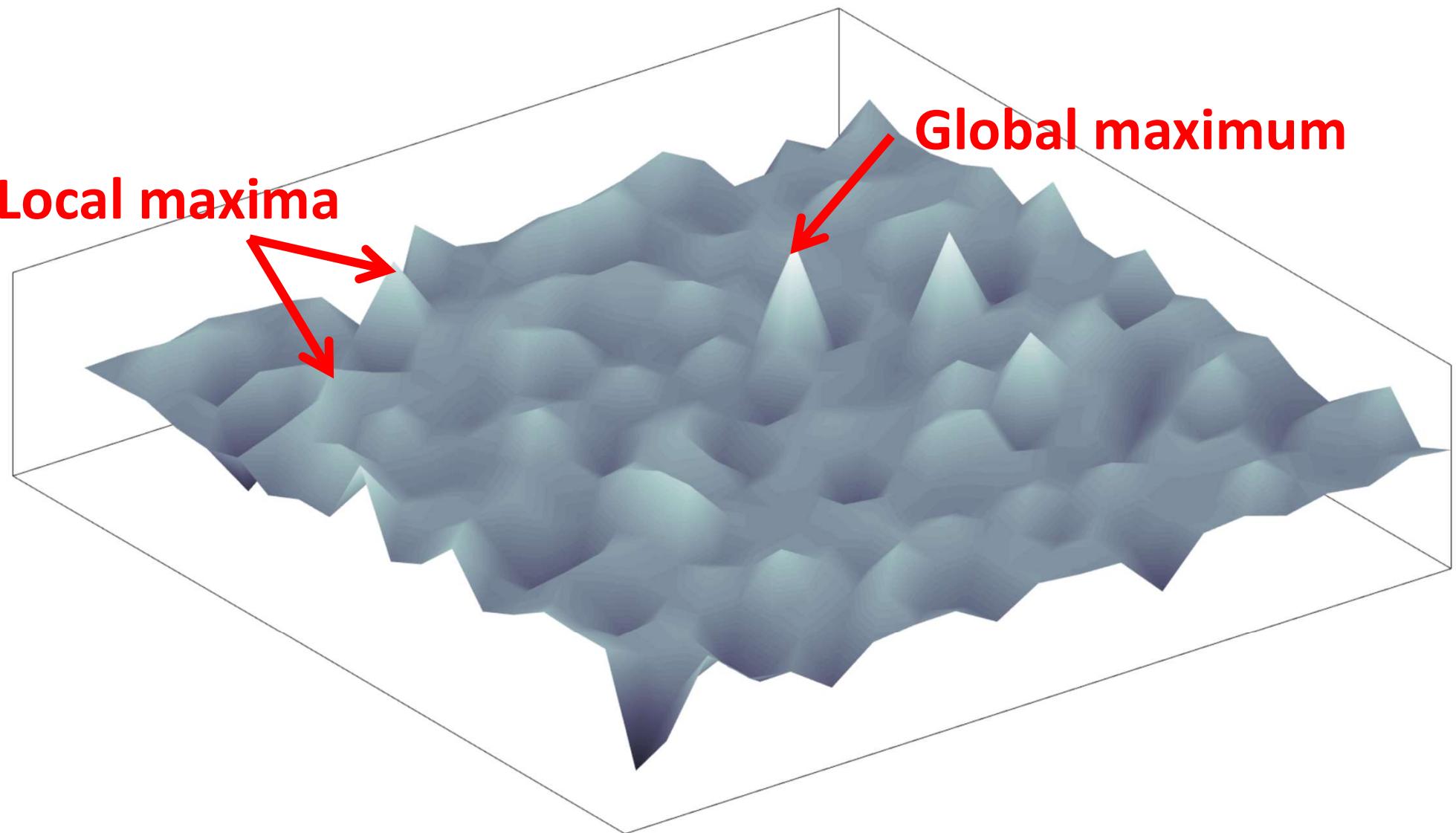
Romanian Roadtrip: Hill Climbing



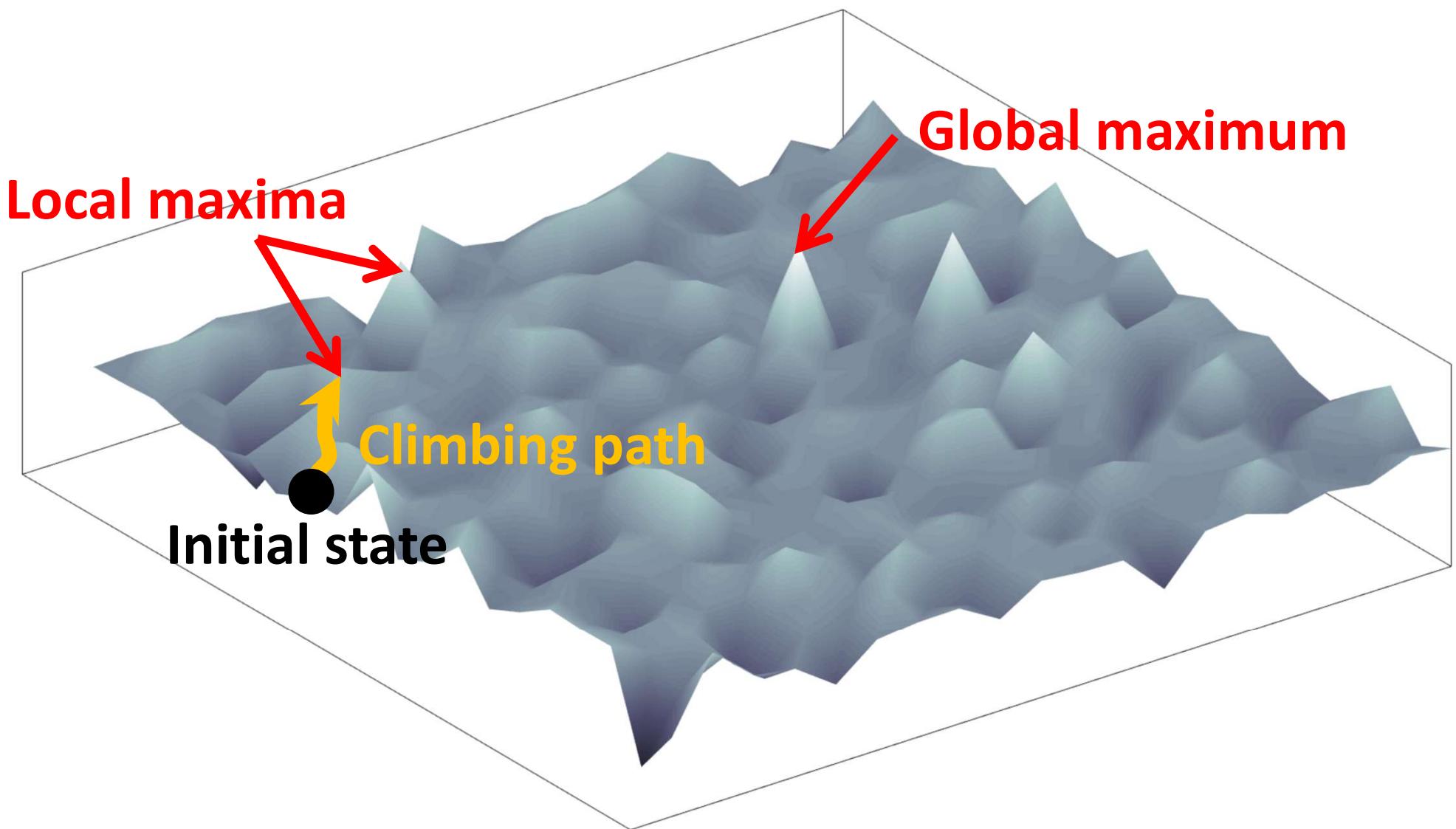
Alternatively:

- We could go to a repeated state end
- get stuck there
- or
- Infinite loop

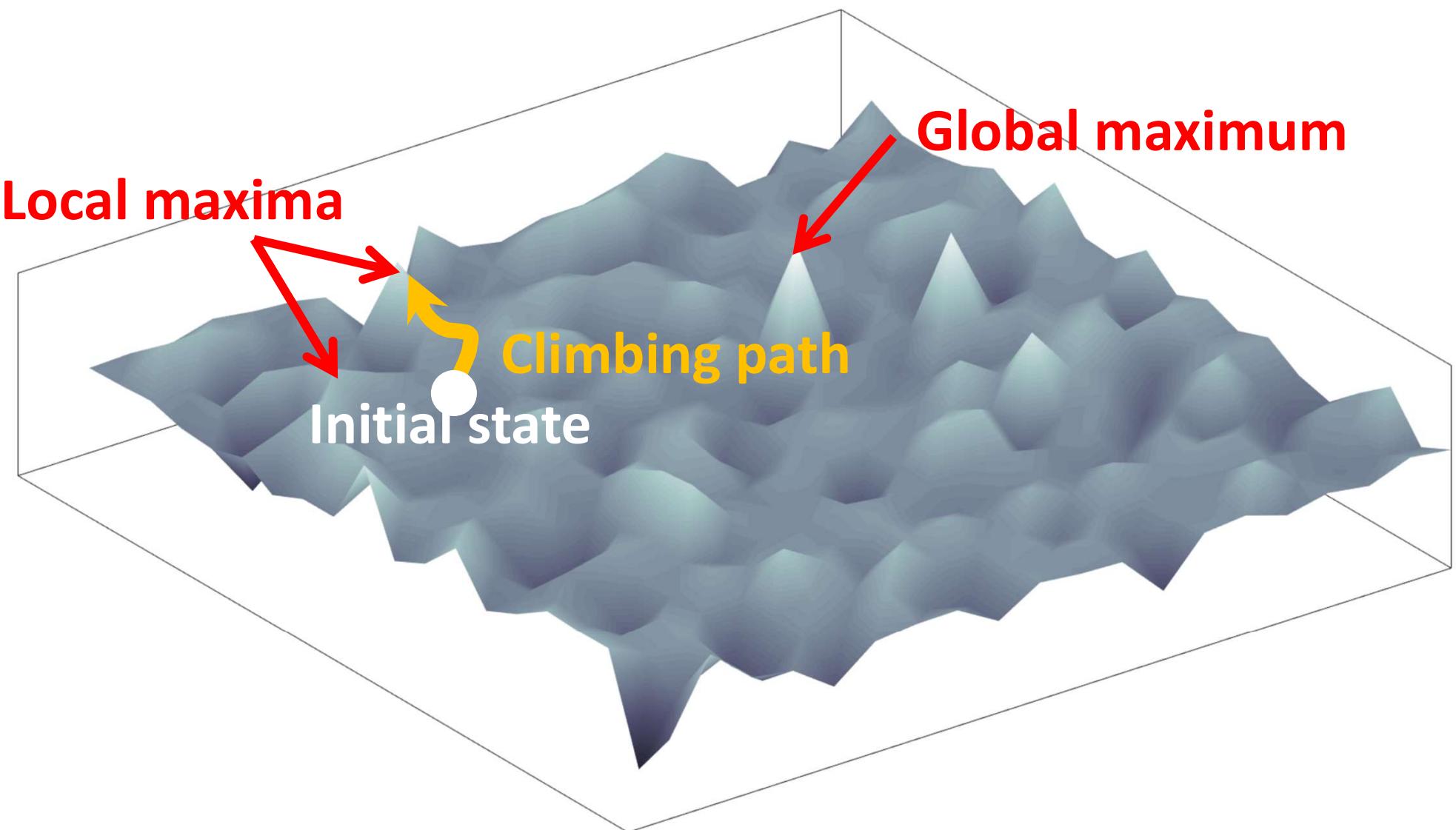
Hill Climbing Problems: Local Maxima



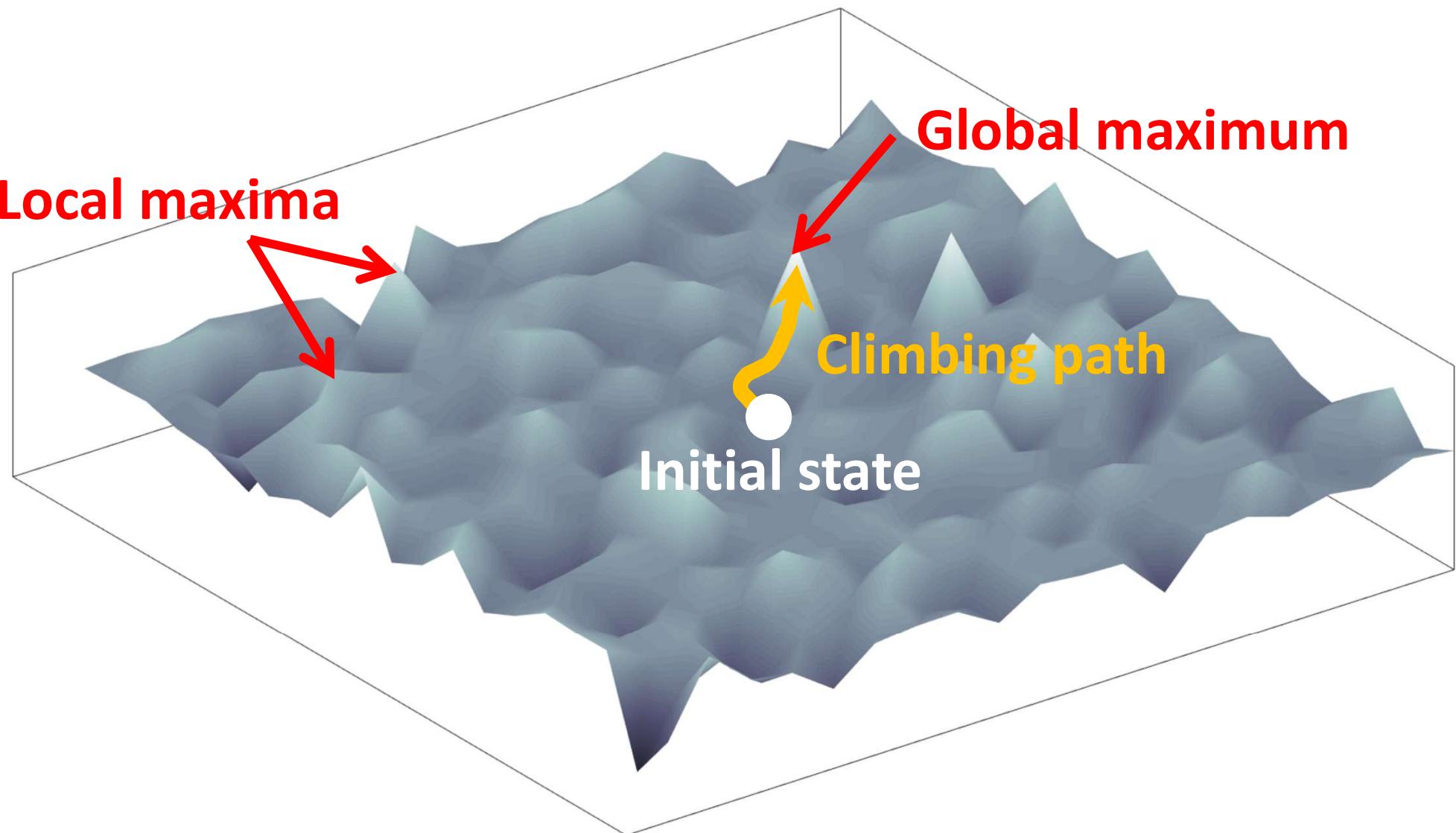
Hill Climbing Problems: Local Maxima



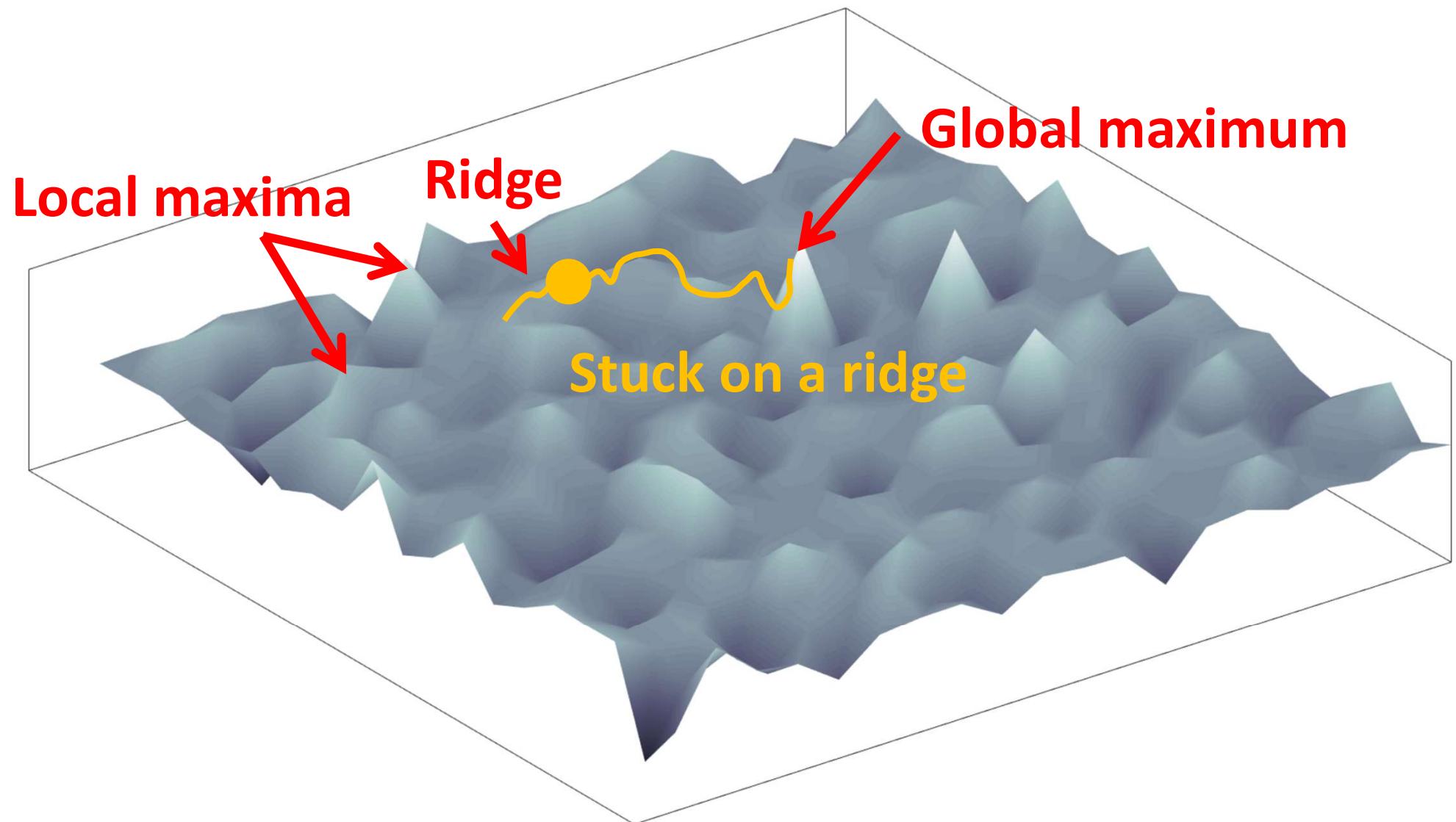
Hill Climbing Problems: Local Maxima



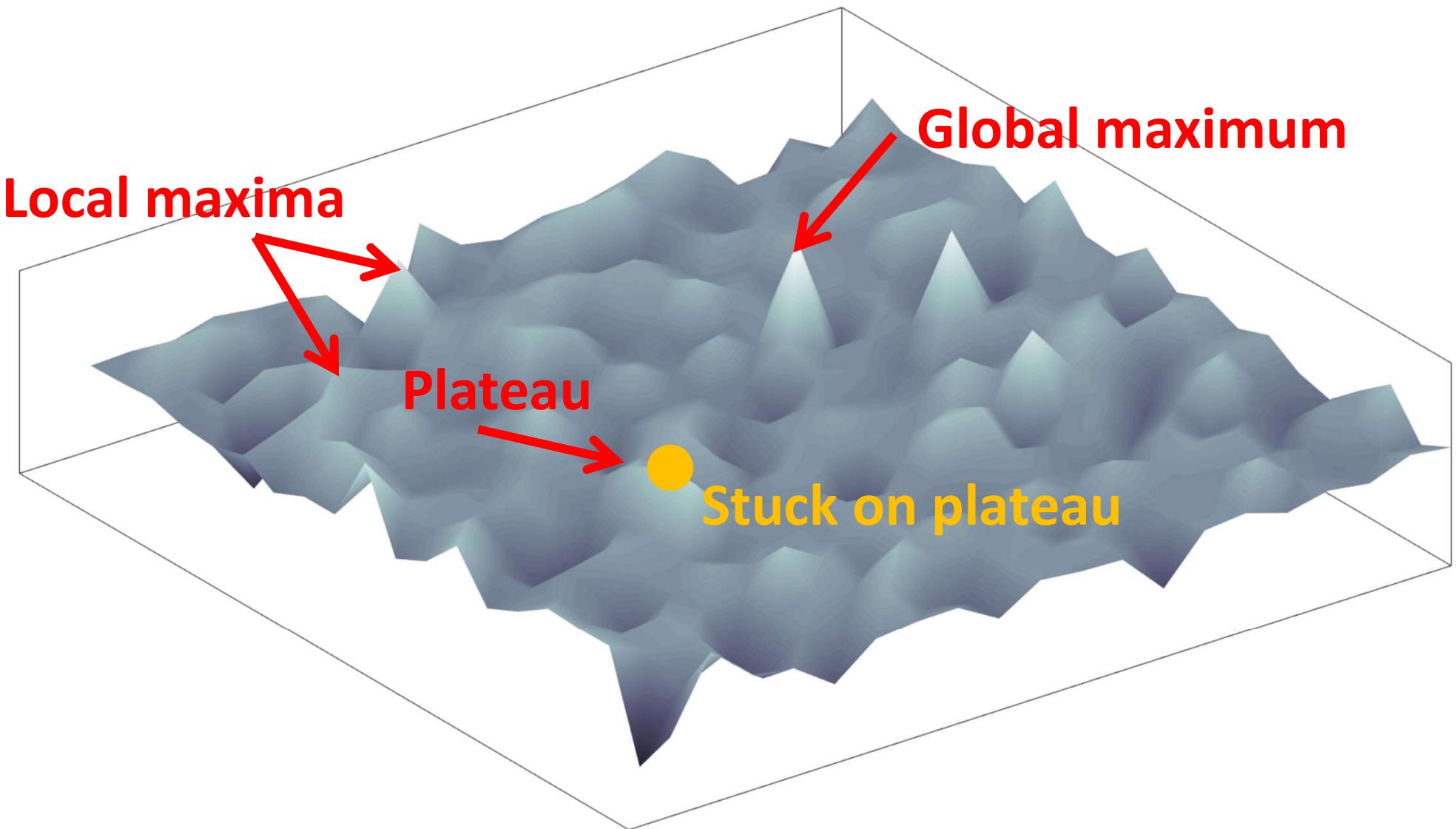
Hill Climbing Problems: Local Maxima



Hill Climbing Problems: Ridges



Hill Climbing Problems: Plateaus



Greedy Best First Search

- Also a rather primitive informed search approach
 - a naive greedy algorithm
 - evaluation function: **heuristics $h(n)$**
 - tries to not “move farther away” from the goal
 - does not care about the total path cost
- Practicalities:
 - it keep track of search history:
 - tracks visited states / nodes
 - tracks frontier states / nodes

Greedy Best First: Evaluation Function

Calculate / obtain:

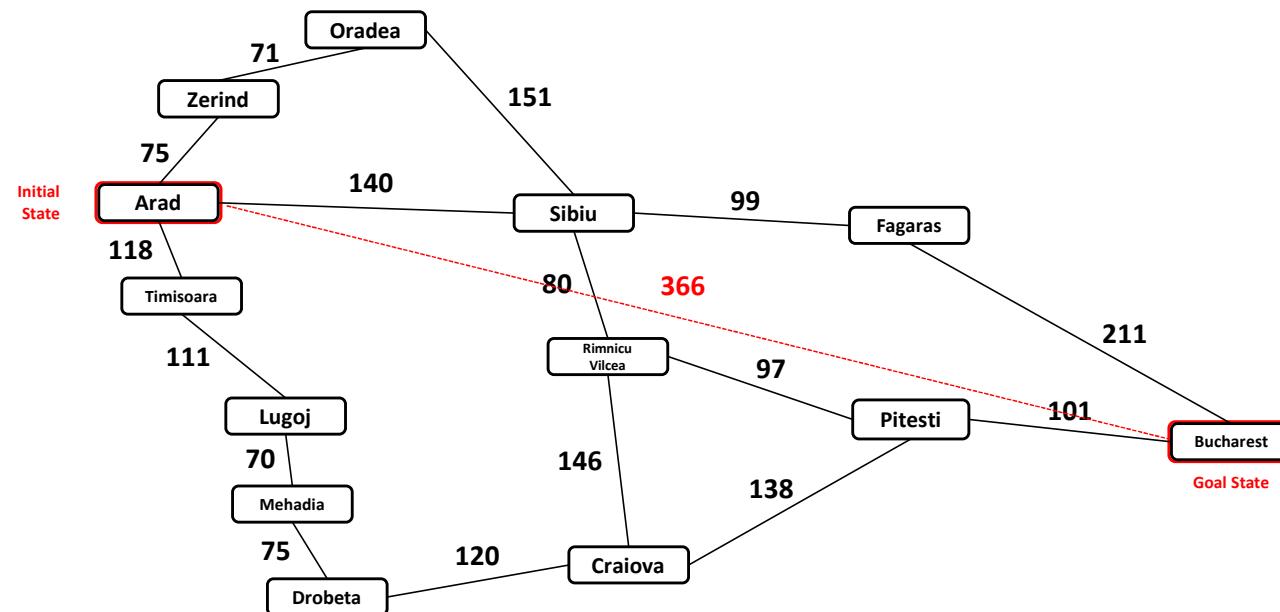
$$f(n) = h(\text{State}_n)$$

A node n with minimum (or maximum) $f(n)$ should be chosen for expansion

Romanian Roadtrip: Heuristics $h(n)$

For this particular problem the heuristic function $h(n)$ is defined by a **straight-line (Euclidean) distance** between two states (cities).

“As the crows flies” in other words.



Best-First Search: GBFS Pseudocode

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.Is-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

$f(n) = h(\text{State}_n)$

```
function EXPAND(problem, node) yields nodes
    s  $\leftarrow$  node.STATE
    for each action in problem.ACTIONS(s) do
        s'  $\leftarrow$  problem.RESULT(s, action)
        cost  $\leftarrow$  node.PATH-COST + problem.ACTION-COST(s, action, s')
        yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Best-First Search: GBFS Pseudocode

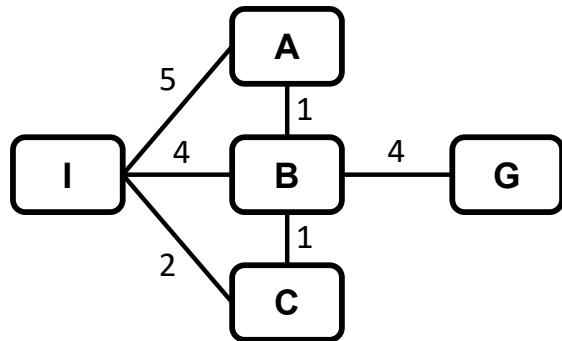
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.Is-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

$f(n) = h(\text{State}_n)$

Best-First Search is really a class of search algorithms that:

- Use the evaluation function $f(n)$ to pick next action
- Keep track of visited states
- Keep track of frontier states
- Evaluation function $f(n)$ choice controls their behavior

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	$f(\text{Node})$						

Reached	Parent						
	Key/State						
	Path cost						

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

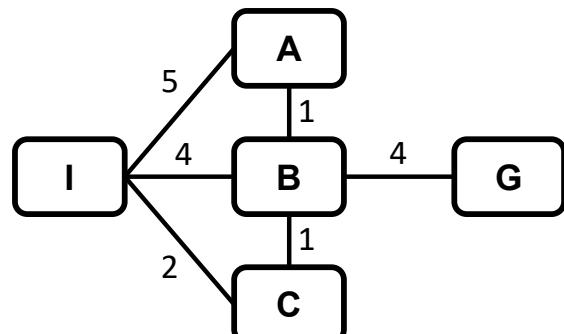
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

Greedy Best First Search: Example



g(state)
Path cost

Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

h(state)
Heuristic

Frontier	Parent					
	Node					
	f(Node)					

Reached	Parent					
	Key/State					
	Path cost					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

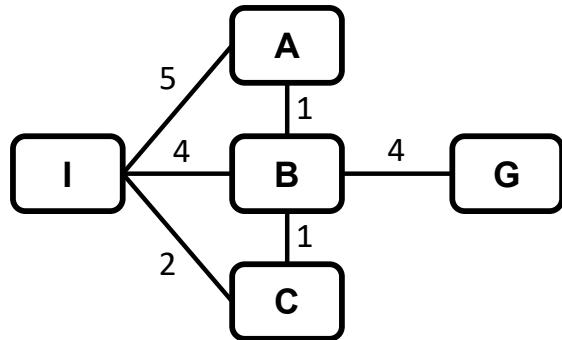
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

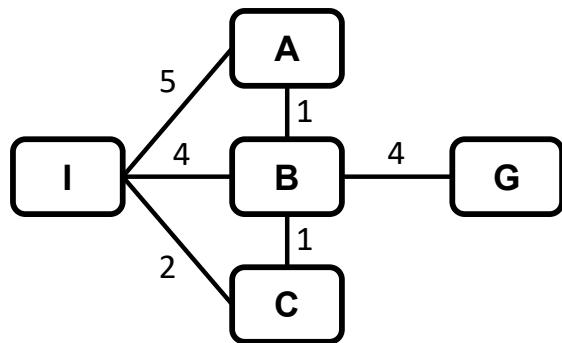
node \rightarrow

I



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	f(Node)						

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

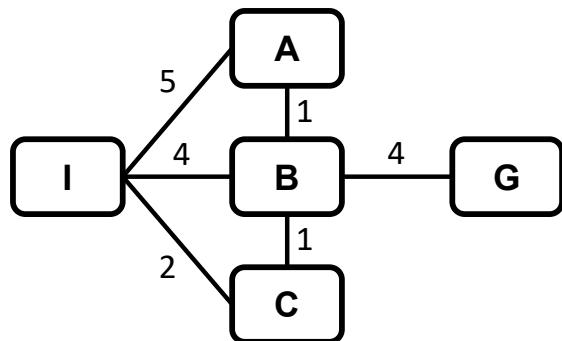
return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	$f(\text{Node})$	7					

```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node

  while not IS-EMPTY(frontier) do
    node ← POP(frontier)

    if problem.IS-GOAL(node.STATE) then return node

    for each child in EXPAND(problem, node) do
      s ← child.STATE

      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier

  return failure
  
```

State Space Graph	Frontier / Reached
Algorithm	Search Tree

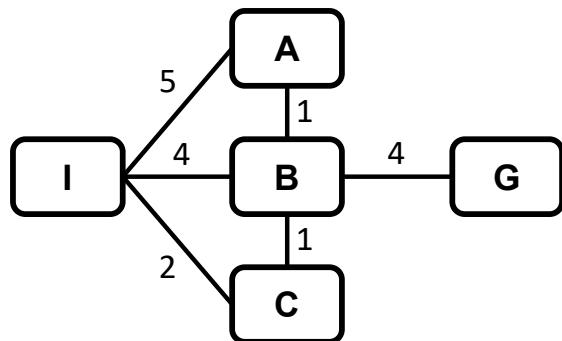


$$f(I) = h(I) = 7$$



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	$f(\text{Node})$	7					

Reached	Parent						
	Key/State						
	Path cost						

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

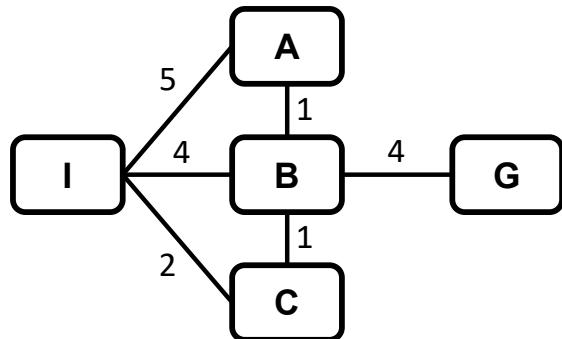
 add *child* to *frontier*

return failure



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	$f(\text{Node})$	7					

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

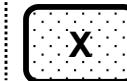
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

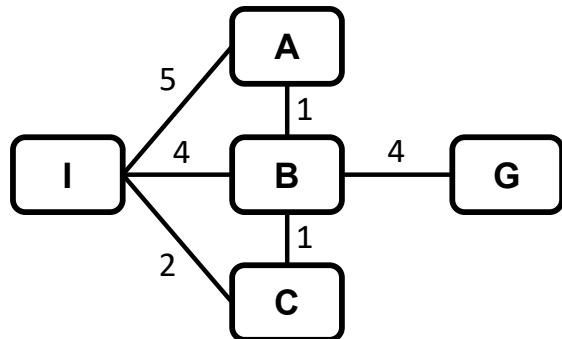


Path cost 0



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	$f(\text{Node})$	7					

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

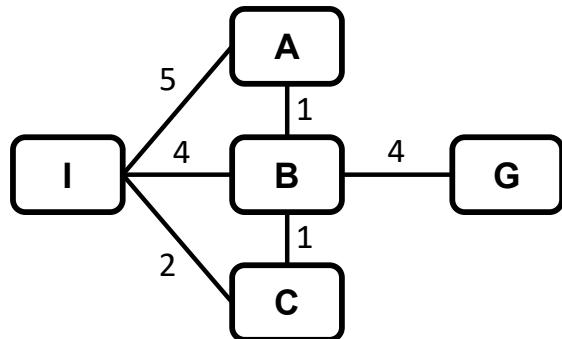
 add *child* to *frontier*

return failure



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	$f(\text{Node})$	7					

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

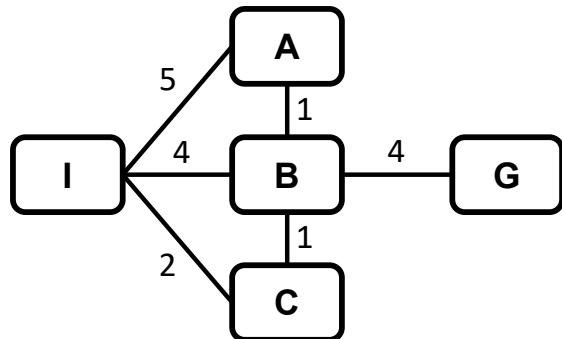
 add *child* to *frontier*

return failure



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	$f(\text{Node})$	7					

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

NOT EMPTY!

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

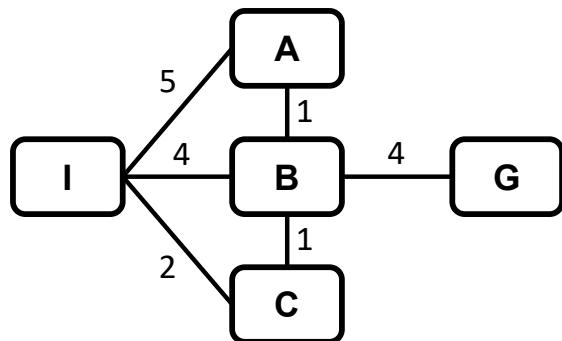
 add *child* to *frontier*

return failure



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	$f(\text{Node})$						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

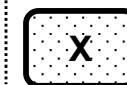
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

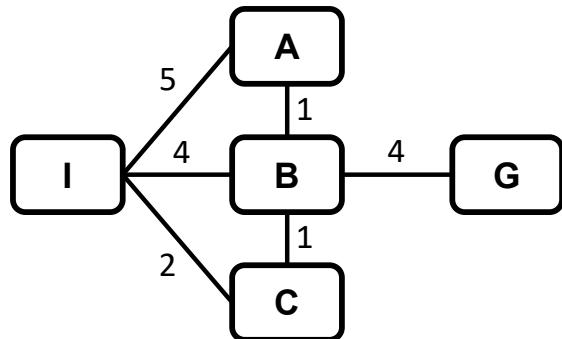
return failure

node \rightarrow



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	$f(\text{Node})$						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

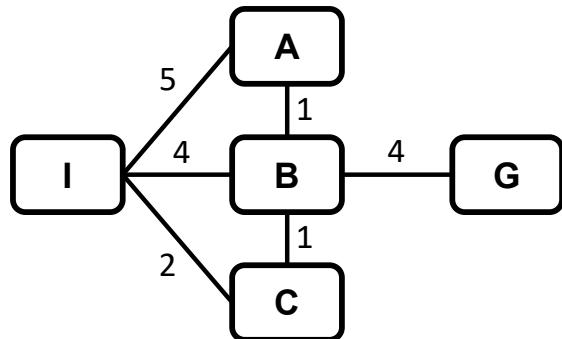
return failure

node \rightarrow



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	$f(\text{Node})$						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node* FALSE!

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

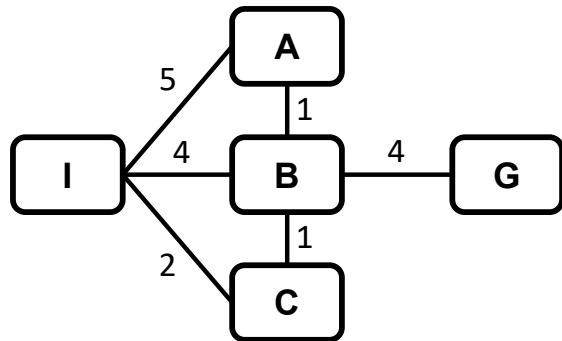
return failure

node \rightarrow



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	$f(\text{Node})$						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

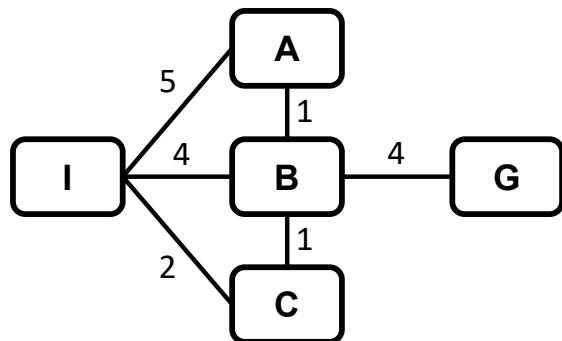
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
  
```

node →



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

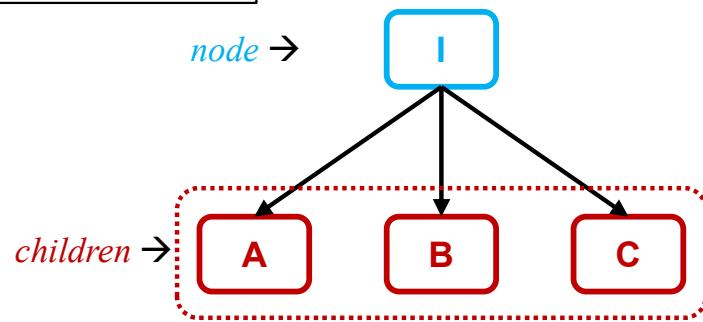
reached[s] \leftarrow *child*

 add *child* to *frontier*

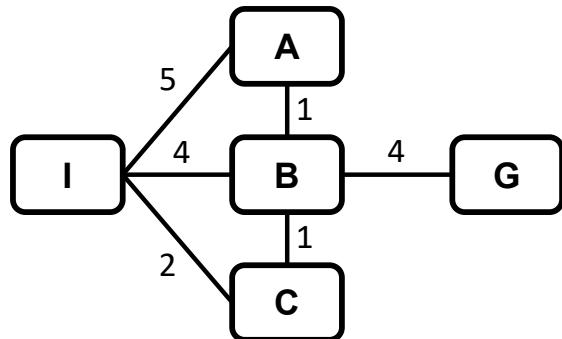
return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent					
	Node					
	$f(\text{Node})$					
Reached	Parent	----				
	Key/State	I				
	Path cost	0				



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent					
	Node					
	$f(\text{Node})$					

Reached	Parent	---				
	Key/State	I				
	Path cost	0				

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

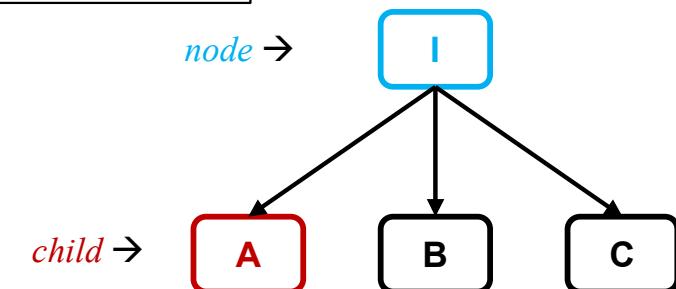
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

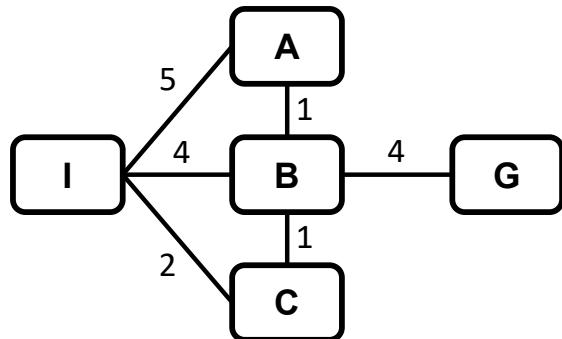
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent					
	Node					
	$f(\text{Node})$					

Reached	Parent	---				
	Key/State	I				
	Path cost	0				

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL*(*node.STATE*) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

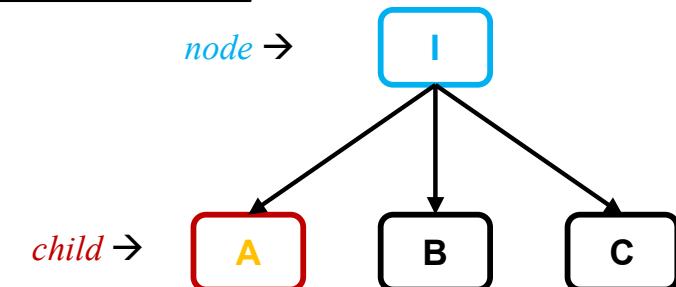
$s \leftarrow \text{child.STATE}$

 if s is not in *reached* or *child.PATH-COST* $<$ *reached*[s].*PATH-COST* then

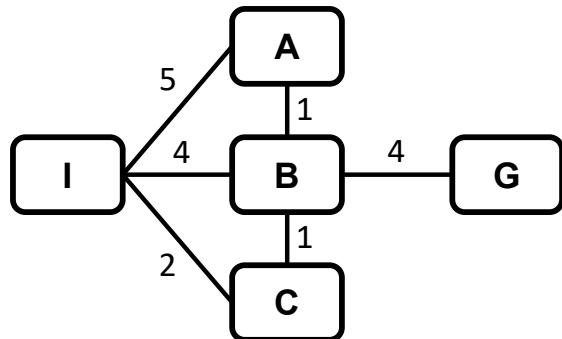
reached[s] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

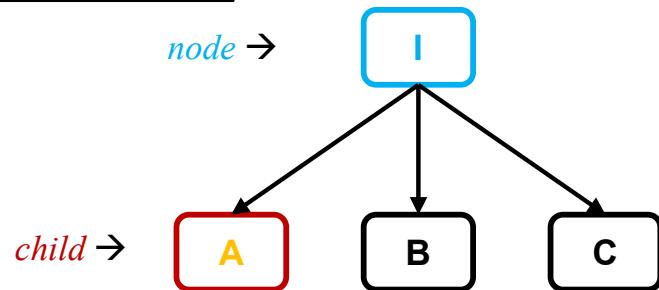
Frontier	Parent					
	Node					
	$f(\text{Node})$					

Reached	Parent	---				
	Key/State	I				
	Path cost	0				

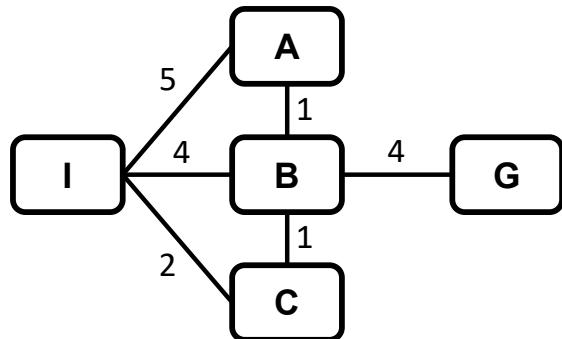
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

function BEST-FIRST-SEARCH(problem, f)
    node ← NODE(STATE=problem.INITIAL)
    frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
    reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
    while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s ← child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s] ← child
                add child to frontier
    return failure
    
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

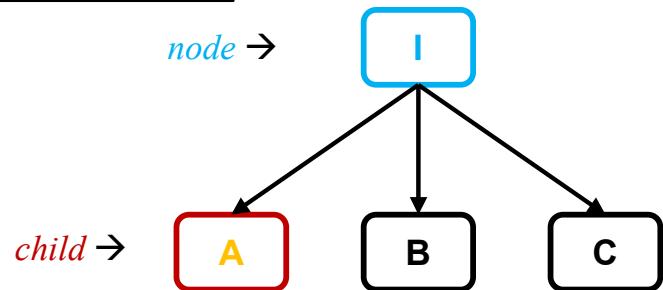
Frontier	Parent						
	Node						
	$f(\text{Node})$						

Reached	Parent	---					
	Key/State	I					
	Path cost	0					

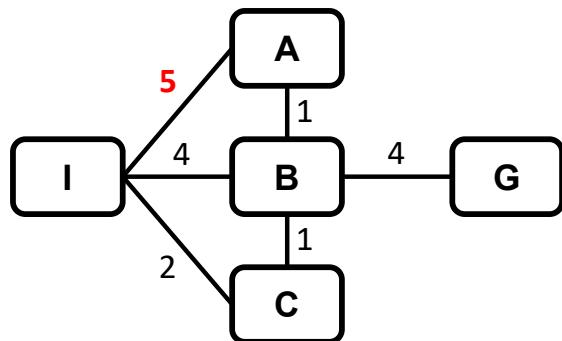
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
       $s \leftarrow child.\text{STATE}$ 
      if  $s$  is not in reached or  $child.\text{PATH-COST} < \text{reached}[s].\text{PATH-COST}$  then
        reached[ $s$ ] ← child
        add child to frontier
  return failure
  
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent					
	Node					
	f(Node)					

Reached	Parent	---	I			
	Key/State	I	A			
	Path cost	0	5			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

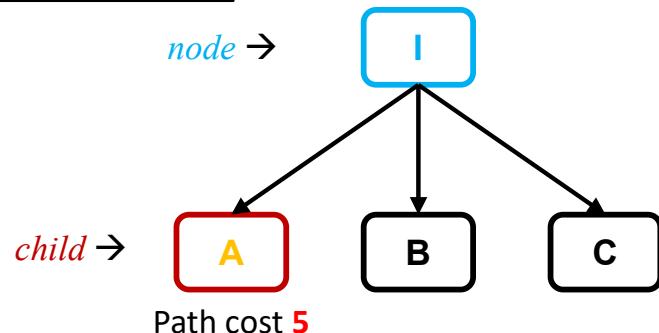
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

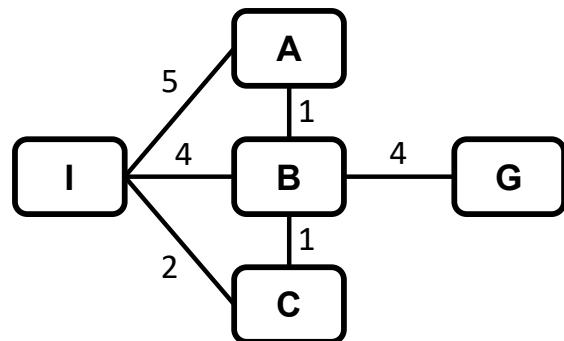
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

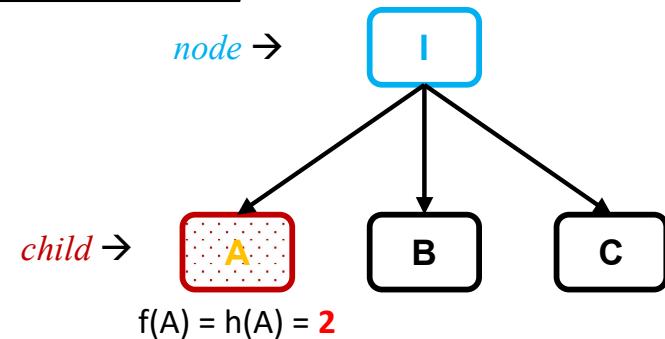
 add *child* to *frontier*

 return failure

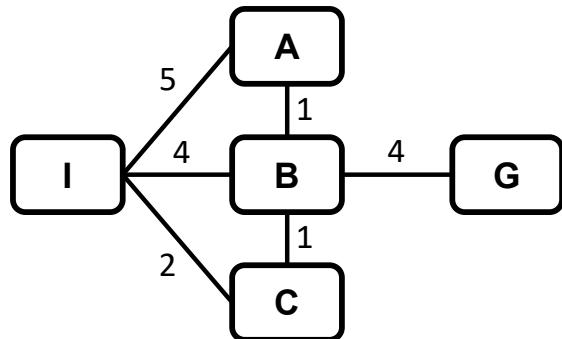
Frontier	Parent	I					
	Node	A					
	<i>f</i> (Node)	2					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				

State Space Graph	Frontier / Reached
Algorithm	Search Tree



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	A					
	f(Node)	2					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

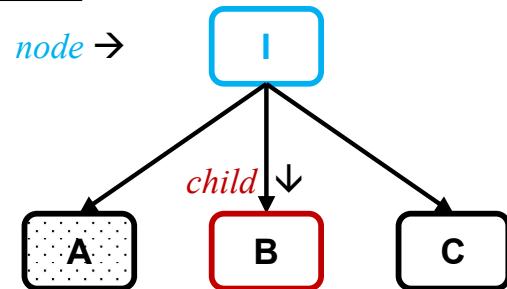
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

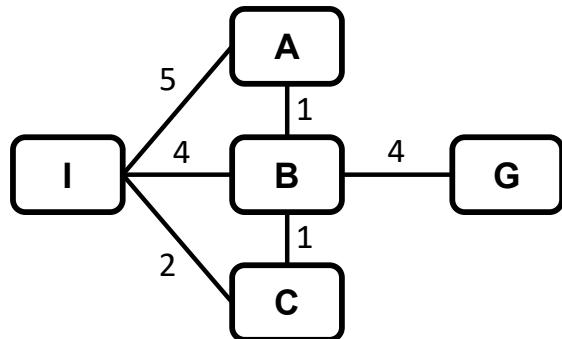
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

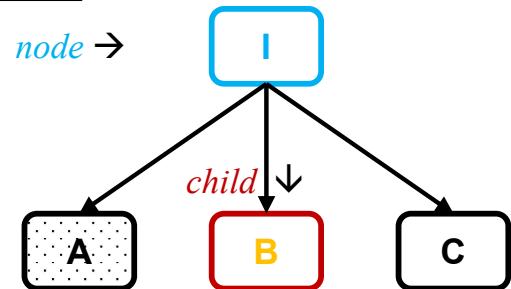
Frontier	Parent	I					
	Node	A					
	f(Node)	2					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				

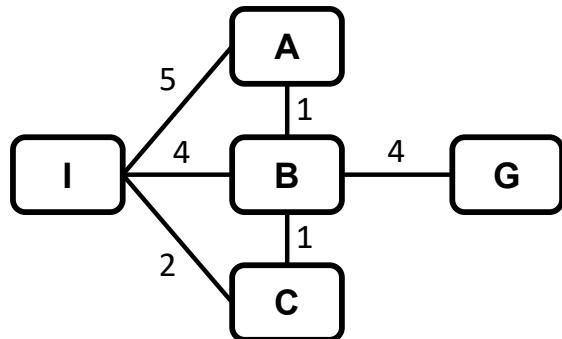
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
       $s \leftarrow child.\text{STATE}$ 
      if  $s$  is not in reached or  $child.\text{PATH-COST} < reached[s].\text{PATH-COST}$  then
        reached[ $s$ ] ← child
        add  $child$  to frontier
  return failure
  
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	A					
	f(Node)	2					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

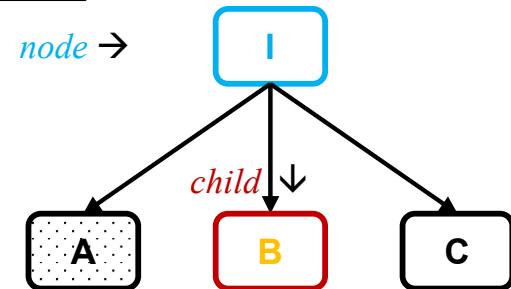
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

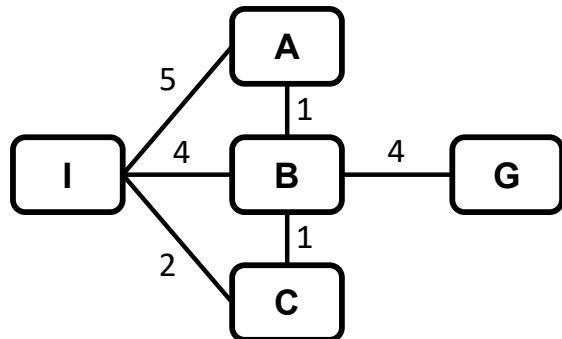
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

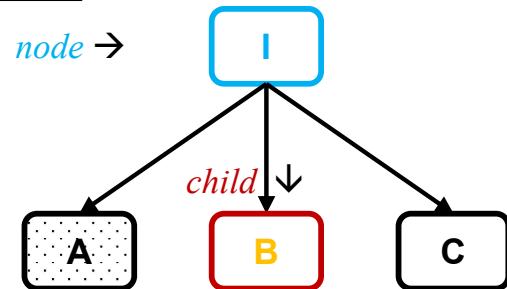
Frontier	Parent	I					
	Node	A					
	f(Node)	2					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				

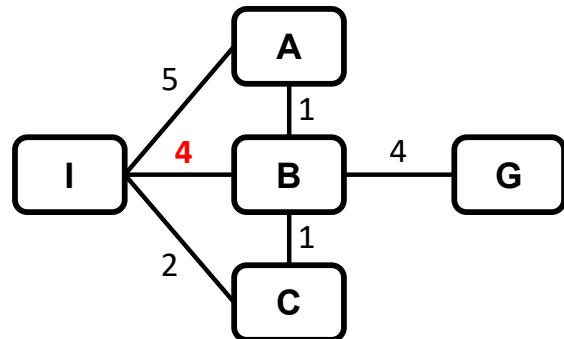
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
       $s \leftarrow child.\text{STATE}$ 
      if  $s$  is not in reached or  $child.\text{PATH-COST} < \text{reached}[s].\text{PATH-COST}$  then
        reached[ $s$ ] ← child
        add child to frontier
  return failure
  
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
Node	A						
f(Node)	2						

Reached	Parent	---	I	I			
Key/State	I	A	B				
Path cost	0	5	4				

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

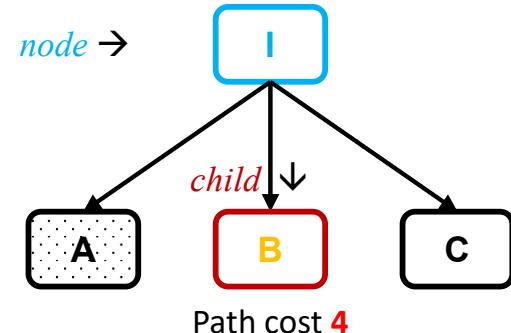
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

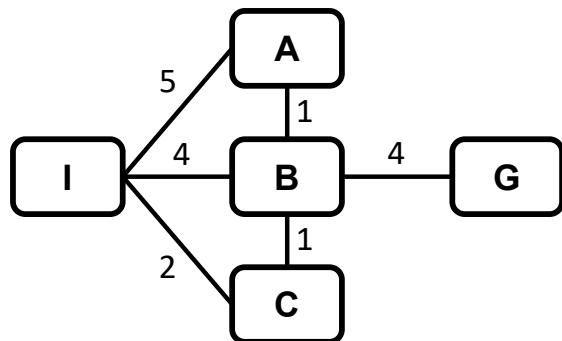
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	A	B				
	$f(\text{Node})$	2	3				

Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

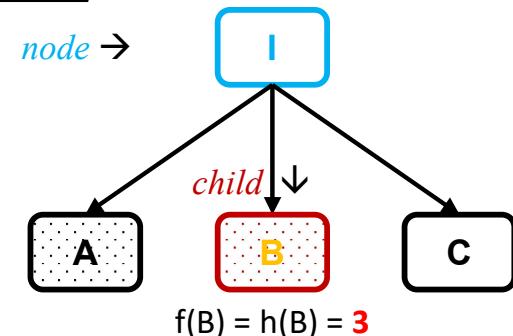
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

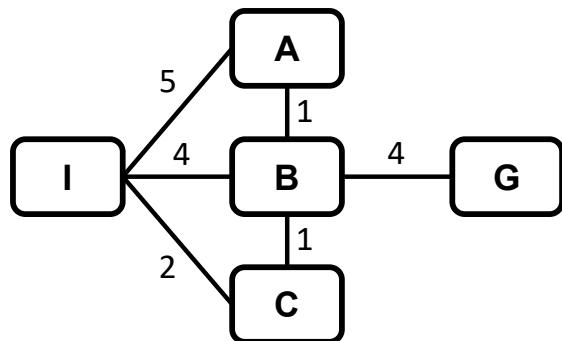
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST $<$ *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

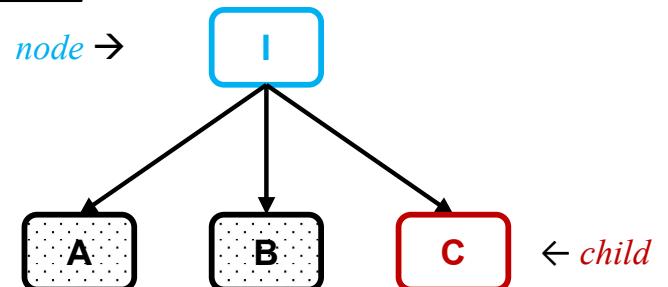
 add *child* to *frontier*

return failure

Frontier	Parent	I	I			
	Node	A	B			
	<i>f</i> (Node)	2	3			

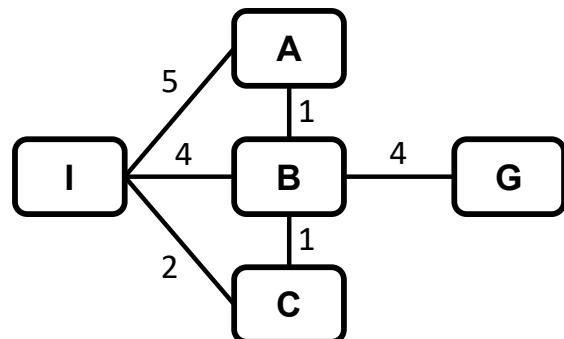
Reached	Parent	---	I	I		
	Key/State	I	A	B		
	Path cost	0	5	4		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	A	B				
	f(Node)	2	3				

Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

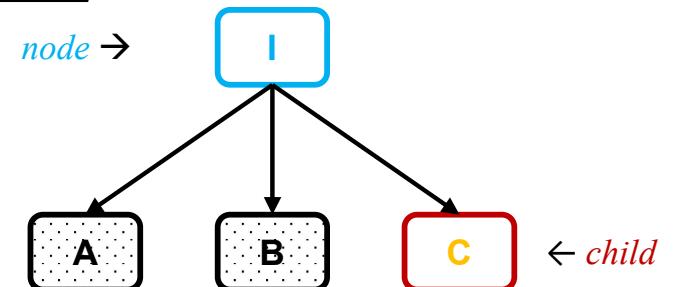
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

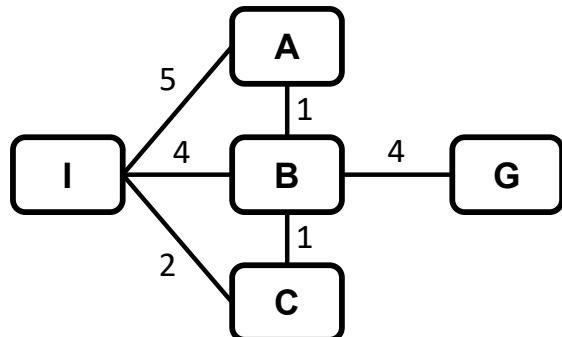
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

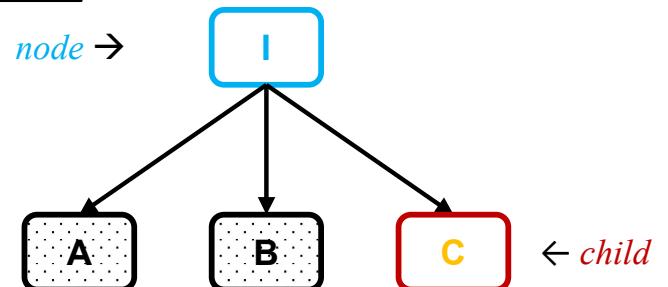
Frontier	Parent	I	I				
	Node	A	B				
	f(Node)	2	3				

Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

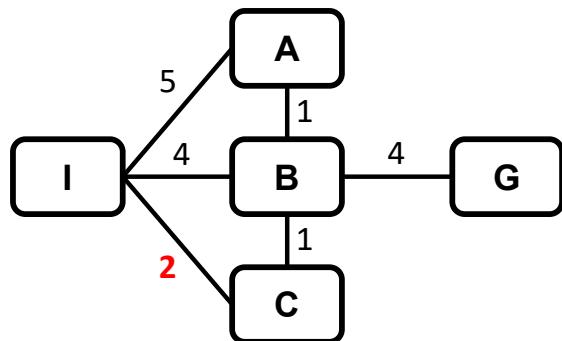
```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
       $s \leftarrow child.STATE$ 
      if  $s$  is not in reached or  $child.PATH-COST < reached[s].PATH-COST$  then
        reached[ $s$ ] ← child
        add child to frontier
  return failure
  
```



X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	A	B				
	f(Node)	2	3				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

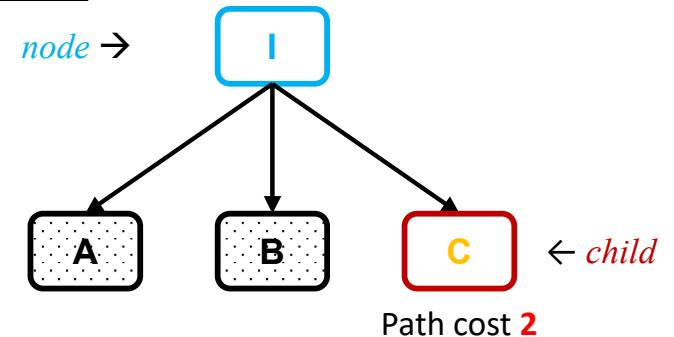
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

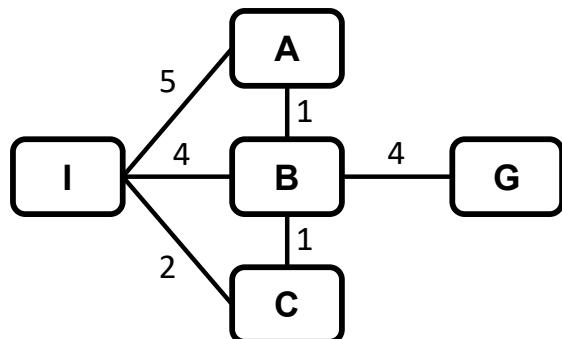
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I	I		
	Node	A	B	C		
	f(Node)	2	3	4		

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL*(*node.STATE*) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

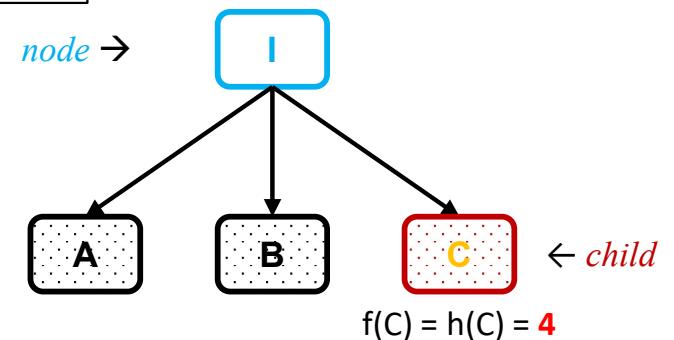
s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* then

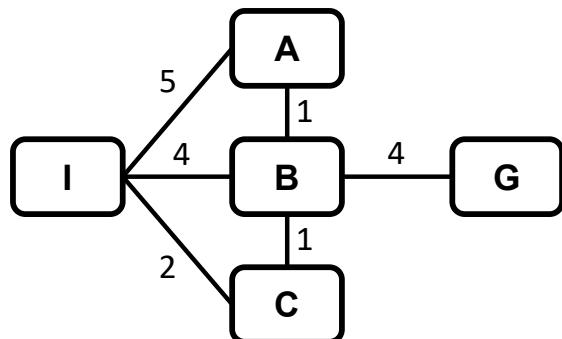
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I	I			
	Node	A	B	C			
	f(Node)	2	3	4			

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

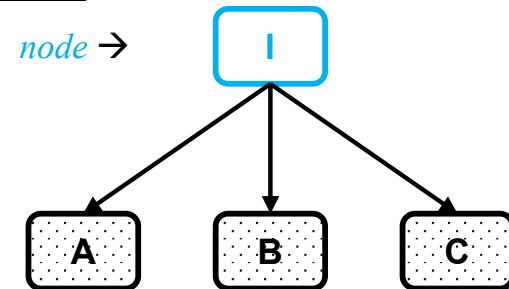
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

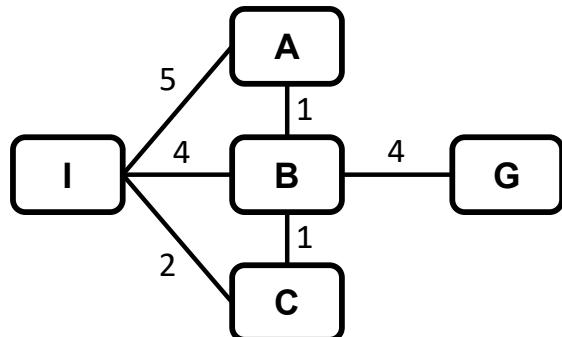
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I	I			
	Node	A	B	C			
	f(Node)	2	3	4			

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

NOT EMPTY!

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

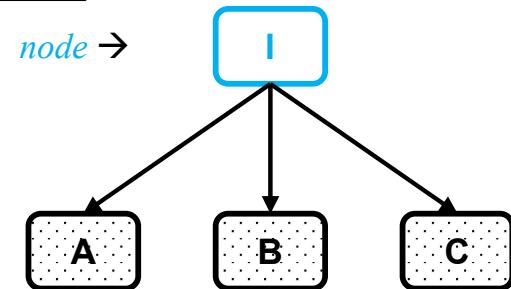
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

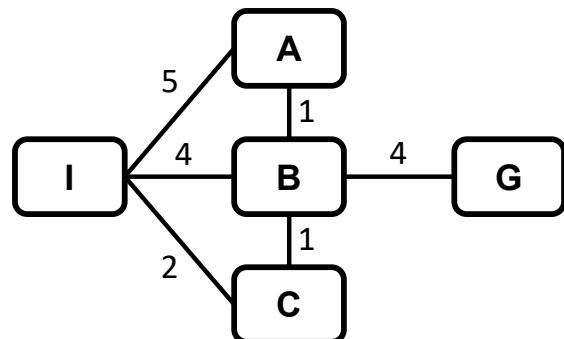
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

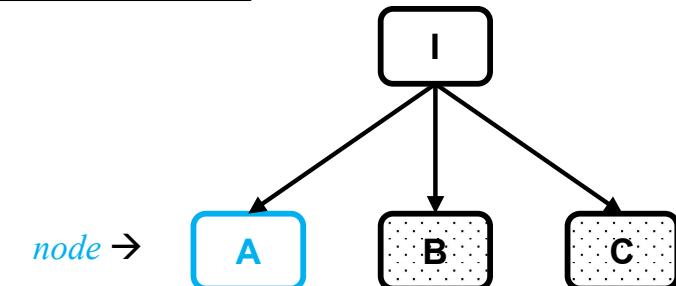
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

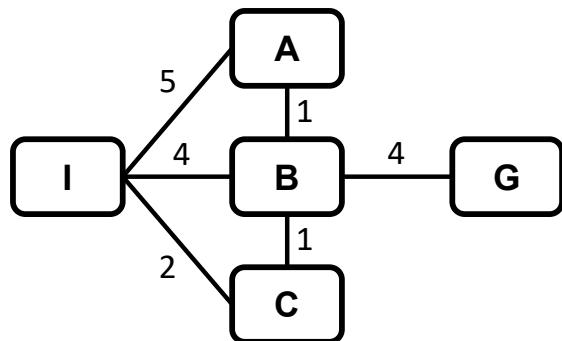
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

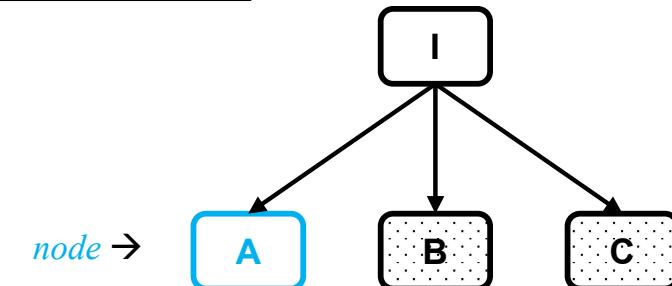
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

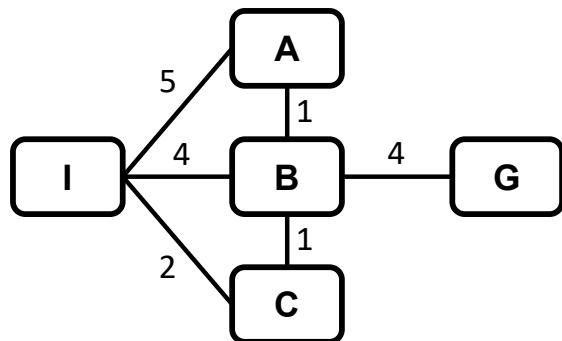
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node* FALSE!

 for each *child* in EXPAND(*problem*, *node*) do

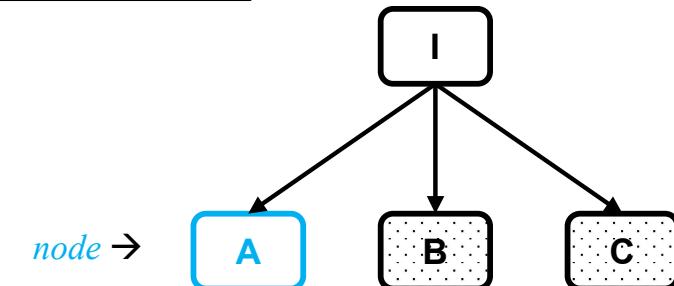
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

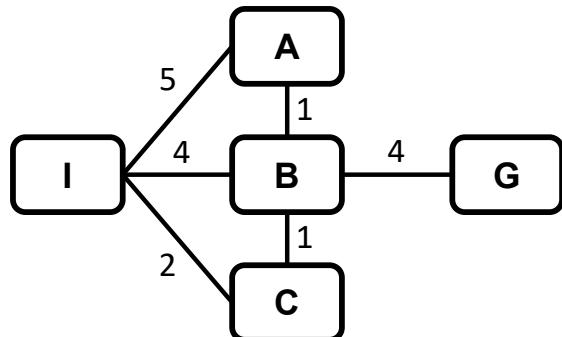
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

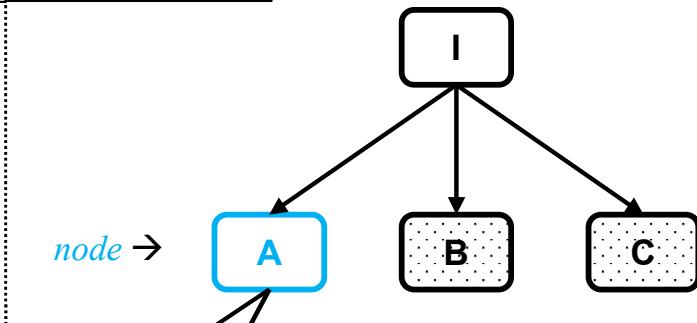
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

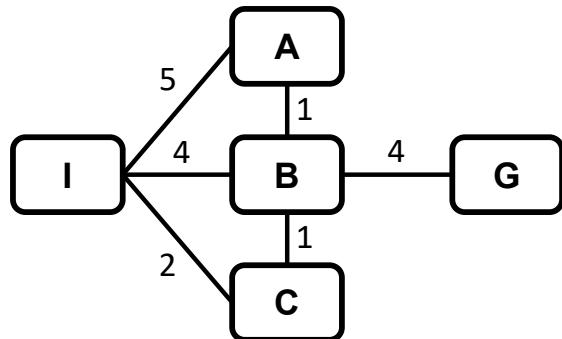
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

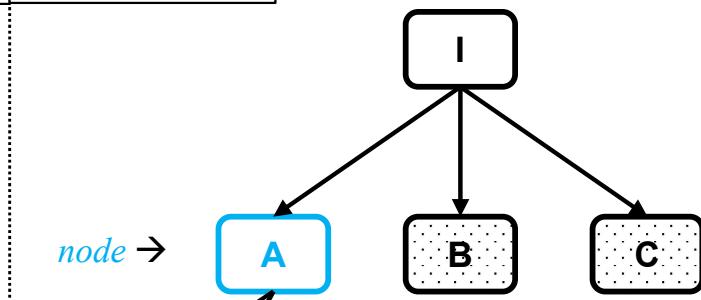
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

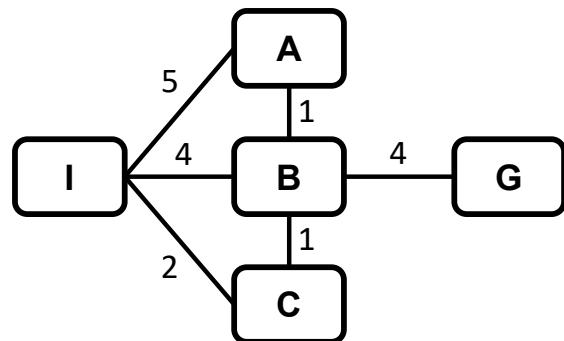
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

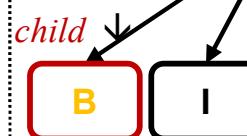
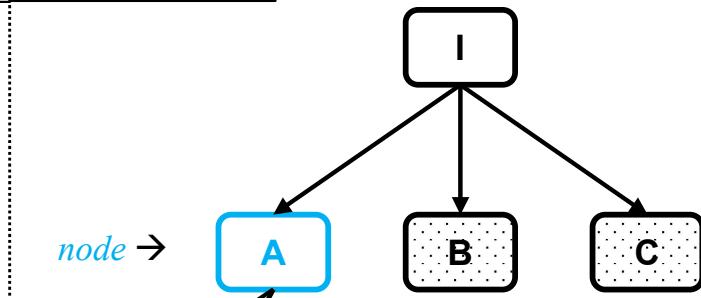
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

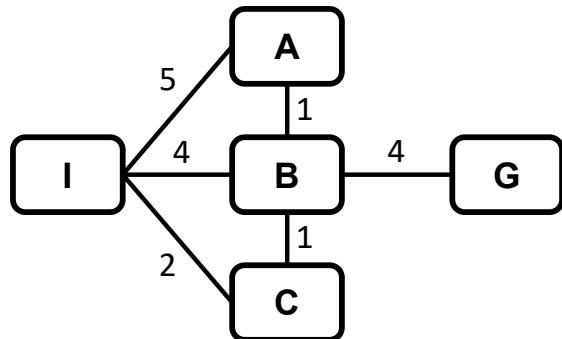
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
f(Node)	3	4					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

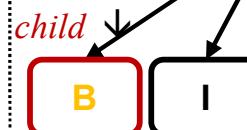
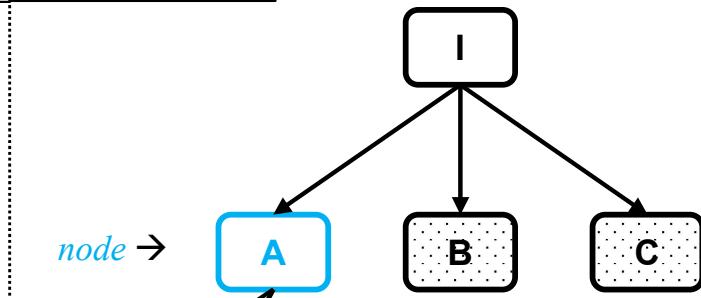
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

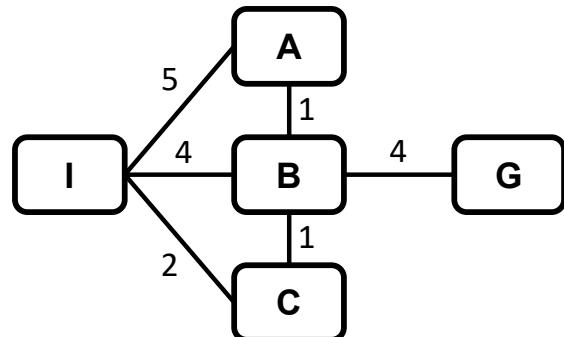
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
f(Node)	3	4					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

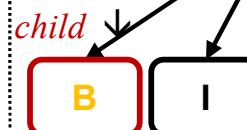
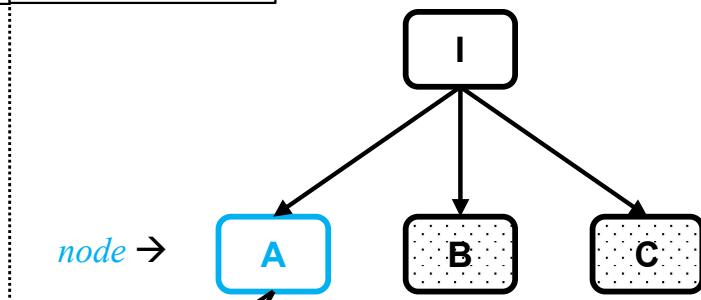
s \leftarrow *child*.STATE

 if *s* is not ~~X~~ *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

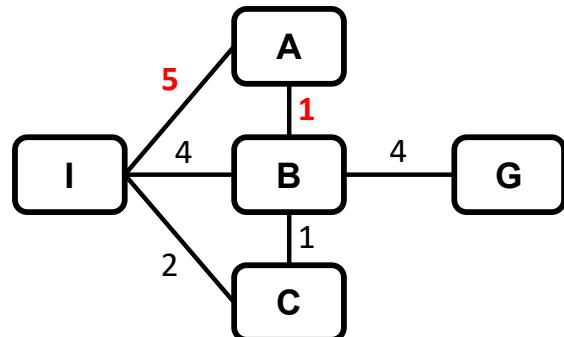
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I			
Node	B	C				
$f(\text{Node})$	3	4				

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

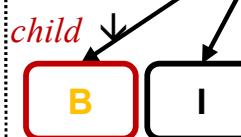
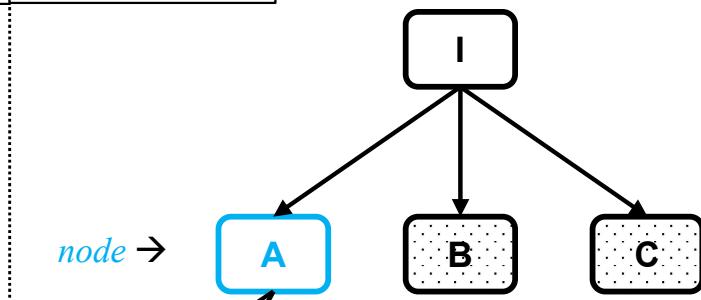
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<=~~ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

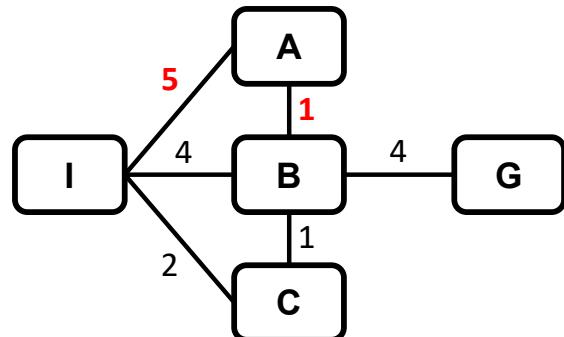
 return failure



Path cost
6

X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
$f(\text{Node})$	3	4					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

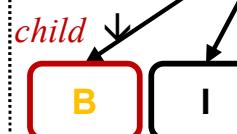
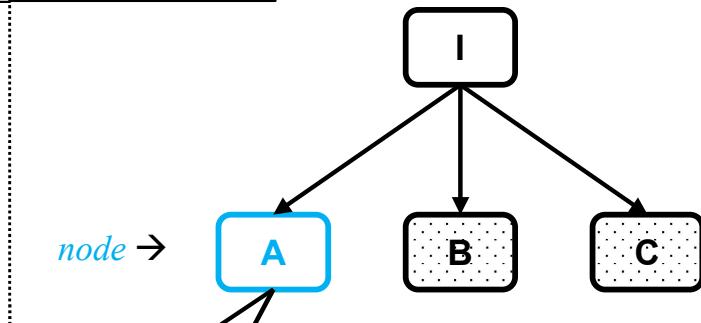
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<=~~ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

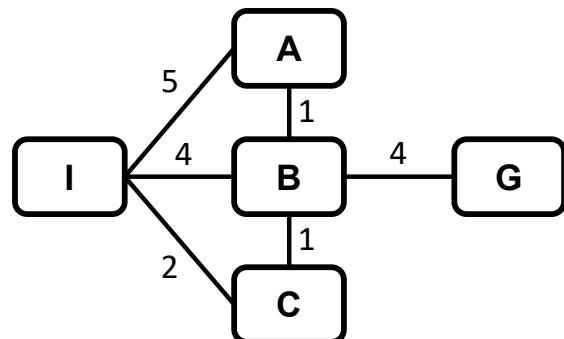
 return failure



Path cost
 $6 > 4$

X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
$f(\text{Node})$	3	4					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

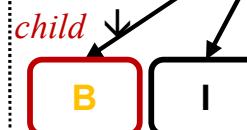
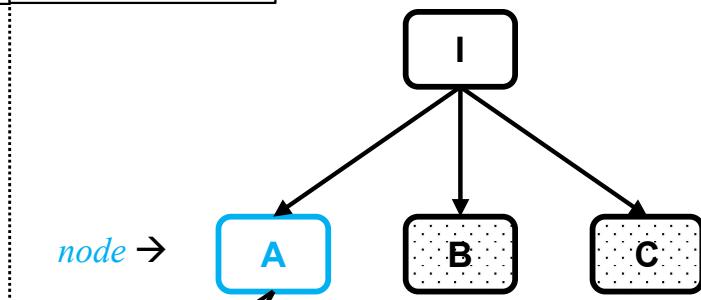
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<~~ *reached*[*s*].PATH-COST then

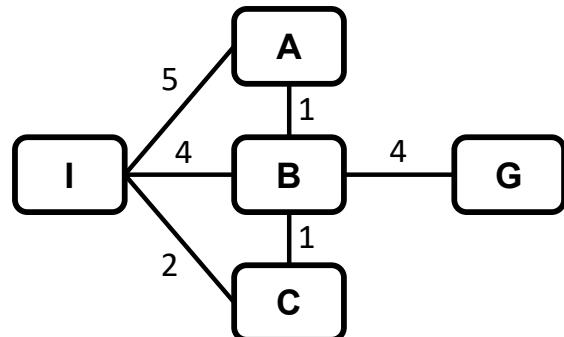
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

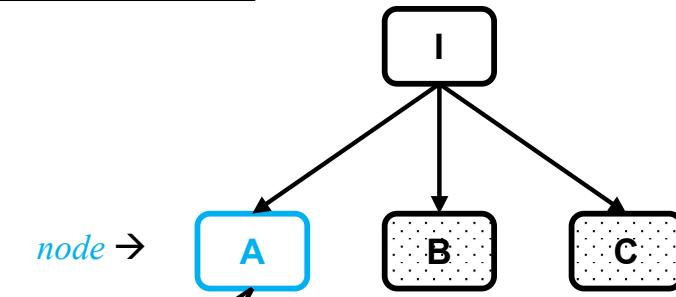
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

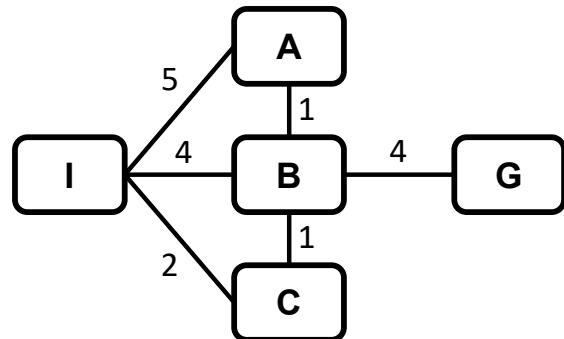
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
f(Node)	3	4					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

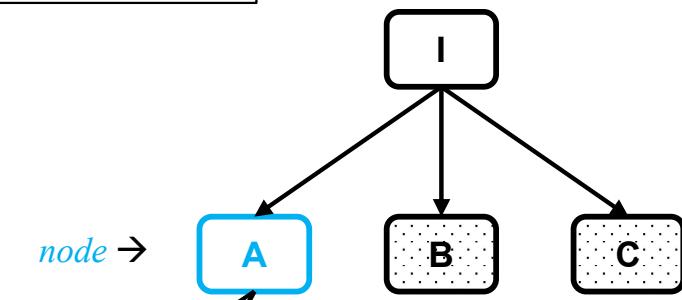
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

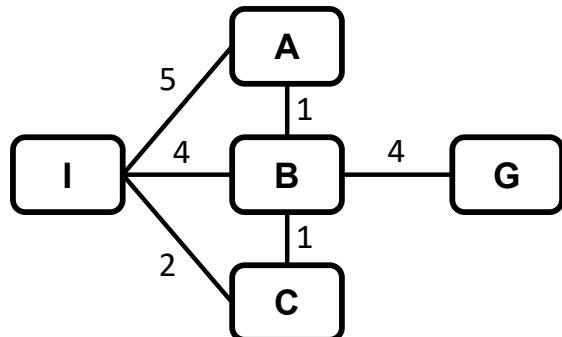
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
$f(\text{Node})$	3	4					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

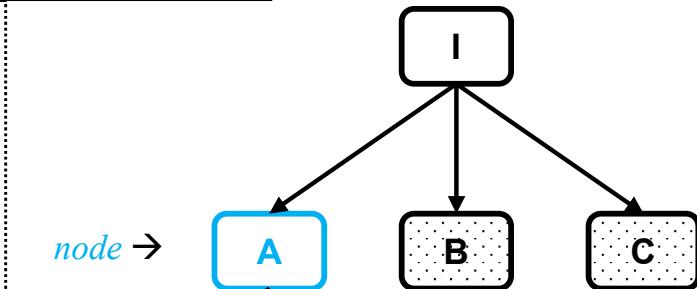
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

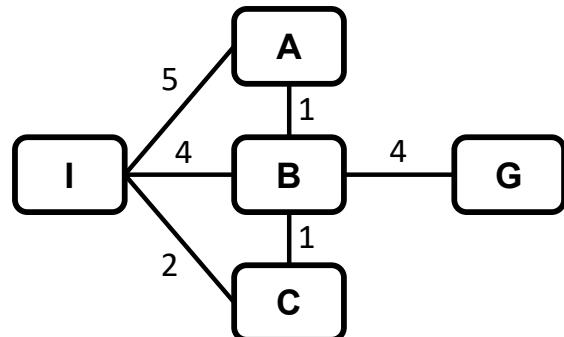
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
$f(\text{Node})$	3	4					

Reached	Parent	---	I	I	I			
Key/State	I	A	B	C				
Path cost	0	5	4	2				

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

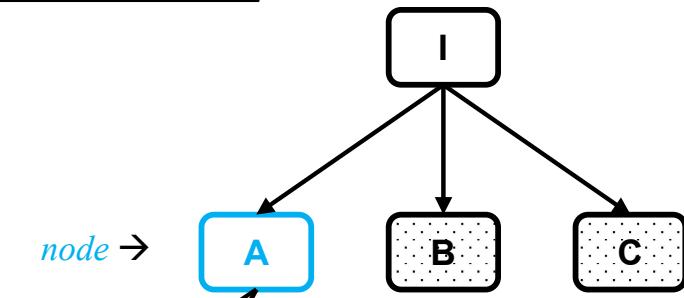
$s \leftarrow$ *child*.STATE

 if s is not ~~X~~ reached or *child*.PATH-COST $<$ *reached*[s].PATH-COST then

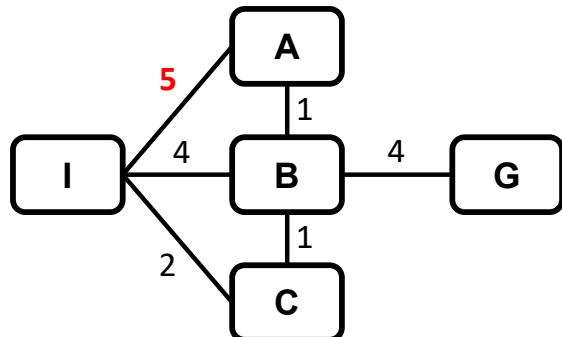
reached[s] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I			
Node	B	C				
f(Node)	3	4				

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

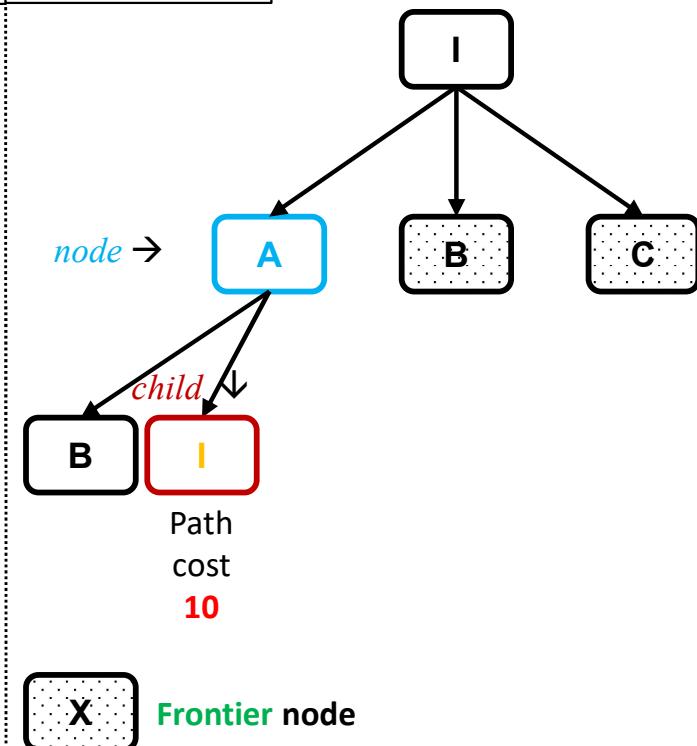
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<~~ *reached*[*s*].PATH-COST then

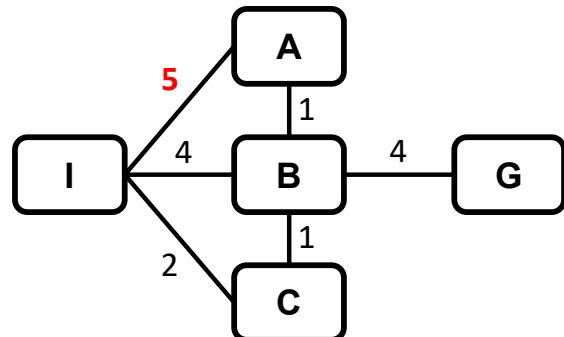
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
$f(\text{Node})$	3	4					

Reached	Parent	---	I	I	I			
Key/State	I	A	B	C				
Path cost	0	5	4	2				

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

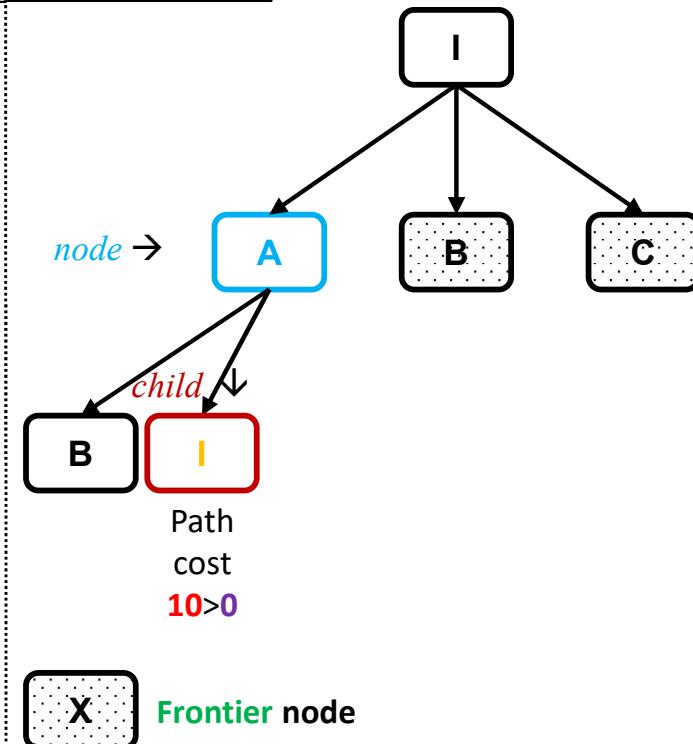
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<=~~ *reached*[*s*].PATH-COST then

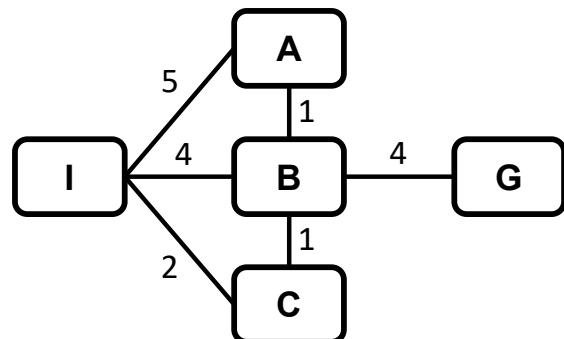
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	C					
f(Node)	3	4					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

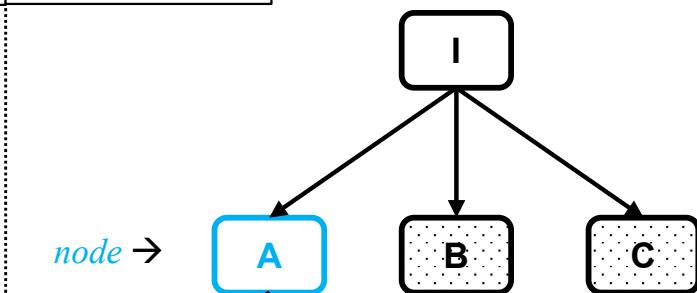
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<~~ *reached*[*s*].PATH-COST then

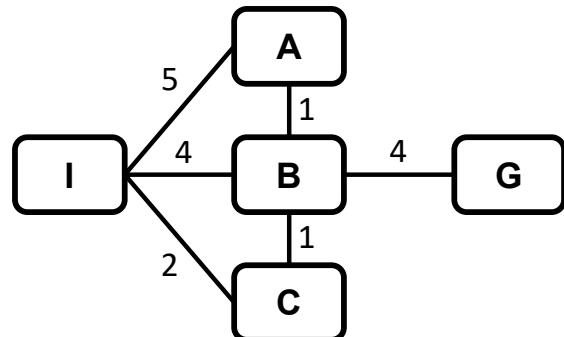
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

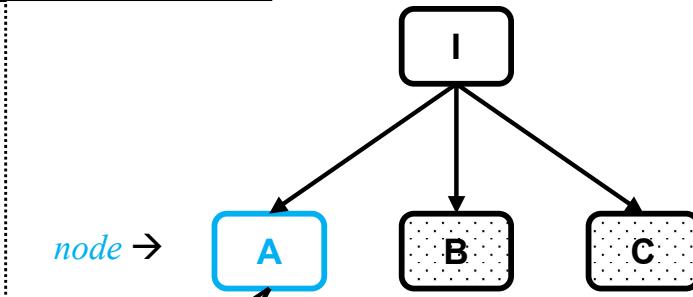
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

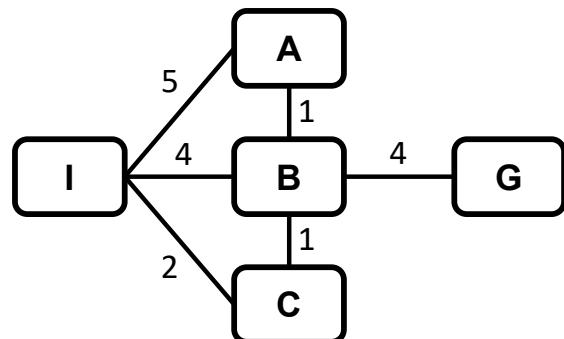
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	C				
	f(Node)	3	4				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

 NOT EMPTY!

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

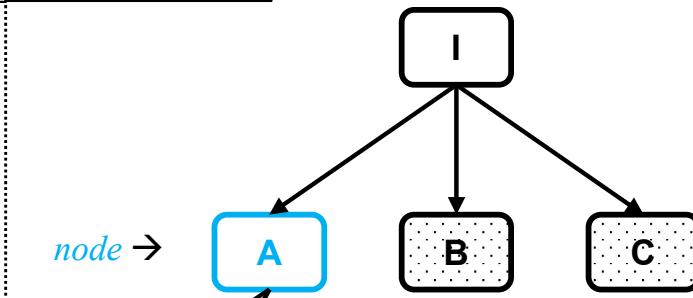
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

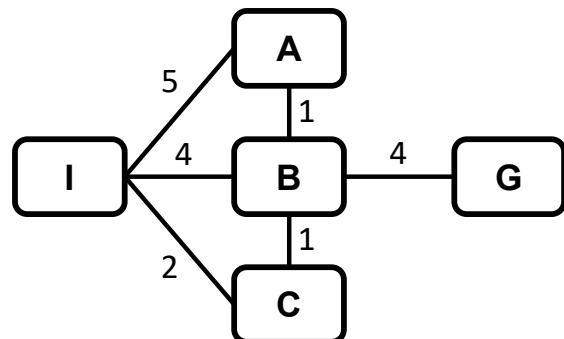
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

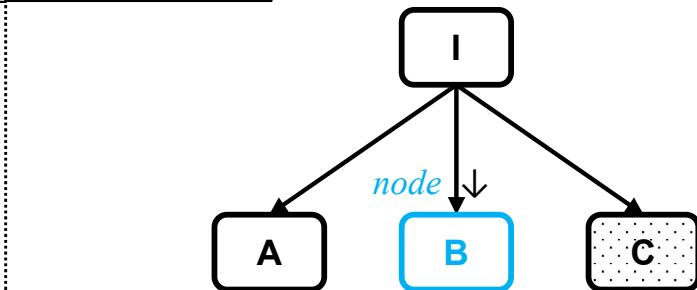
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

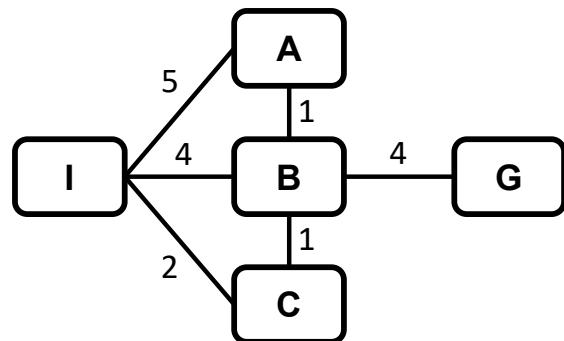
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

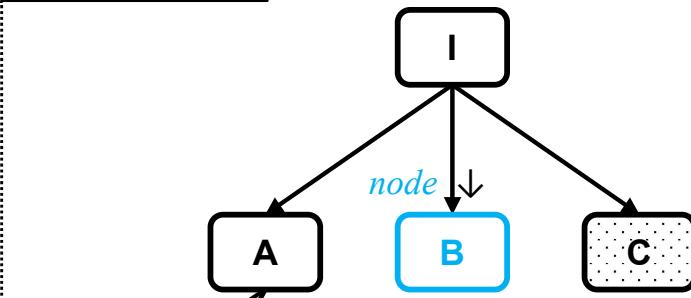
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

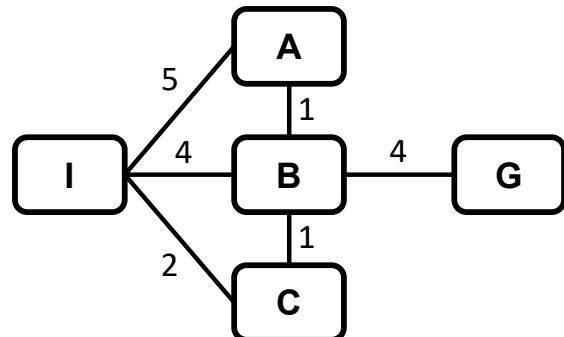
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
Node	C						
f(Node)	4						

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node* FALSE!

 for each *child* in EXPAND(*problem*, *node*) do

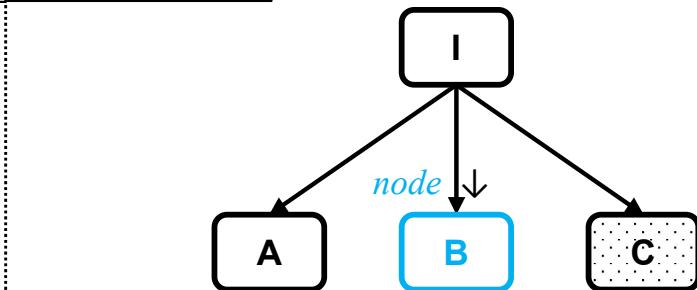
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

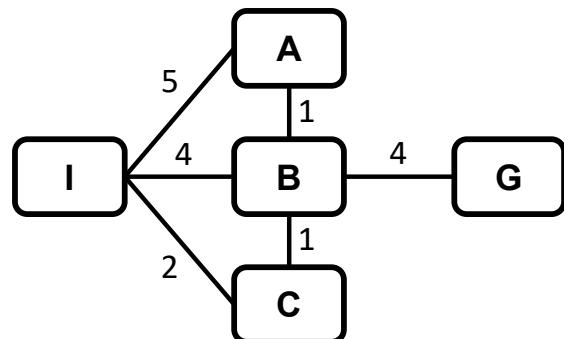
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

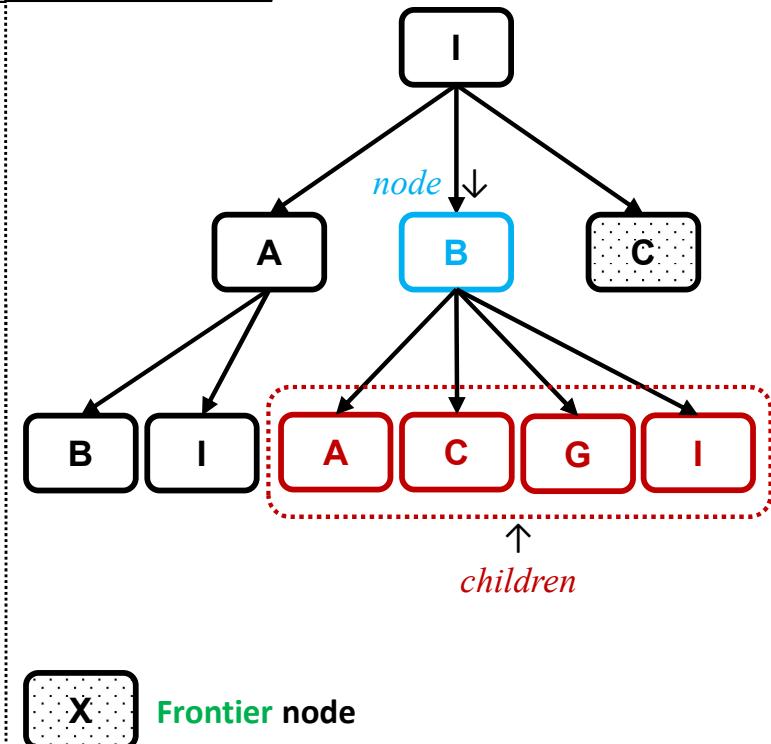
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

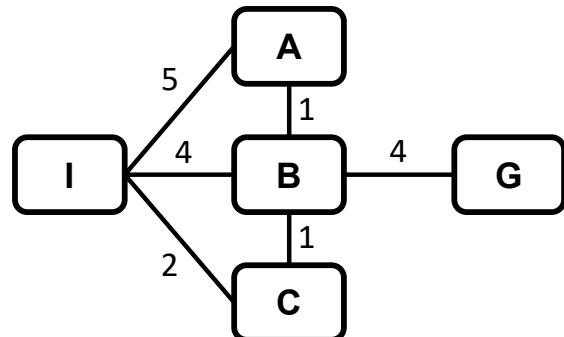
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

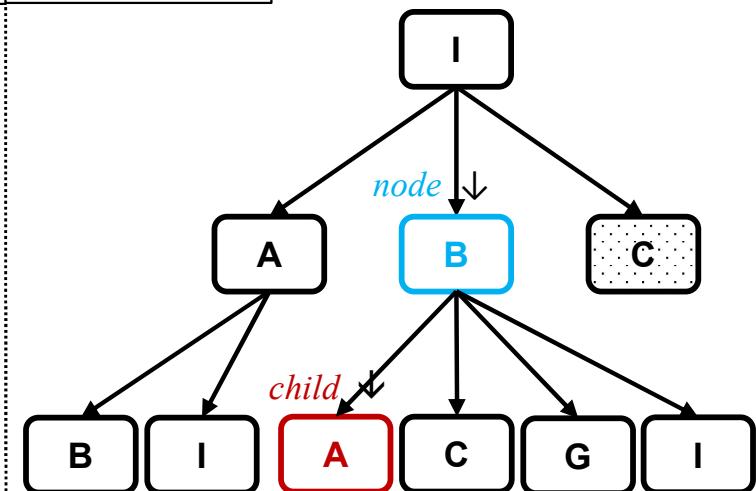
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

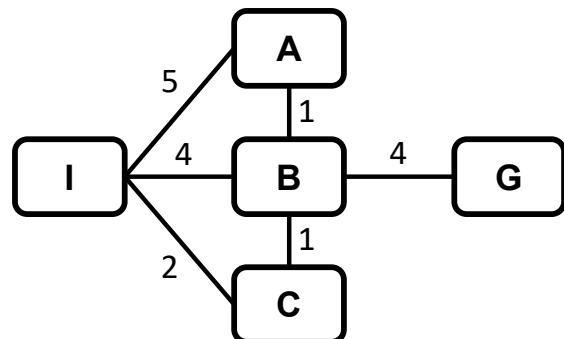
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

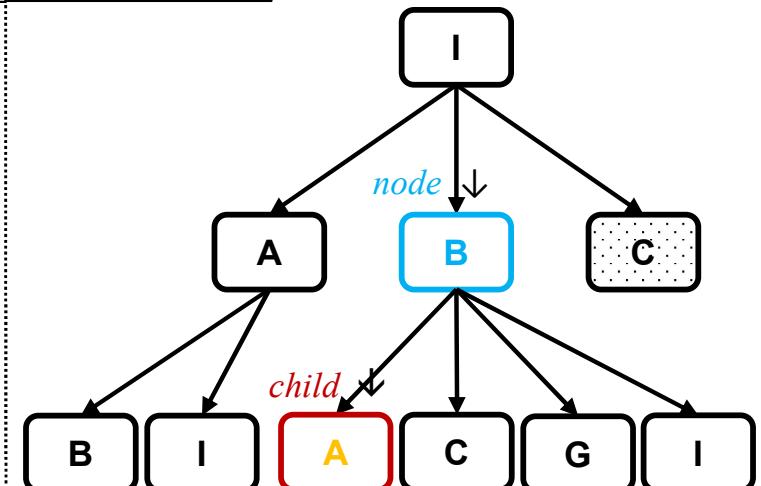
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

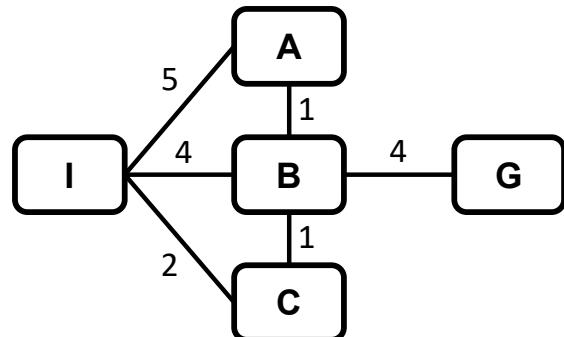
 add *child* to *frontier*

 return failure



X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

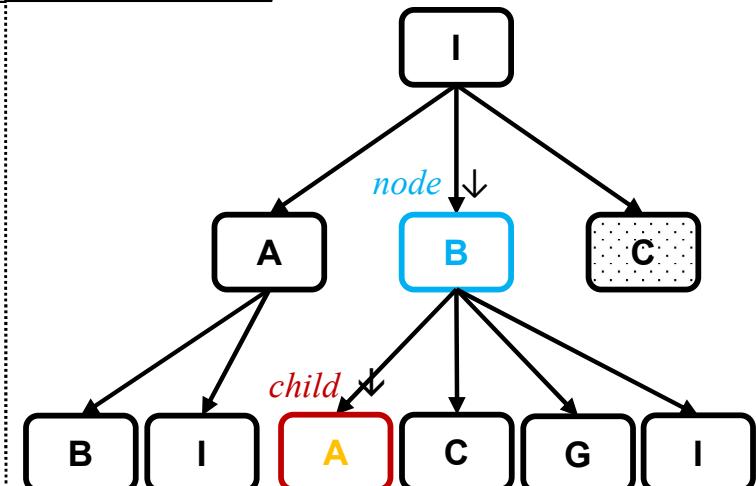
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

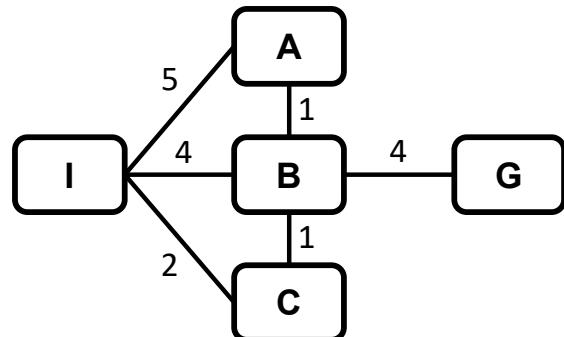
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
Node	C						
f(Node)	4						

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

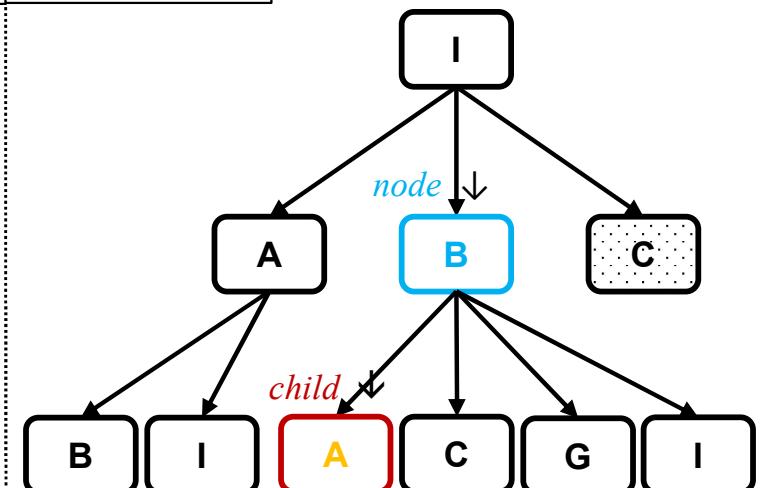
s \leftarrow *child*.STATE

 if *s* is not ~~X~~ reached or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

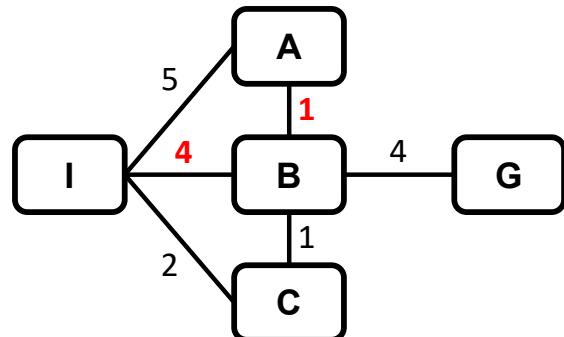
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
    node ← NODE(STATE=problem.INITIAL)
```

```
    frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
```

```
    reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
```

```
    while not IS-EMPTY(frontier) do
```

```
        node ← POP(frontier)
```

```
        if problem.IS-GOAL(node.STATE) then return node
```

```
        for each child in EXPAND(problem, node) do
```

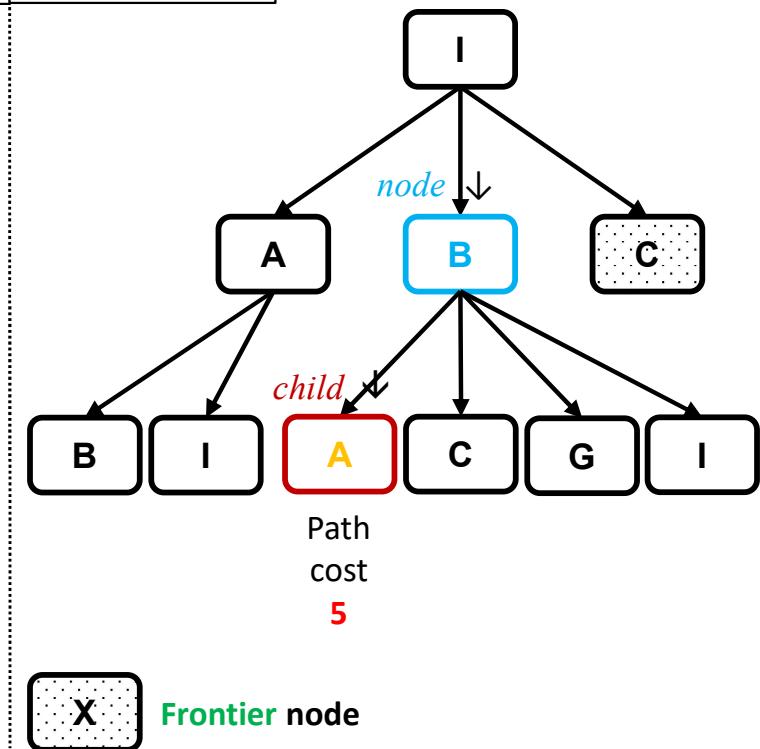
```
            s ← child.STATE
```

```
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
```

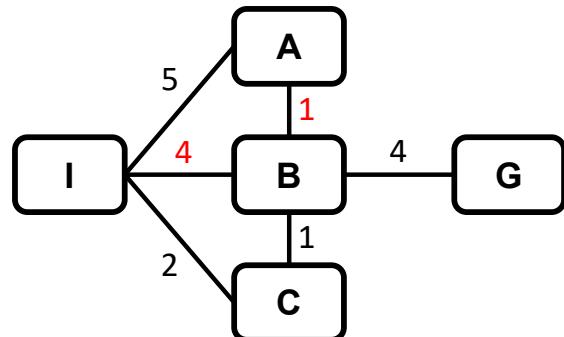
```
                reached[s] ← child
```

```
                add child to frontier
```

```
    return failure
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
    node ← NODE(STATE=problem.INITIAL)
```

```
    frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
```

```
    reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
```

```
    while not IS-EMPTY(frontier) do
```

```
        node ← POP(frontier)
```

```
        if problem.IS-GOAL(node.STATE) then return node
```

```
        for each child in EXPAND(problem, node) do
```

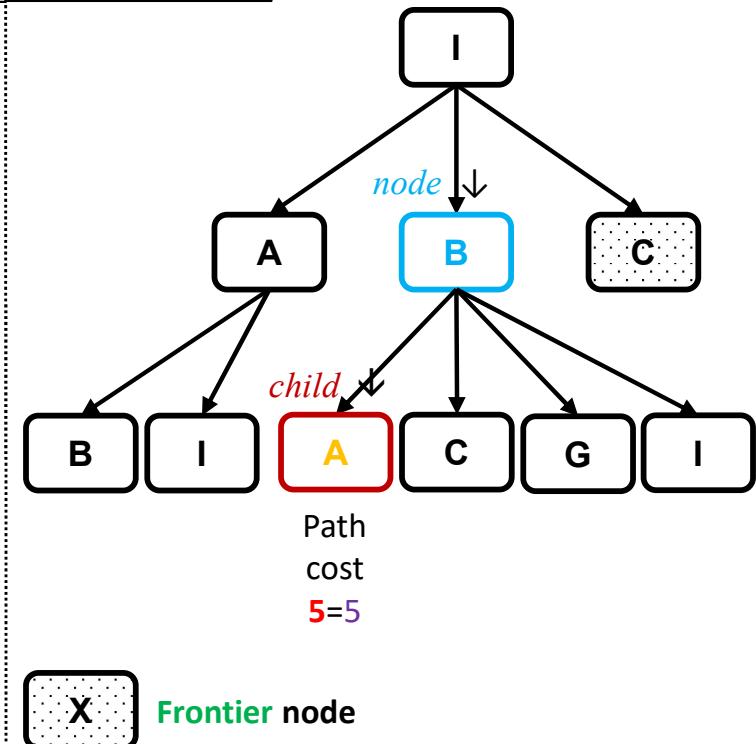
```
            s ← child.STATE
```

```
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
```

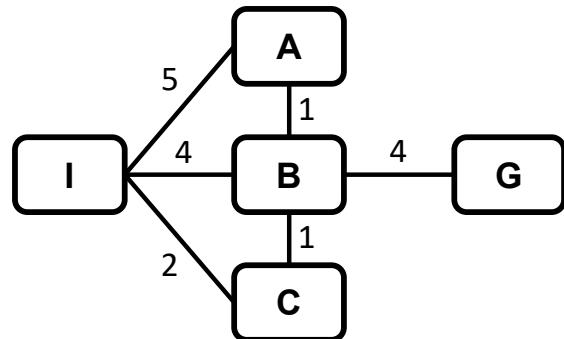
```
                reached[s] ← child
```

```
                add child to frontier
```

```
    return failure
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

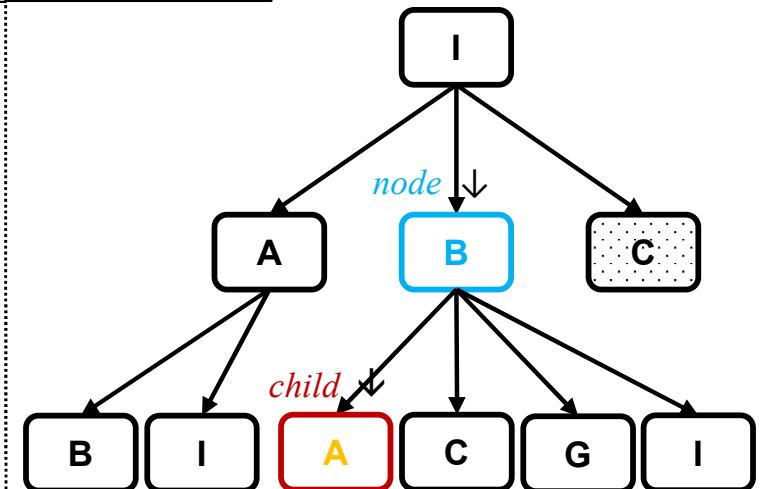
Frontier	Parent	I					
Node	C						
f(Node)	4						

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

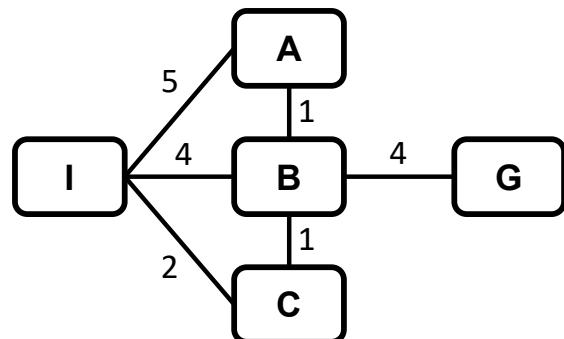
```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
  
```



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

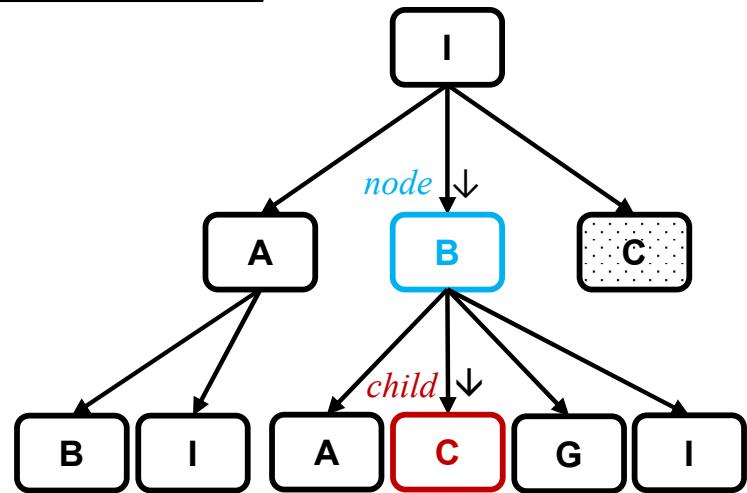
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

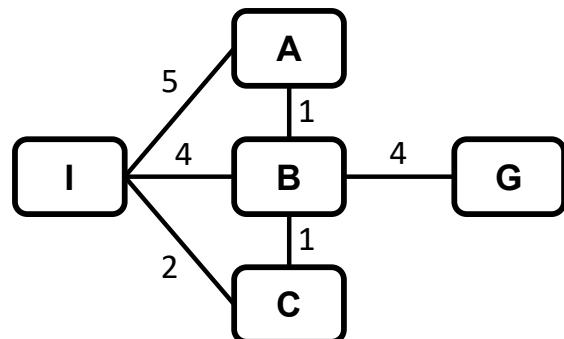
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

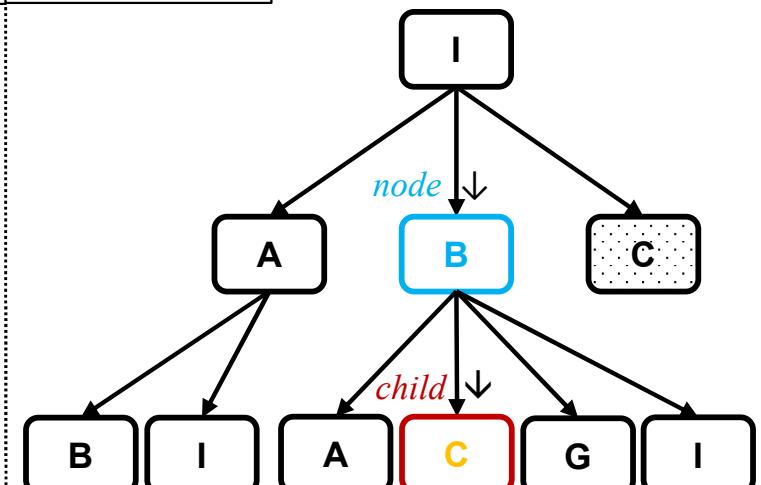
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

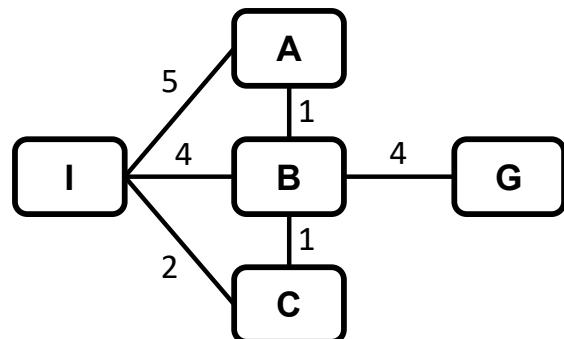
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	$f(\text{Node})$	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
    node ← NODE(STATE=problem.INITIAL)
```

```
    frontier ← a priority queue ordered by f, with node as an element
```

```
    reached ← a lookup table, with one entry key problem.INITIAL and value node
```

```
    while not IS-EMPTY(frontier) do
```

```
        node ← POP(frontier)
```

```
        if problem.IS-GOAL(node.STATE) then return node
```

```
        for each child in EXPAND(problem, node) do
```

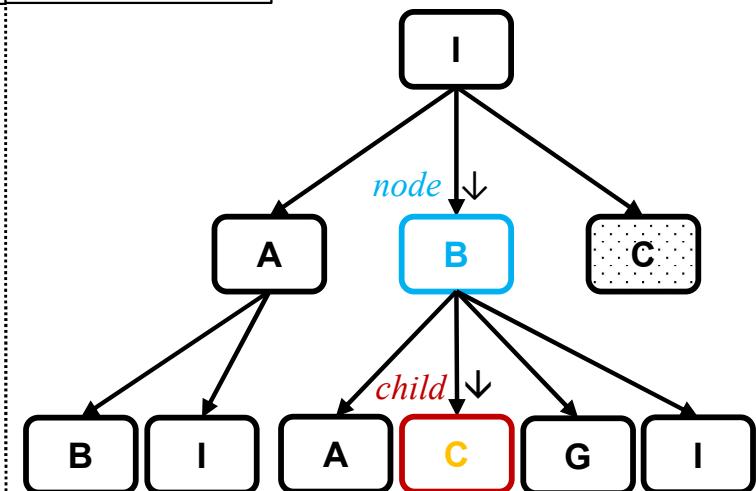
```
            s ← child.STATE
```

```
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
```

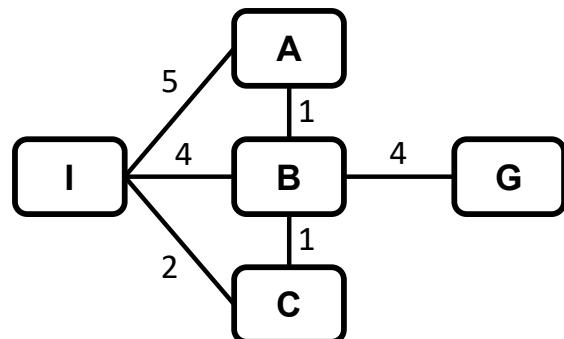
```
                reached[s] ← child
```

```
                add child to frontier
```

```
    return failure
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
Node	C						
f(Node)	4						

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

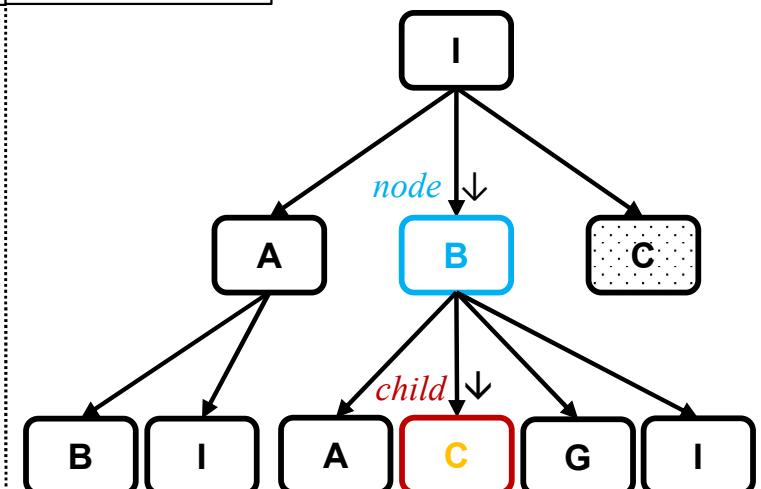
s \leftarrow *child*.STATE

 if *s* is not ~~X~~ reached or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

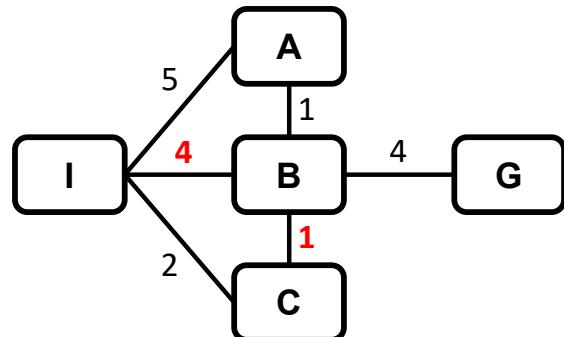
 add *child* to *frontier*

 return failure



Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

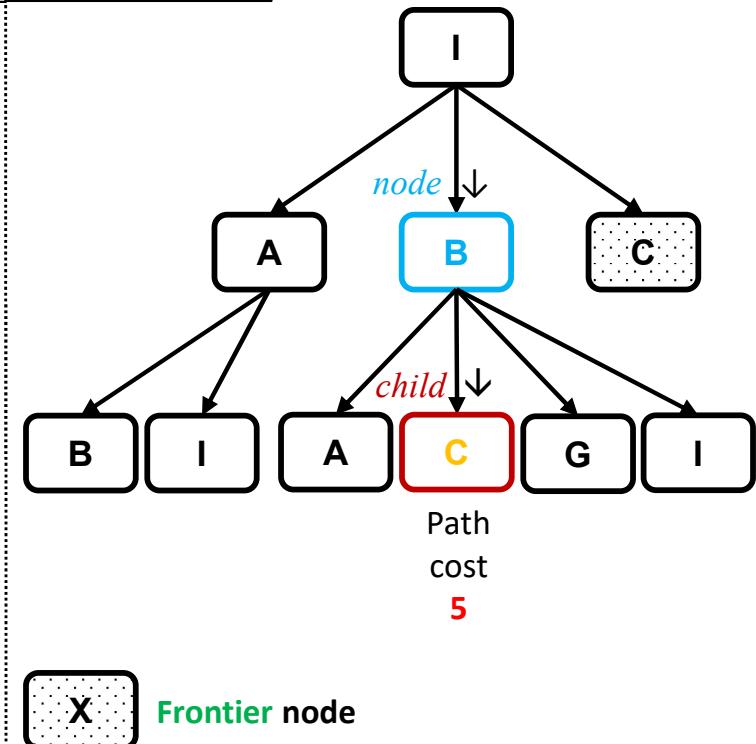
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~< i>reached[i].PATH-COST~~ then

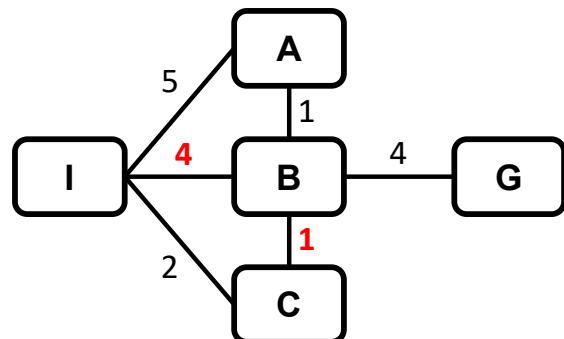
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
Node	C						
f(Node)	4						

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

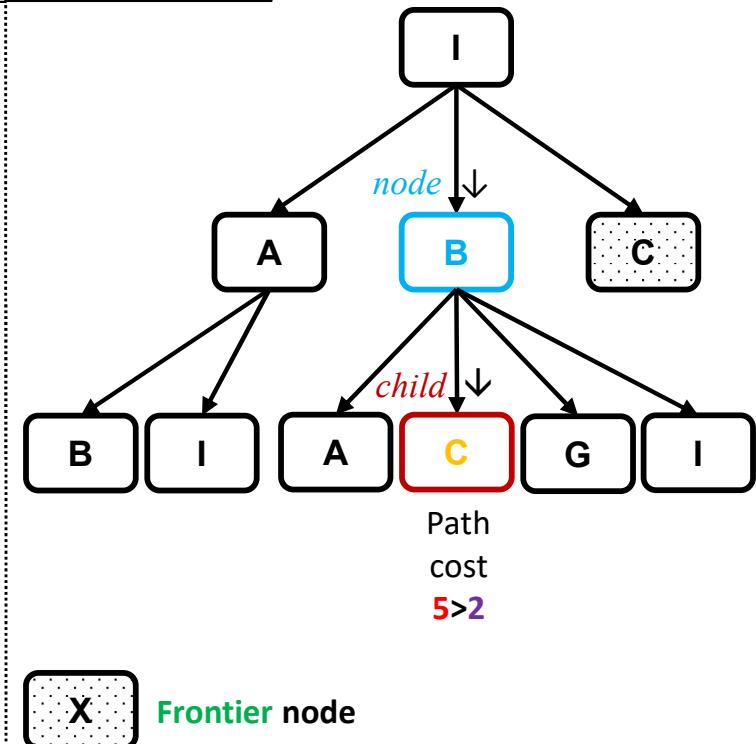
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<~~ *reached*[*s*].PATH-COST then

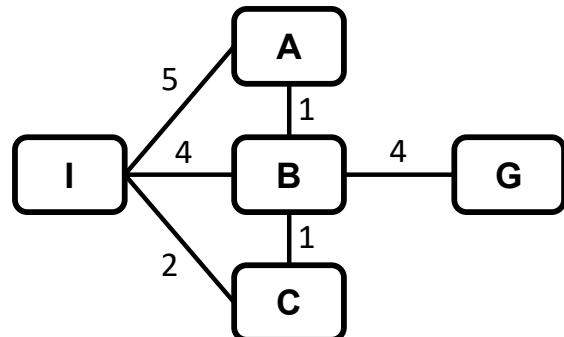
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
    node ← NODE(STATE=problem.INITIAL)
```

```
    frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
```

```
    reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
```

```
    while not IS-EMPTY(frontier) do
```

```
        node ← POP(frontier)
```

```
        if problem.IS-GOAL(node.STATE) then return node
```

```
        for each child in EXPAND(problem, node) do
```

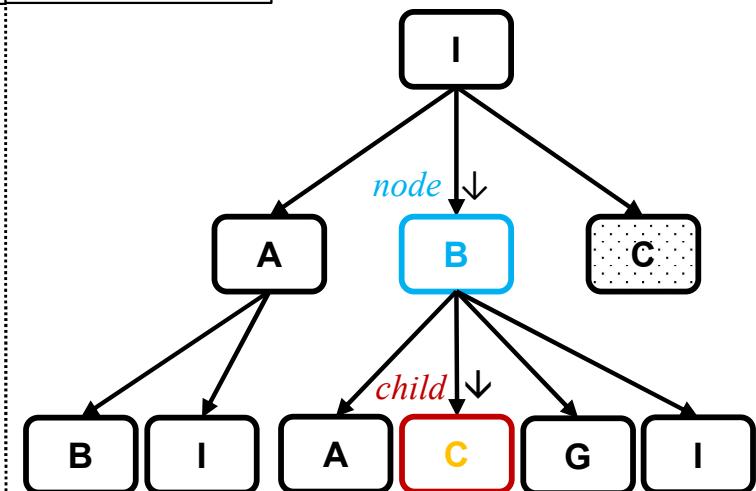
```
            s ← child.STATE
```

```
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
```

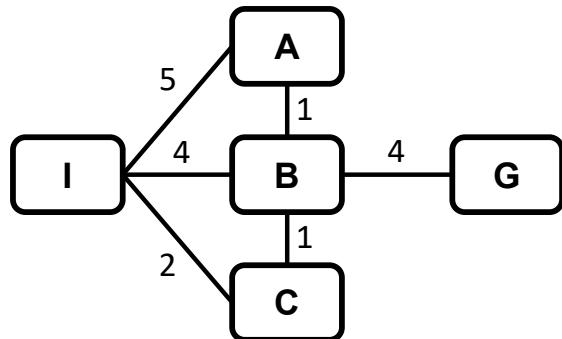
```
                reached[s] ← child
```

```
                add child to frontier
```

```
    return failure
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

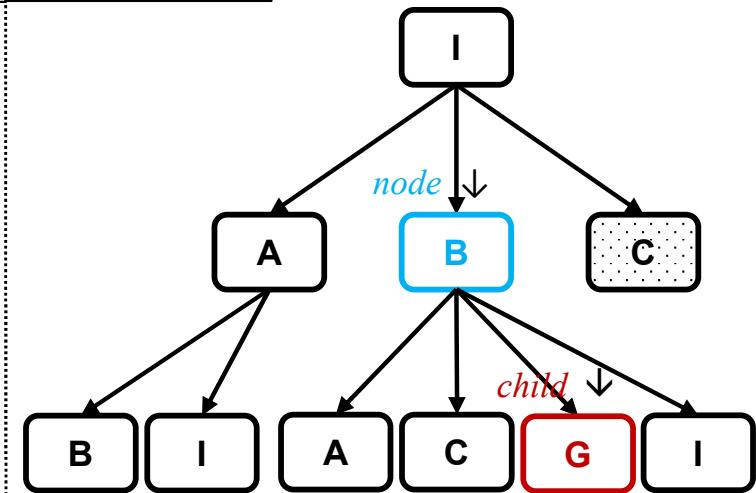
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

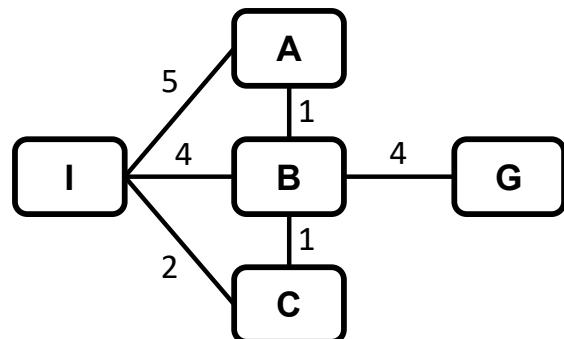
 add *child* to *frontier*

 return failure



X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

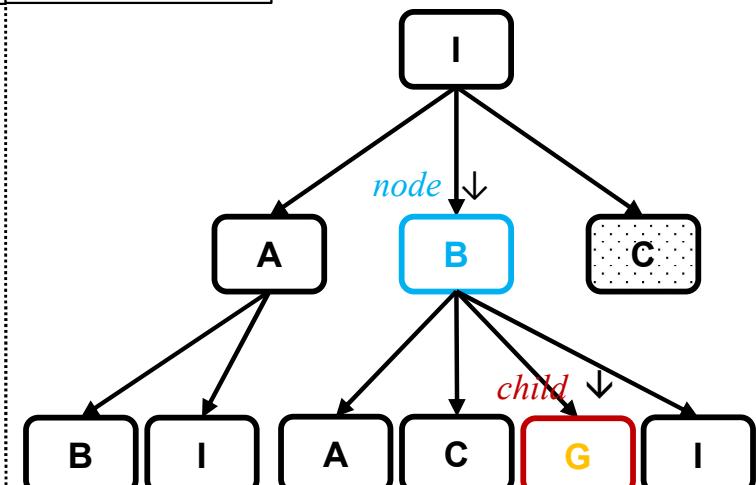
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

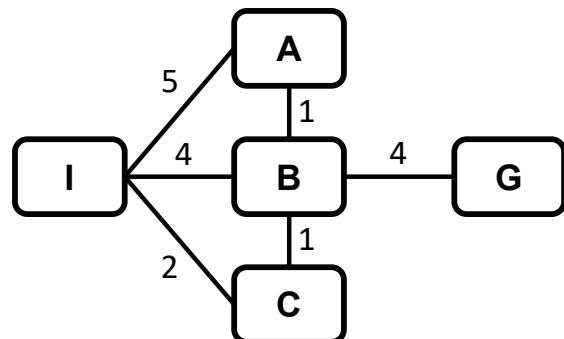
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
    node ← NODE(STATE=problem.INITIAL)
```

```
    frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
```

```
    reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
```

```
    while not IS-EMPTY(frontier) do
```

```
        node ← POP(frontier)
```

```
        if problem.IS-GOAL(node.STATE) then return node
```

```
        for each child in EXPAND(problem, node) do
```

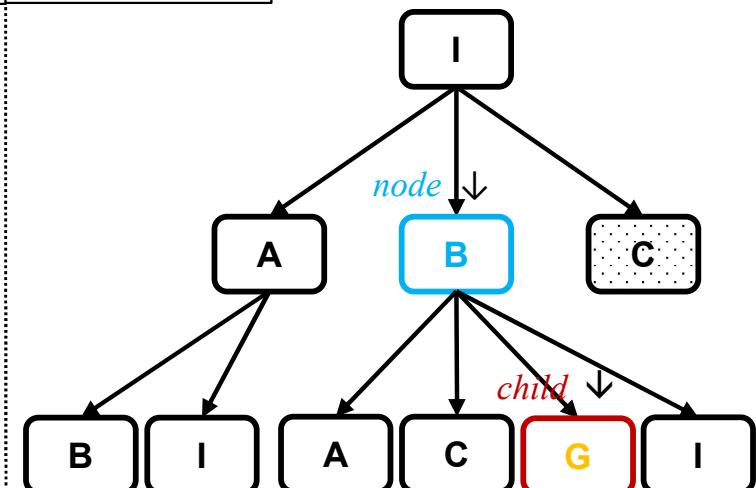
```
            s ← child.STATE
```

```
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
```

```
                reached[s] ← child
```

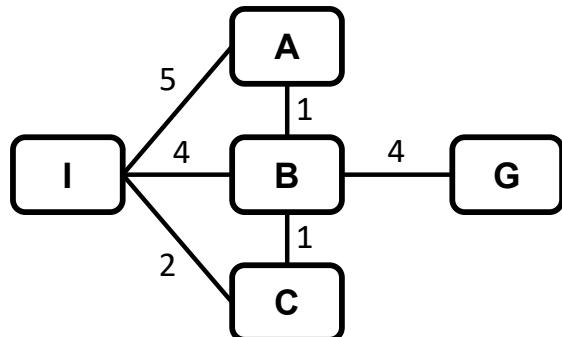
```
                add child to frontier
```

```
    return failure
```



X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

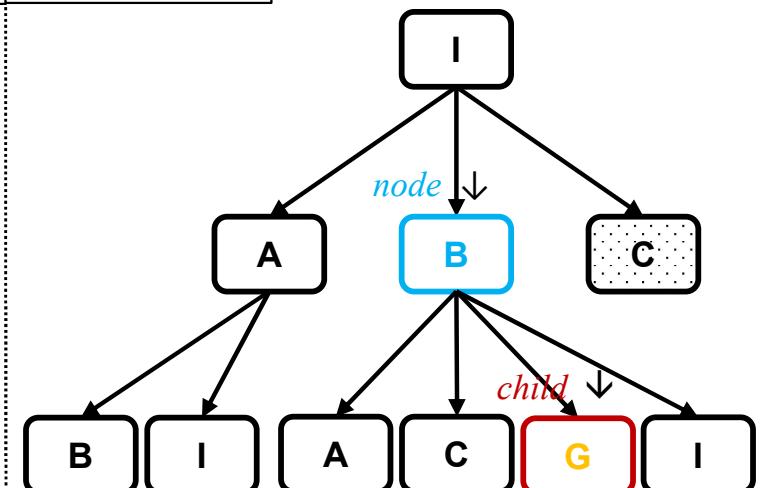
Frontier	Parent	I					
Node	C						
f(Node)	4						

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

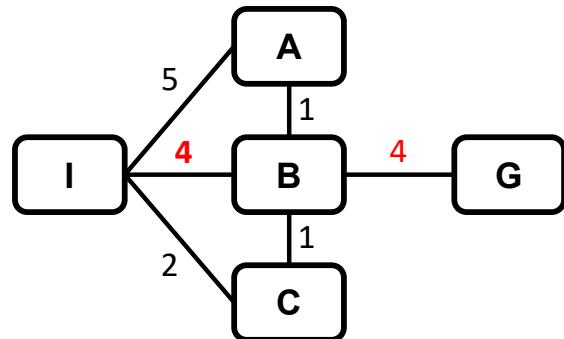
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
       $s \leftarrow child.STATE$ 
      if  $s$  is not in reached or  $child.PATH-COST < reached[s].PATH-COST$  then
        reached[ $s$ ] ← child
        add child to frontier
  return failure
  
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
Frontier	Node	C					
Frontier	f(Node)	4					

Reached	Parent	---	I	I	I	B	
Reached	Key/State	I	A	B	C	G	
Reached	Path cost	0	5	4	2	8	

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

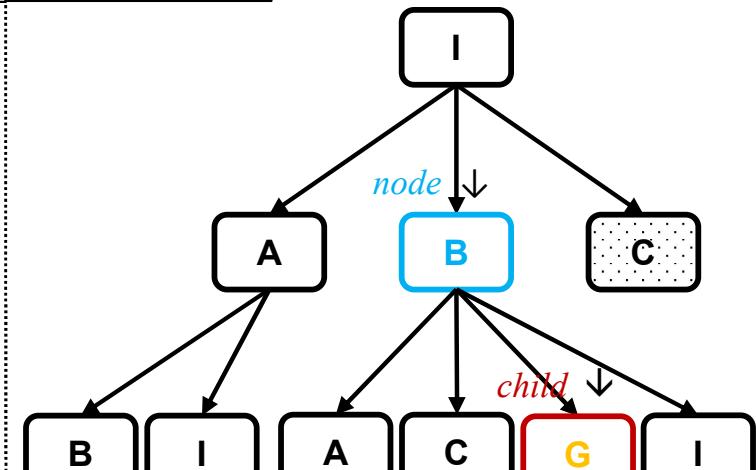
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

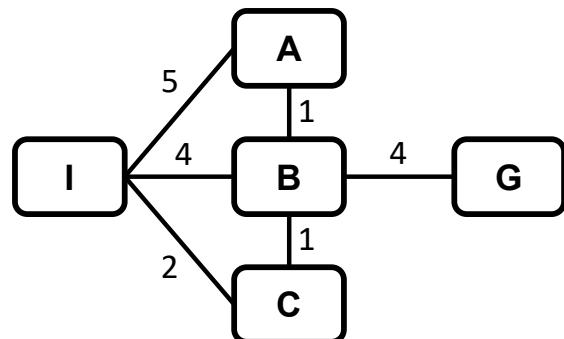
 return failure



Path cost
8

X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
	Node	G	C				
	f(Node)	0	4				

Reached	Parent	---	I	I	I	B	
	Key/State	I	A	B	C	G	
	Path cost	0	5	4	2	8	

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

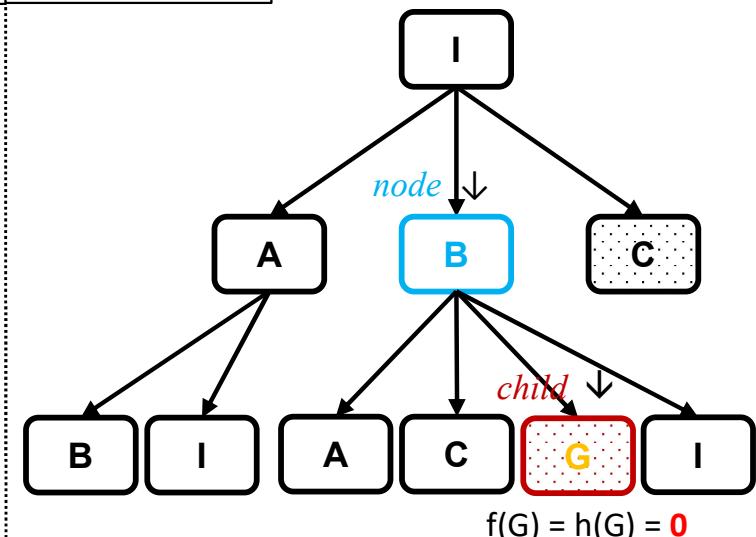
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

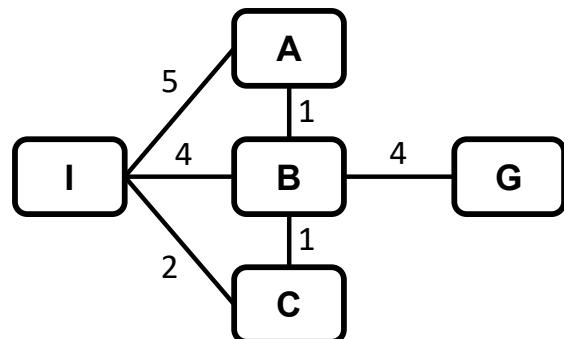
 add *child* to *frontier*

 return failure



X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
    node ← NODE(STATE=problem.INITIAL)
```

```
    frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
```

```
    reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
```

```
    while not IS-EMPTY(frontier) do
```

```
        node ← POP(frontier)
```

```
        if problem.IS-GOAL(node.STATE) then return  $node$ 
```

```
        for each child in EXPAND(problem, node) do
```

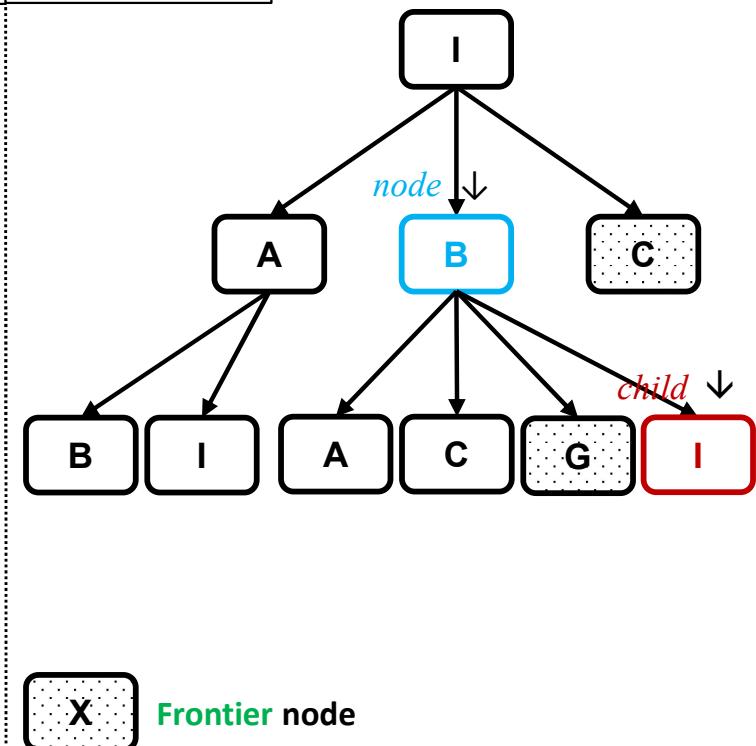
```
            s ← child.STATE
```

```
            if  $s$  is not in reached or child.PATH-COST < reached[ $s$ ].PATH-COST then
```

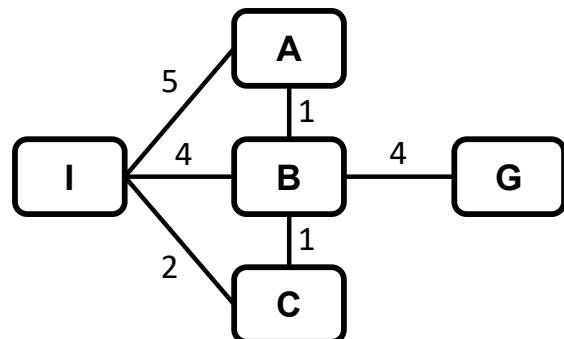
```
                reached[ $s$ ] ← child
```

```
                add child to frontier
```

```
    return failure
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	4	2	8			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

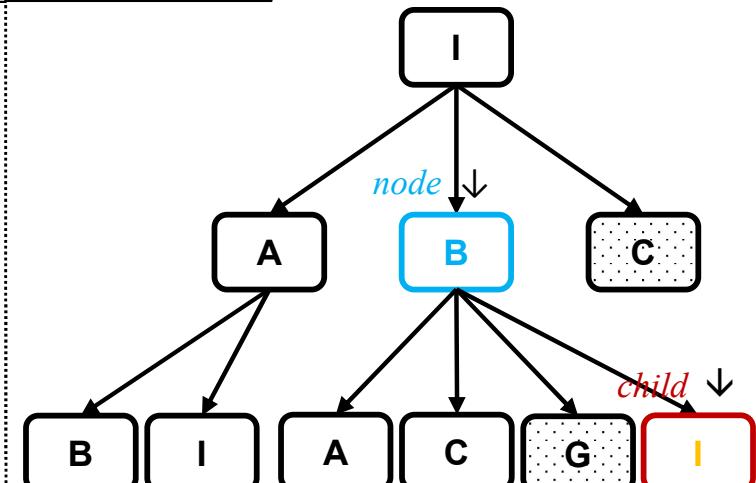
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

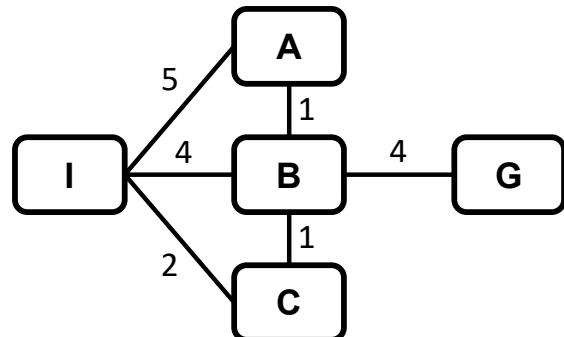
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

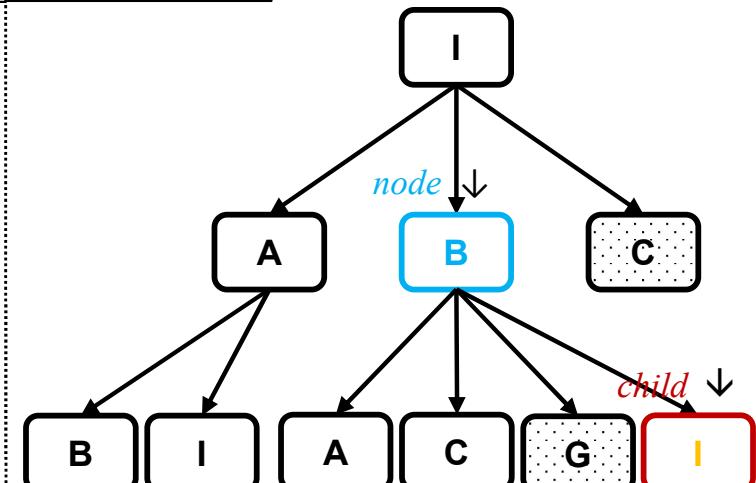
Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	4	2	8			

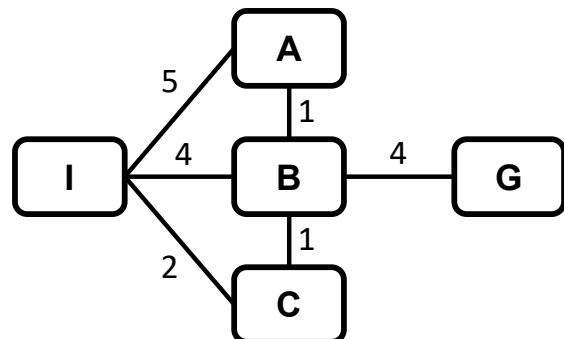
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
  
```



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	4	2	8			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

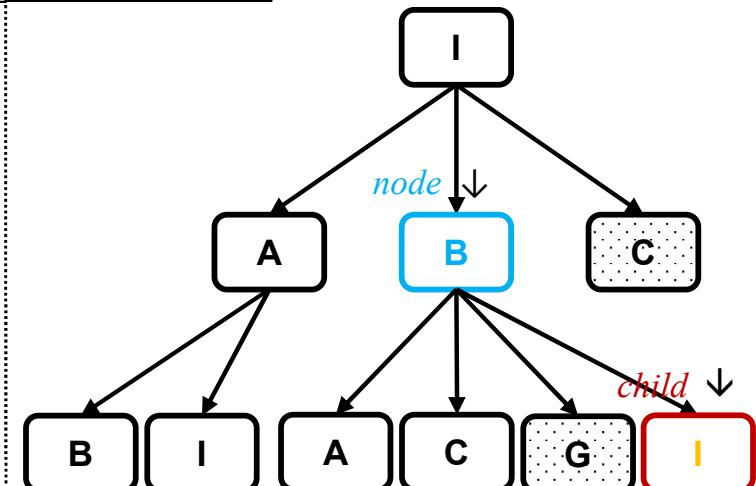
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

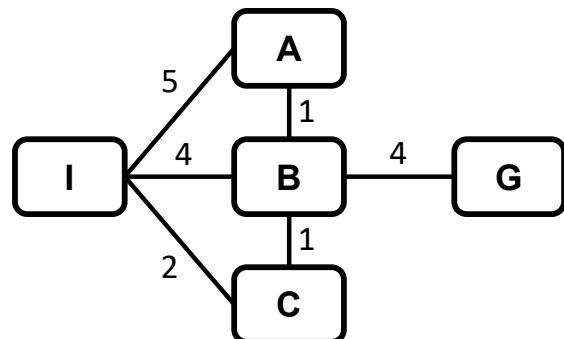
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

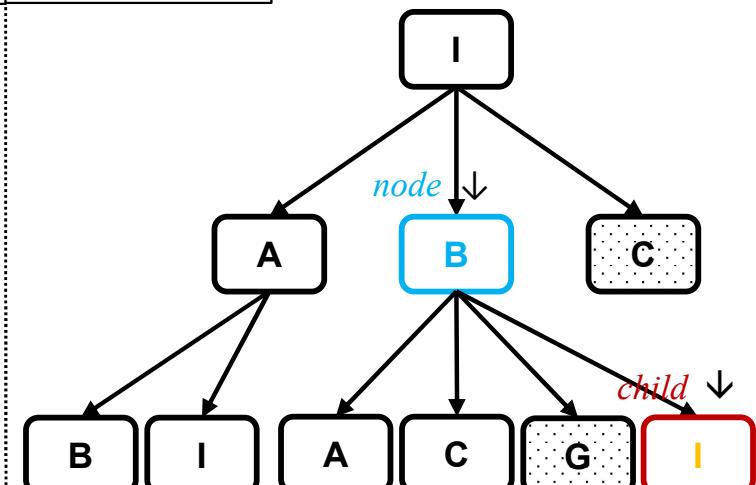
s \leftarrow *child*.STATE

 if *s* is not ~~X~~ reached or *child*.PATH-COST < *reached*[*s*].PATH-COST then

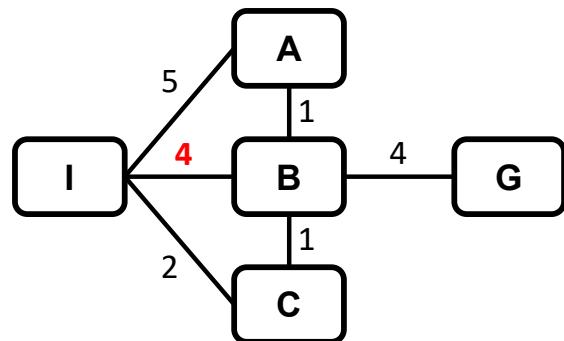
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

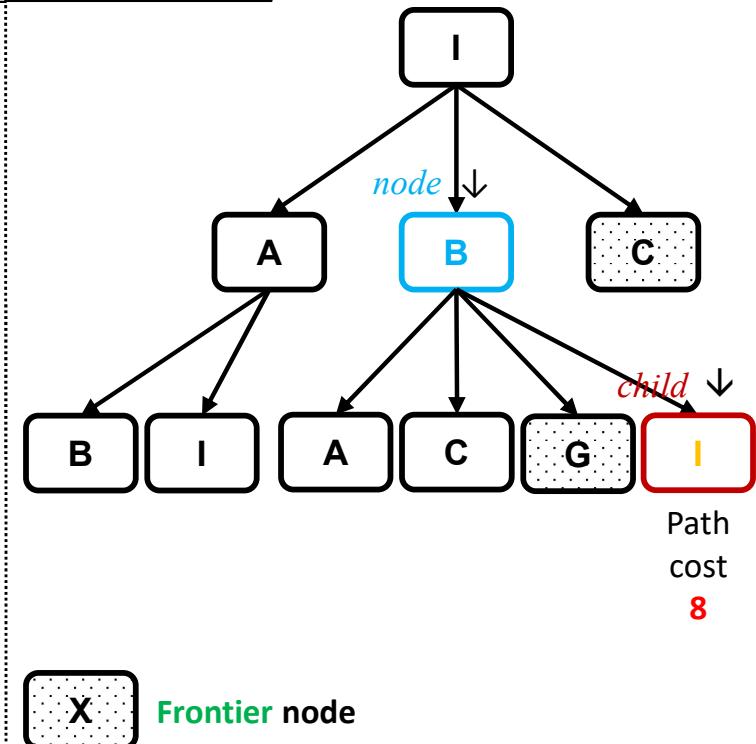
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~< i>reached~~[*s*].PATH-COST then

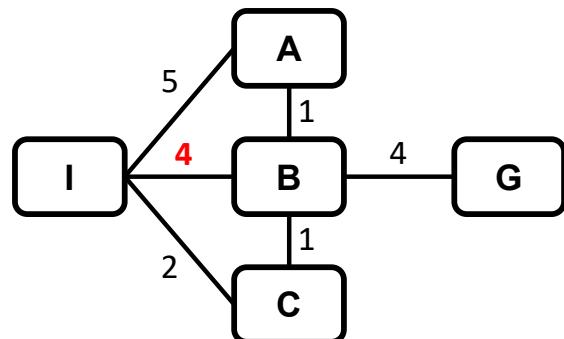
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

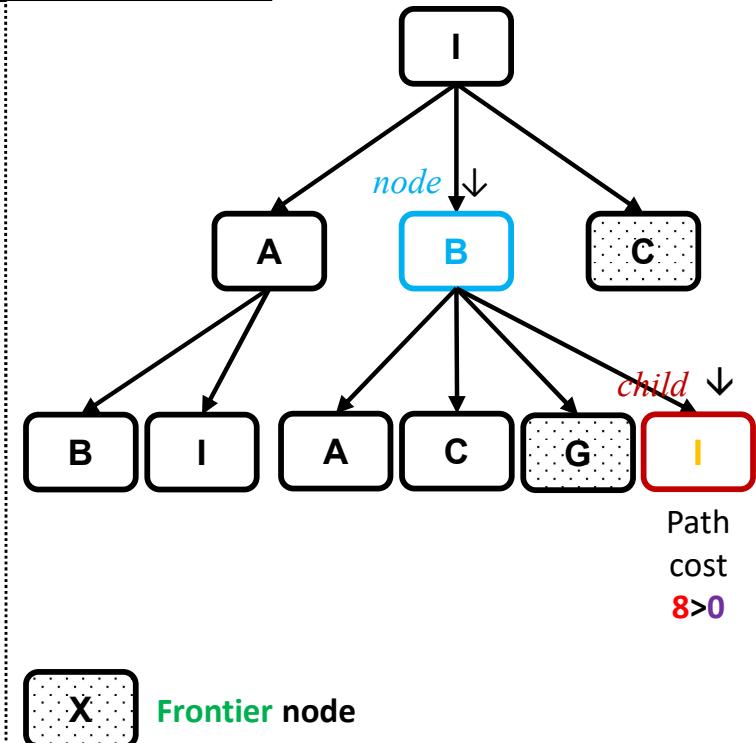
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

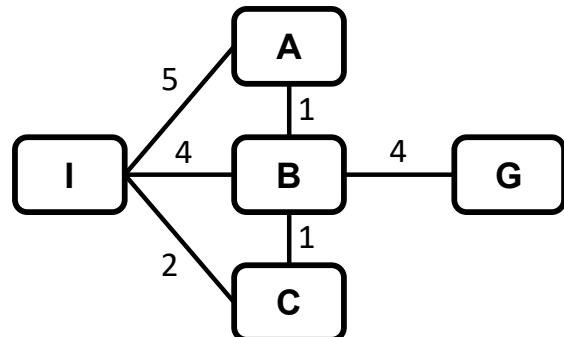
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	4	2	8			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

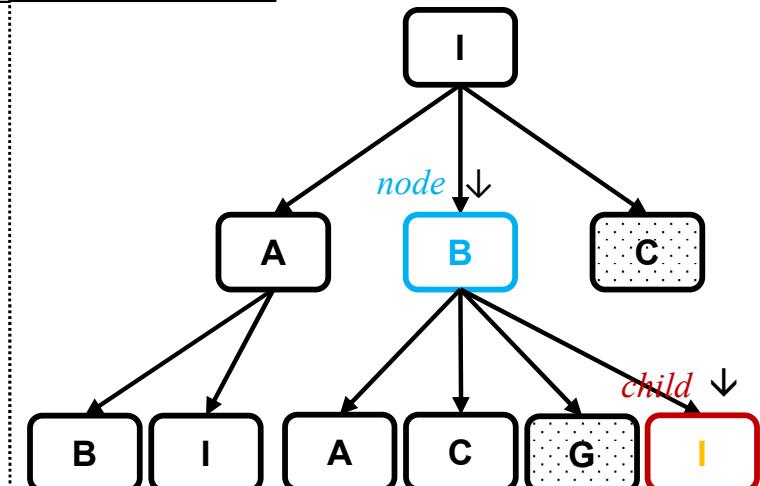
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<~~ *reached*[*s*].PATH-COST then

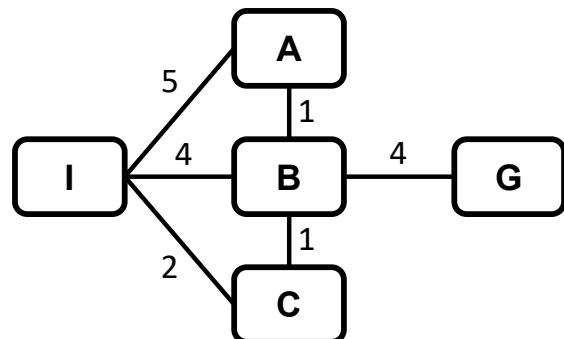
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
Node	G	C					
f(Node)	0	4					

Reached	Parent	---	I	I	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

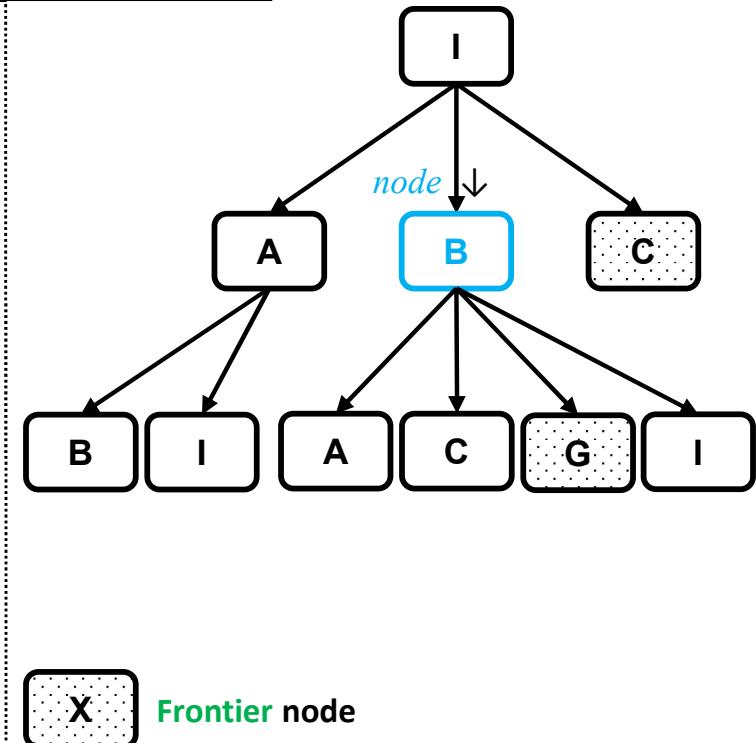
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

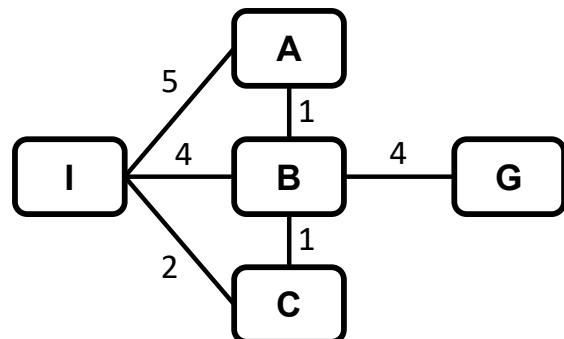
 add *child* to *frontier*

 return failure



X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I				
	Node	G	C				
	f(Node)	0	4				

Reached	Parent	---	I	I	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

 NOT EMPTY!

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

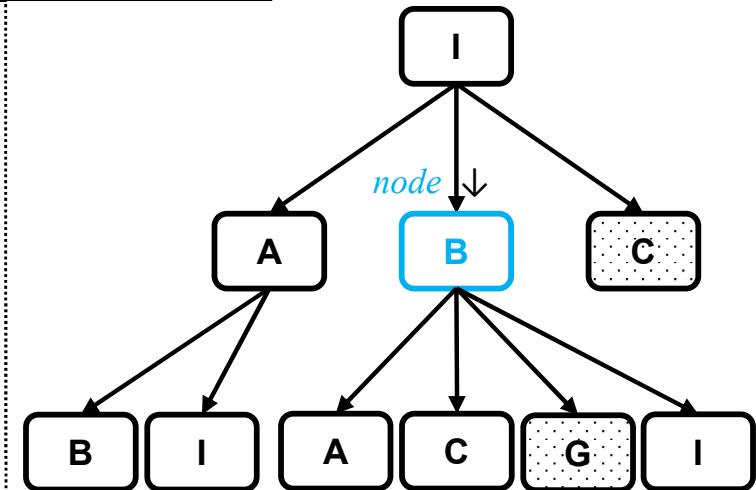
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

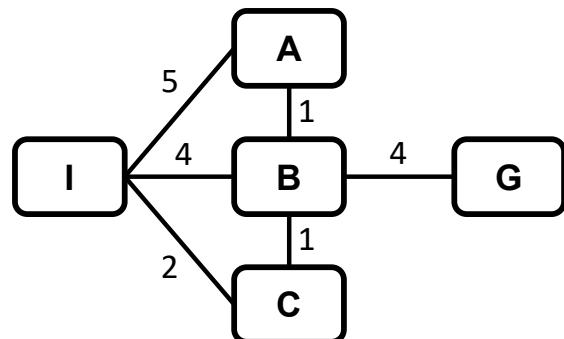
 add *child* to *frontier*

 return failure



X Frontier node

Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

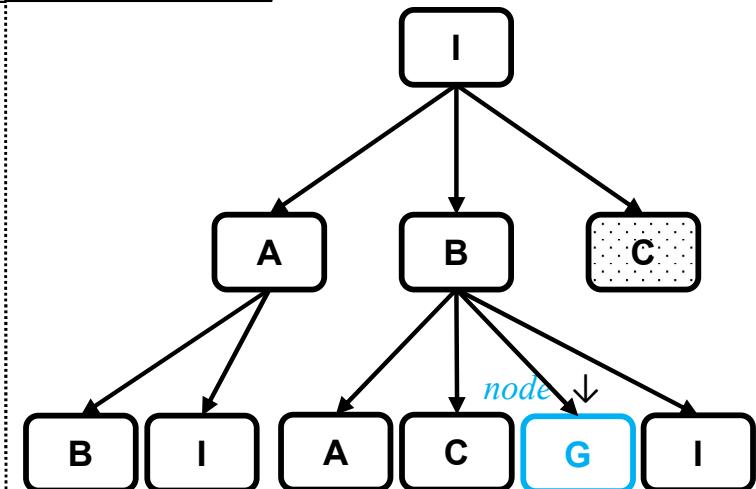
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

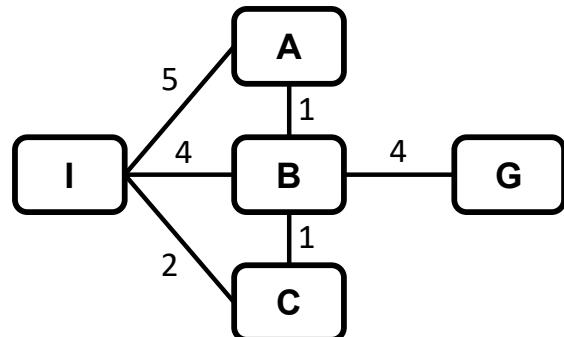
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

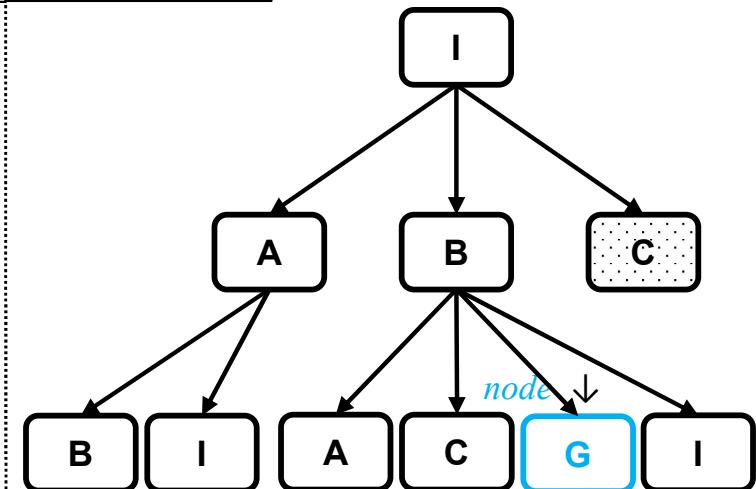
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

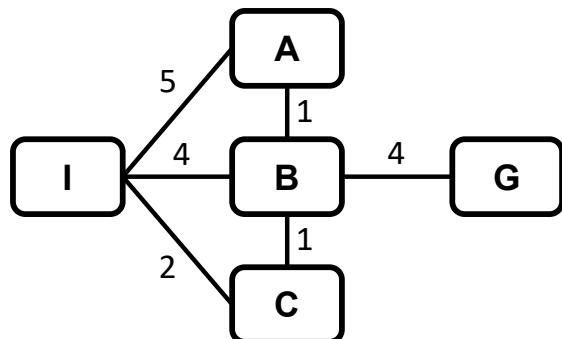
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	5	4	2	8		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node* TRUE!

 for each *child* in EXPAND(*problem*, *node*) do

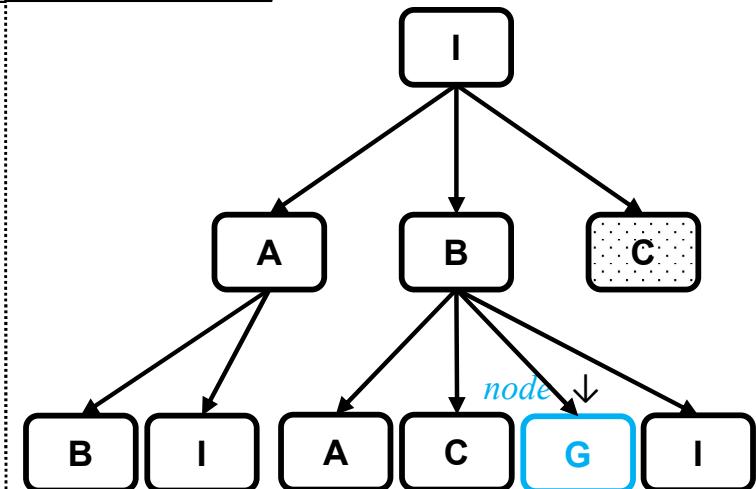
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

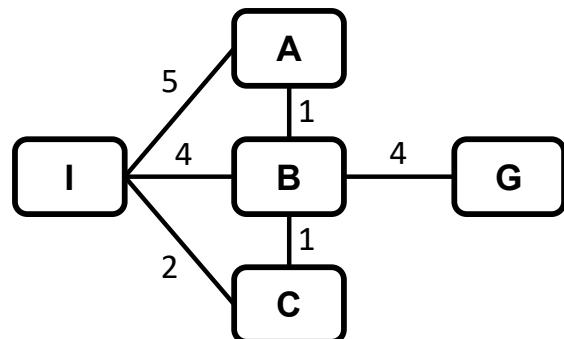
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



Greedy Best First Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

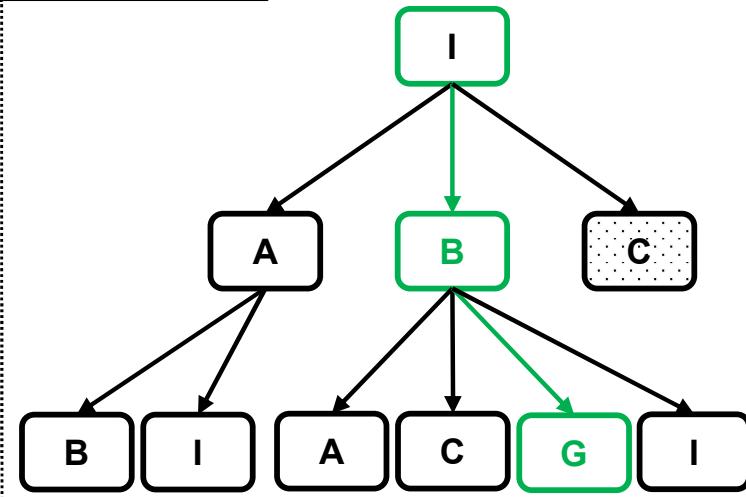
Frontier	Parent	I					
	Node	C					
	f(Node)	4					

Reached	Parent	---	I	I	I	B	
	Key/State	I	A	B	C	G	
	Path cost	0	5	4	2	8	

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node TRUE!
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
  
```



X Frontier node

A* Algorithm: Evaluation Function

Calculate / obtain:

$$f(n) = g(\text{State}_n) + h(\text{State}_n)$$

where:

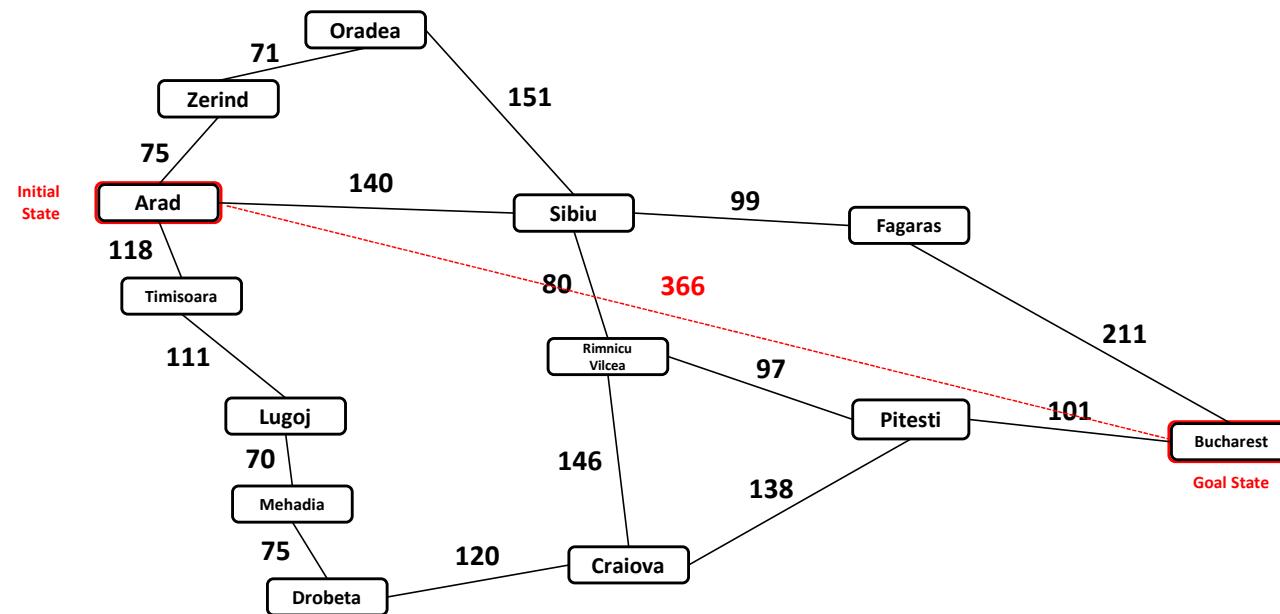
- $g(n)$ - initial node to node n path cost
- $h(n)$ - **estimated cost** of the best path that continues from node n to a goal node

A node n with minimum (maximum) $f(n)$ should be chosen for expansion

Romanian Roadtrip: Heuristics $h(n)$

For this particular problem the heuristic function $h(n)$ is defined by a **straight-line (Euclidean) distance** between two states (cities).

“As the crows flies” in other words.



Best-First Search: A* Pseudocode

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.Is-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

$f(n) = g(\text{State}_n) + h(\text{State}_n)$

```
function EXPAND(problem, node) yields nodes
    s  $\leftarrow$  node.STATE
    for each action in problem.ACTIONS(s) do
        s'  $\leftarrow$  problem.RESULT(s, action)
        cost  $\leftarrow$  node.PATH-COST + problem.ACTION-COST(s, action, s')
        yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Best-First Search: A* Pseudocode

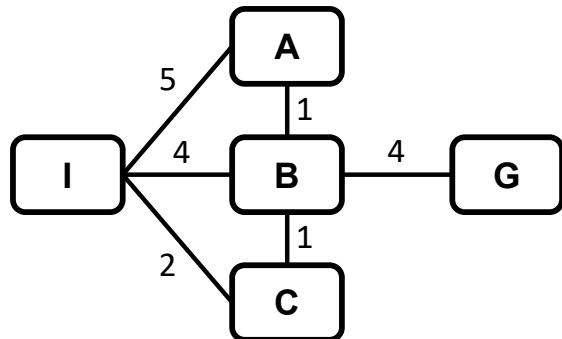
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.Is-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

$f(n) = g(\text{State}_n) + h(\text{State}_n)$

Best-First Search is really a class of search algorithms that:

- Use the evaluation function $f(n)$ to pick next action
- Keep track of visited states
- Keep track of frontier states
- Evaluation function $f(n)$ choice controls their behavior

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	f(Node)						

Reached	Parent						
	Key/State						
	Path cost						

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

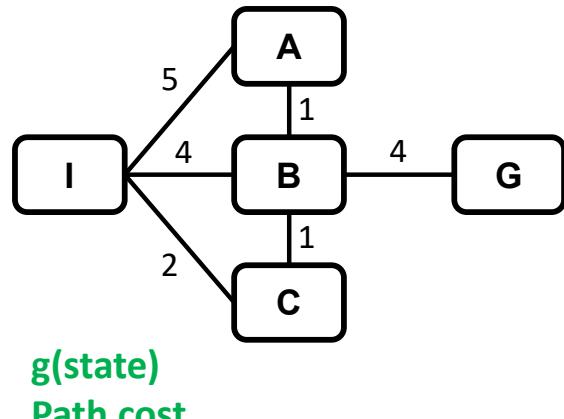
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

h(state)
Heuristic

Frontier	Parent						
	Node						
	f(Node)						

Reached	Parent						
	Key/State						
	Path cost						

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

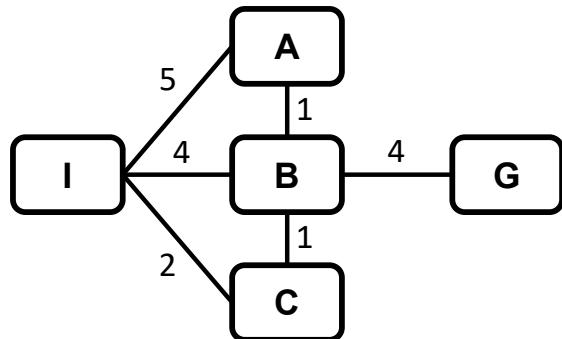
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

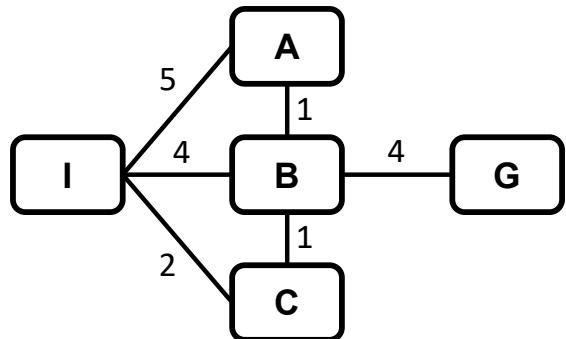
node \rightarrow

I



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
Node							
f(Node)							

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

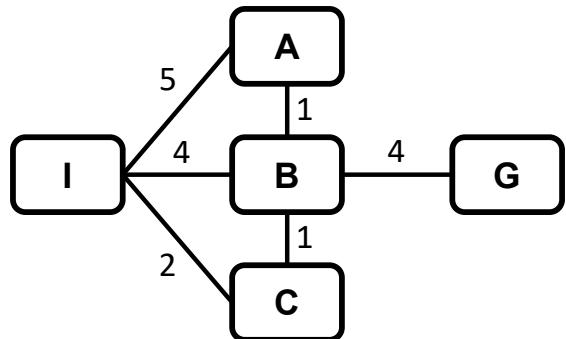
return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

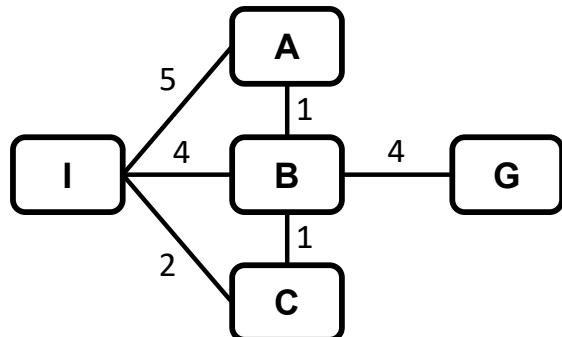
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	----						
Node	I							
f(Node)	7							

I $f(I) = g(I) + h(I) = 0 + 7 = 7$



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

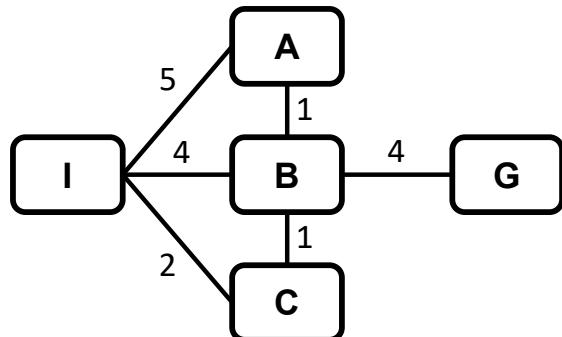
Frontier	Parent	----					
	Node	I					
	f(Node)	7					
Reached	Parent						
	Key/State						
	Path cost						

Reached	Parent						
	Key/State						
	Path cost						



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	----					
	Node	I					
	f(Node)	7					
Reached	Parent	----					
	Key/State	I					
	Path cost	0					

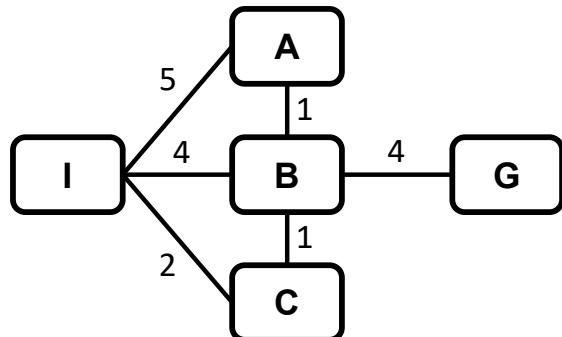


Path cost 0



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

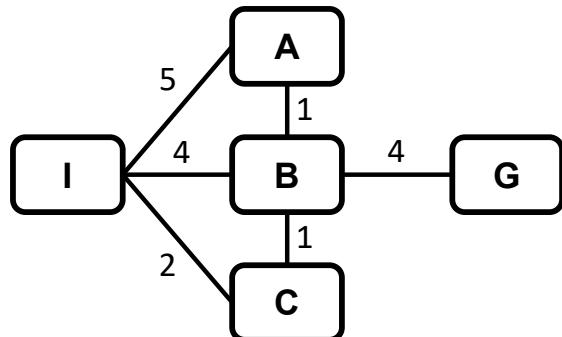
Frontier	Parent	----					
	Node	I					
	f(Node)	7					
Reached	Parent	----					
	Key/State	I					
	Path cost	0					

Reached	Parent	----					
	Key/State	I					
	Path cost	0					



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	f(Node)	7					

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

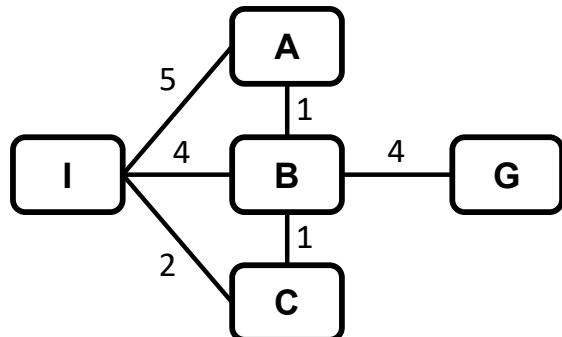
 add *child* to *frontier*

return failure



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	----					
	Node	I					
	f(Node)	7					

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

NOT EMPTY!

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

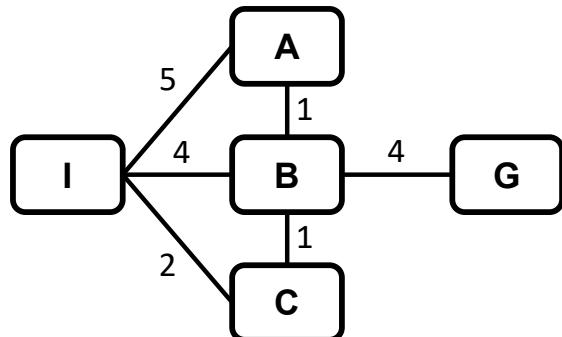
 add *child* to *frontier*

return failure



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

node \rightarrow

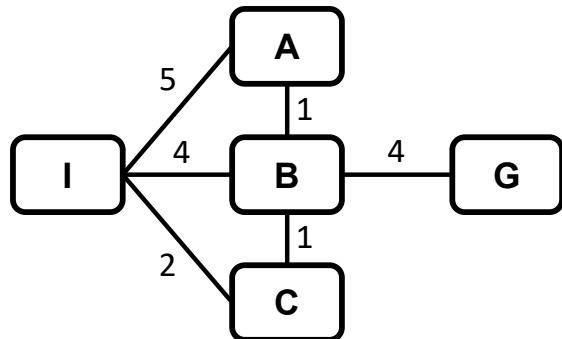


Frontier node

Frontier	Parent						
	Node						
	f(Node)						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

node \rightarrow



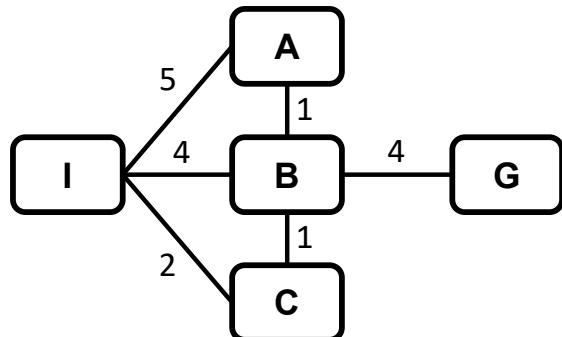
Frontier	Parent						
	Node						
	f(Node)						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent						
	Node						
	f(Node)						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

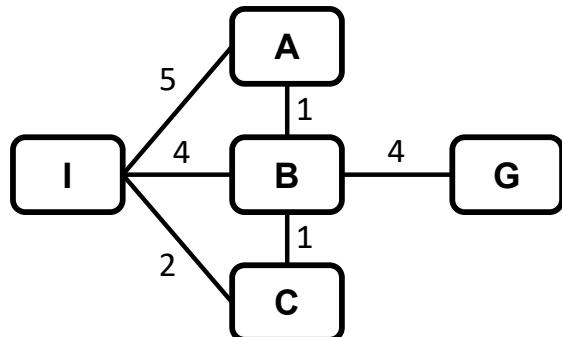
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node FALSE!
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
  
```

node →



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

node \rightarrow

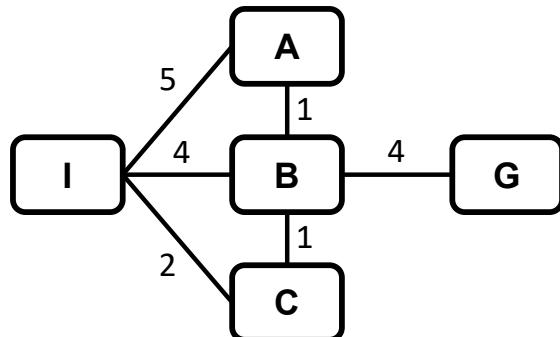


Frontier node

Frontier	Parent						
	Node						
	f(Node)						

Reached	Parent	----					
	Key/State	I					
	Path cost	0					

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

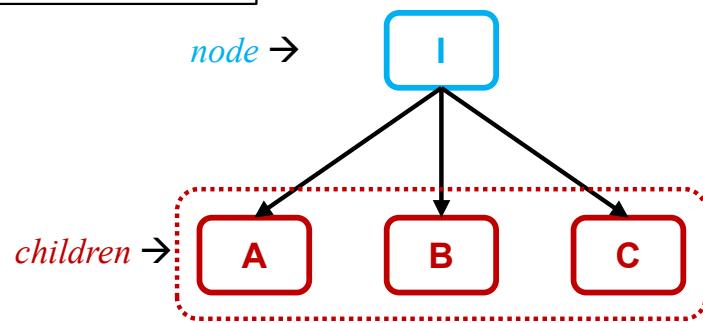
reached[s] \leftarrow *child*

 add *child* to *frontier*

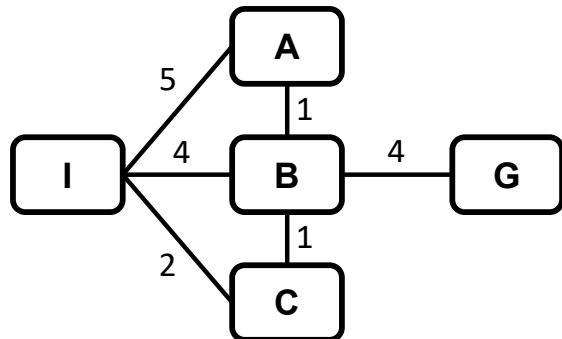
return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent						
	Node						
	$f(\text{Node})$						
Reached	Parent	---					
	Key/State	I					
	Path cost	0					



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

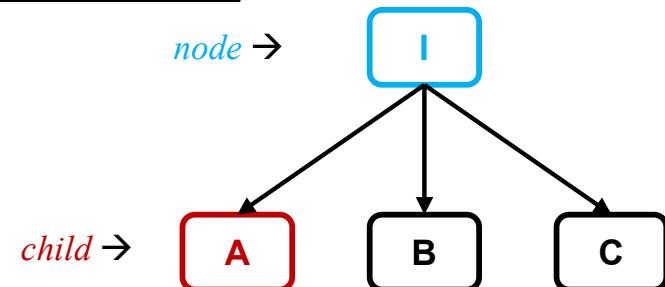
reached[s] \leftarrow *child*

 add *child* to *frontier*

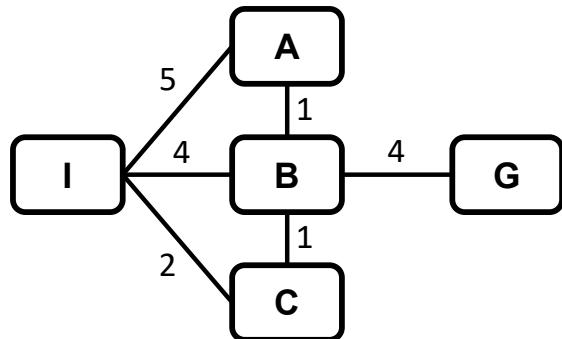
return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent						
	Node						
	$f(\text{Node})$						
Reached	Parent	----					
	Key/State	I					
	Path cost	0					



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

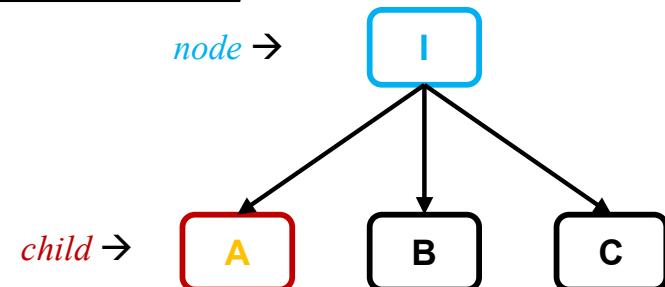
reached[s] \leftarrow *child*

 add *child* to *frontier*

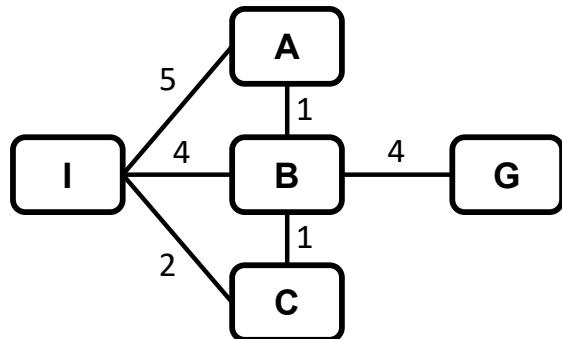
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent						
	Node						
	$f(\text{Node})$						
Reached	Parent	---					
	Key/State	I					
	Path cost	0					



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

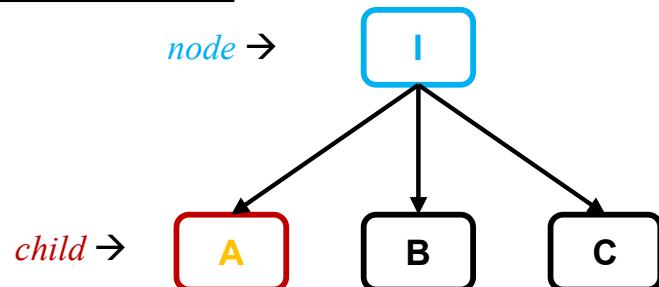
reached[s] \leftarrow *child*

 add *child* to *frontier*

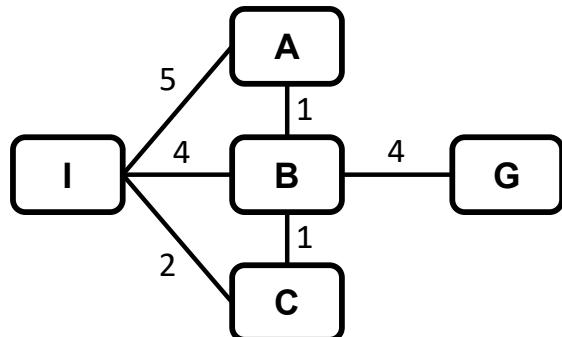
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent						
	Node						
	$f(\text{Node})$						
Reached	Parent	----					
	Key/State	I					
	Path cost	0					



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

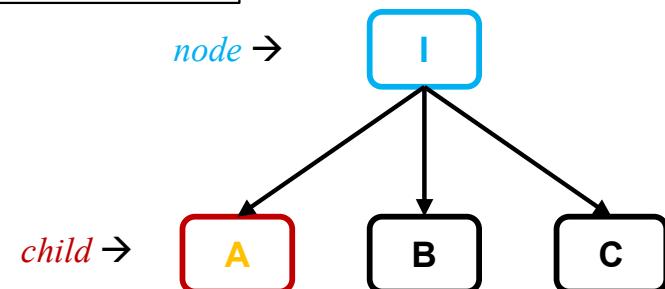
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

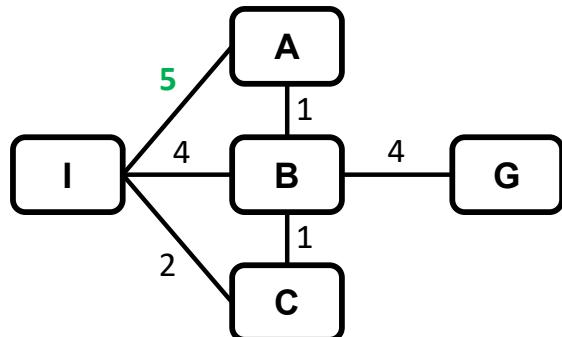
Frontier	Parent						
	Node						
	$f(\text{Node})$						
Reached	Parent	---					
	Key/State	I					
	Path cost	0					



node \rightarrow
child \rightarrow



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

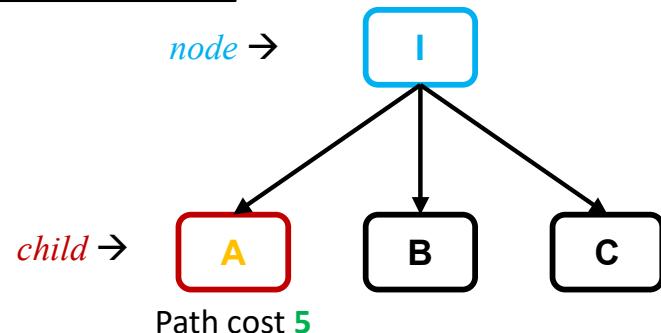
 add *child* to *frontier*

return failure

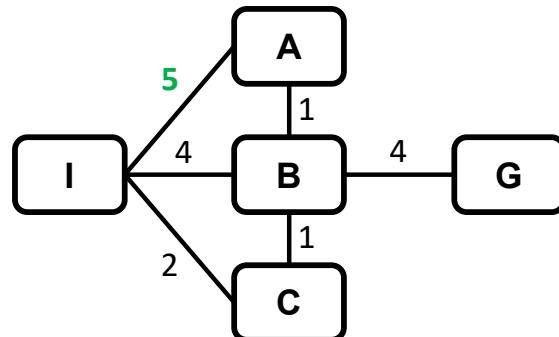
Frontier	Parent						
Node							
<i>f(Node)</i>							

Reached	Parent	---	I				
Key/State	I		A				
Path cost	0		5				

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

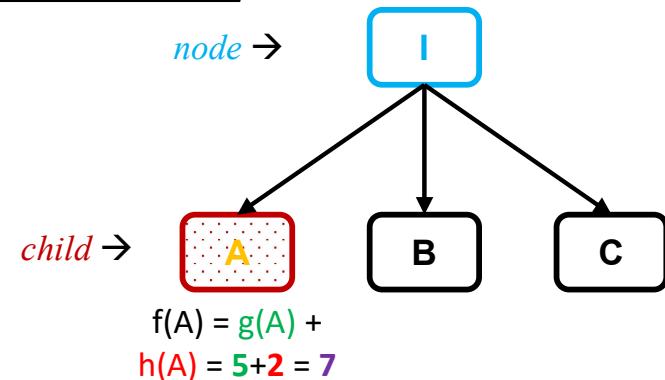
 add *child* to *frontier*

 return failure

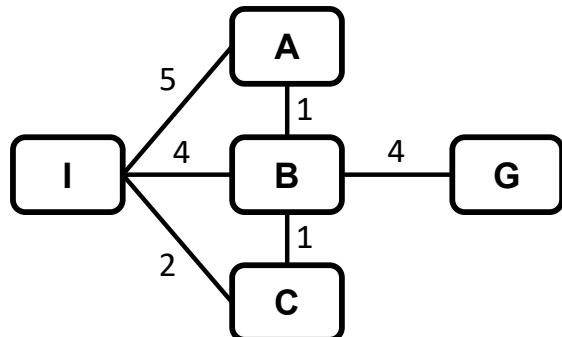
Frontier	Parent	I					
Node	A						
f(Node)	7						

Reached	Parent	---	I				
Key/State	I	A					
Path cost	0	5					

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL*(*node.STATE*) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

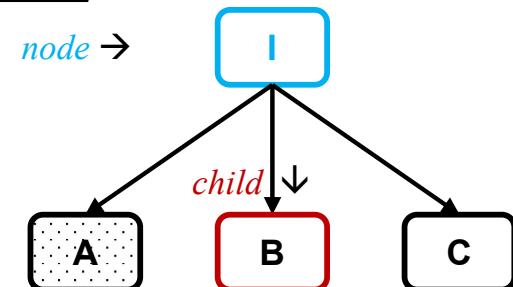
 add *child* to *frontier*

 return failure

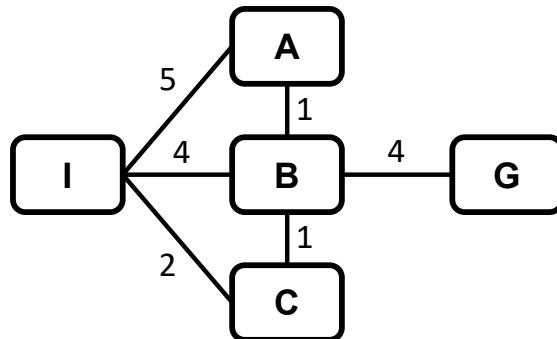
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I					
	Node	A					
	f(Node)	7					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

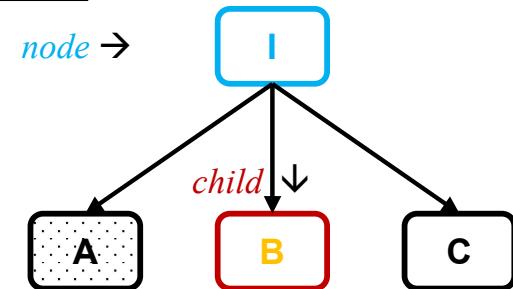
 add *child* to *frontier*

return failure

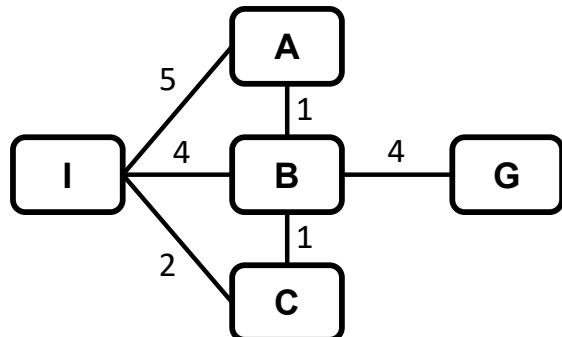
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I					
	Node	A					
	f(Node)	7					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

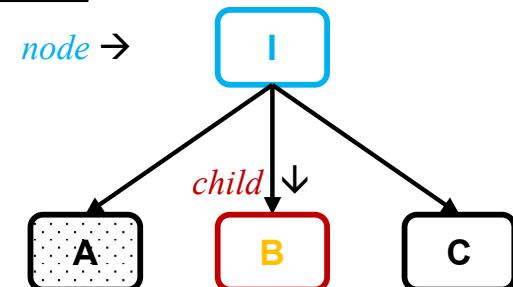
reached[s] \leftarrow *child*

 add *child* to *frontier*

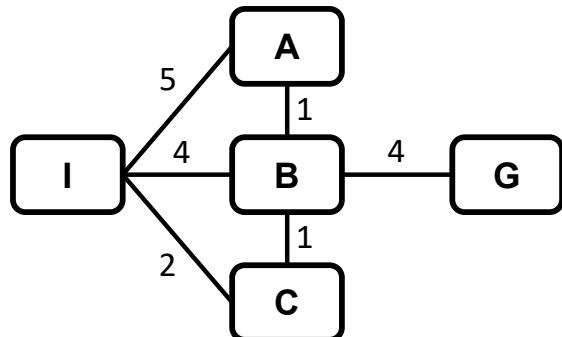
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I					
	Node	A					
	<i>f</i> (Node)	7					
Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

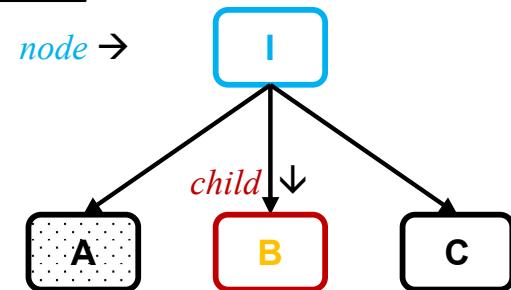
 add *child* to *frontier*

return failure

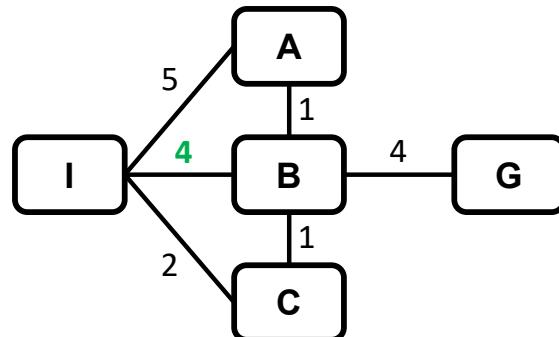
Frontier	Parent	I					
	Node	A					
	f(Node)	7					

Reached	Parent	---	I				
	Key/State	I	A				
	Path cost	0	5				

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

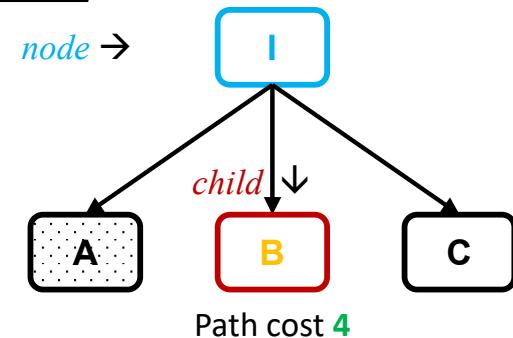
 add *child* to *frontier*

return failure

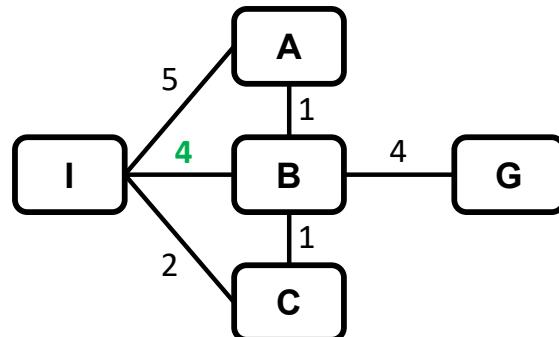
Frontier	Parent	I					
	Node	A					
	<i>f</i> (Node)	7					

Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

reached[s] \leftarrow *child*

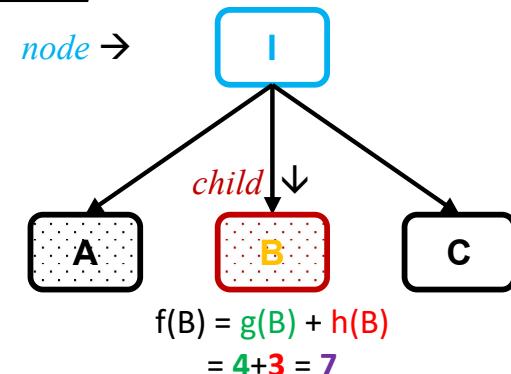
 add *child* to *frontier*

 return failure

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

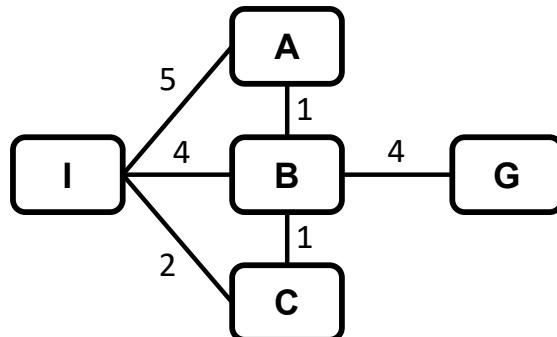
Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



X Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

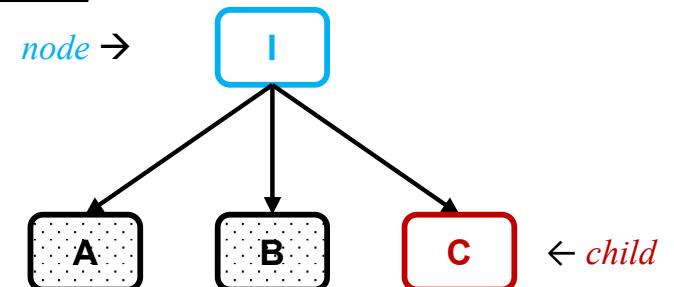
reached[s] \leftarrow *child*

 add *child* to *frontier*

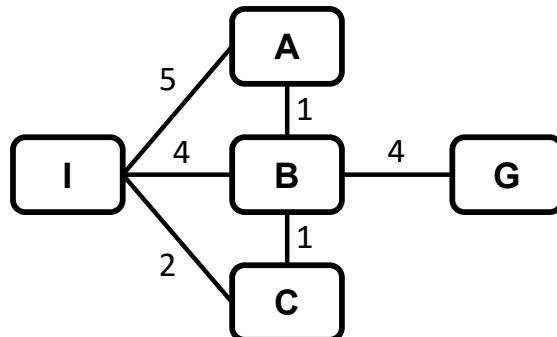
return failure

State Space Graph		Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	<i>f</i> (Node)	7	7				
Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

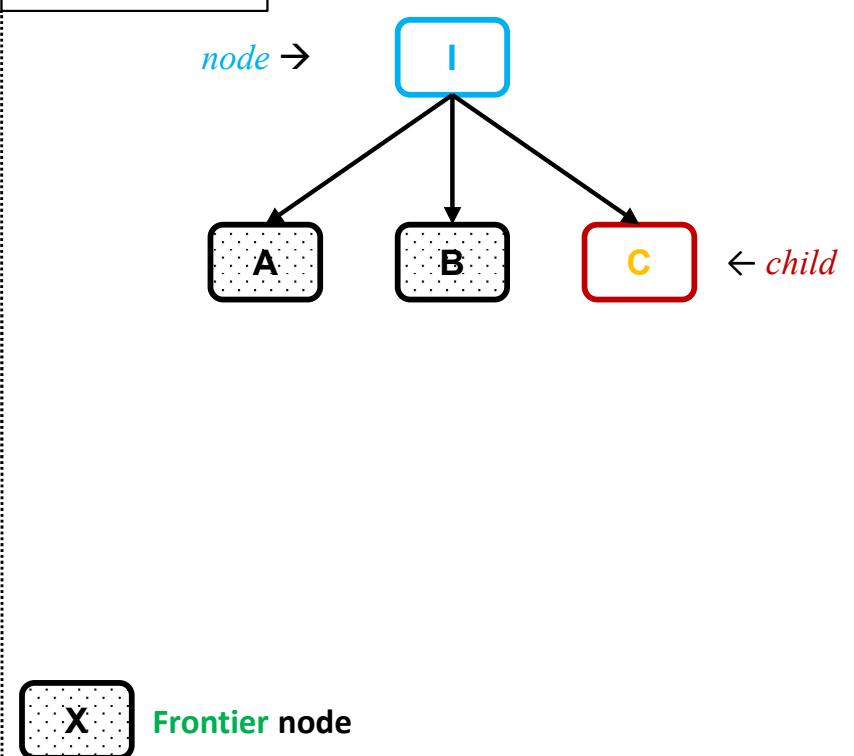
 add *child* to *frontier*

return failure

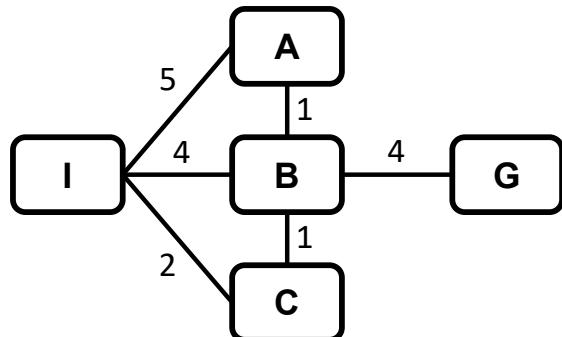
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

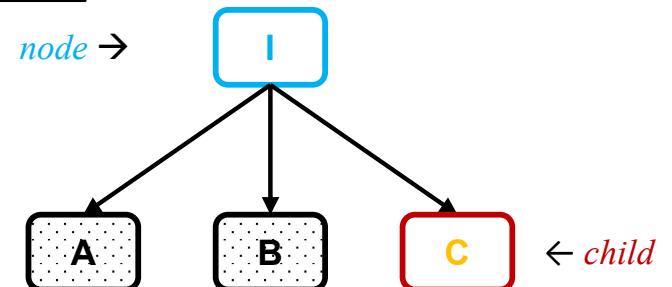
 add *child* to *frontier*

return failure

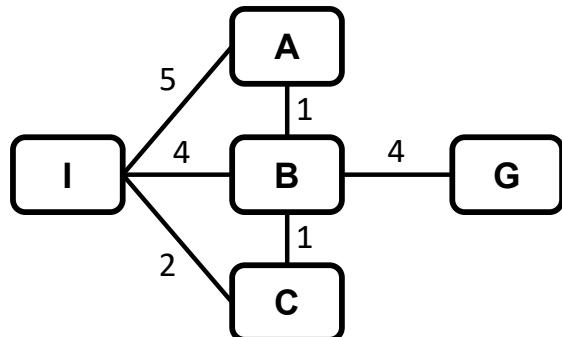
Frontier	Parent	I	I				
	Node	B	A				
	<i>f</i> (Node)	7	7				

Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

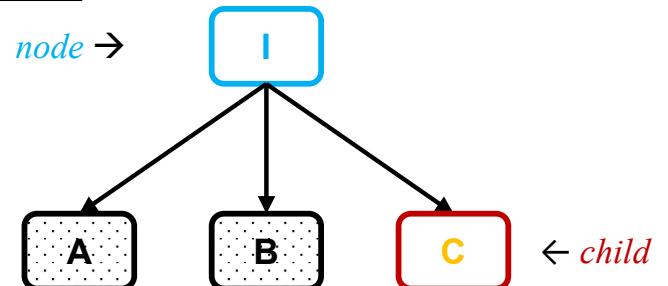
 add *child* to *frontier*

 return failure

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

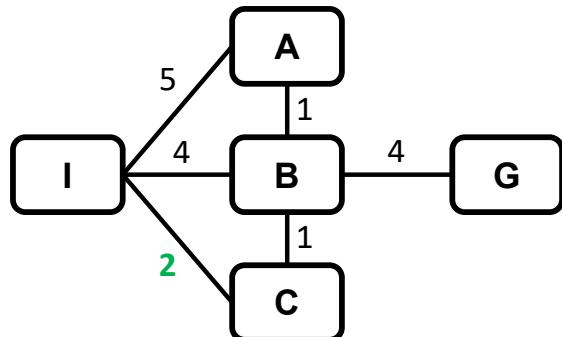
Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

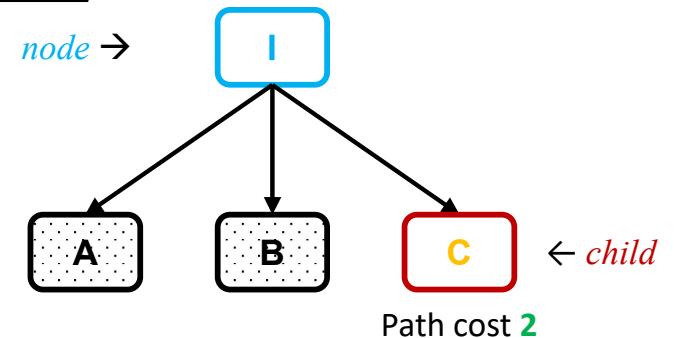
reached[s] \leftarrow *child*

 add *child* to *frontier*

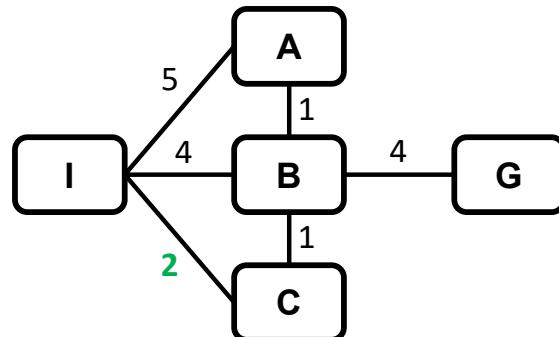
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	<i>f</i> (Node)	7	7				
Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I	I			
	Node	C	B	A			
	f(Node)	6	7	7			

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

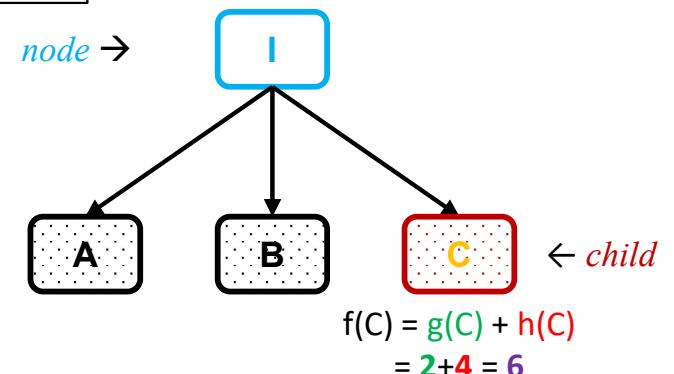
s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

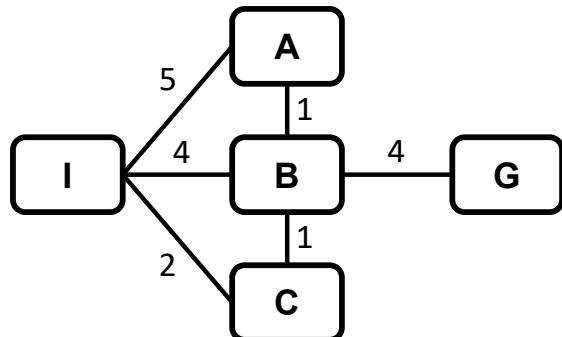
reached[s] \leftarrow *child*

 add *child* to *frontier*

 return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

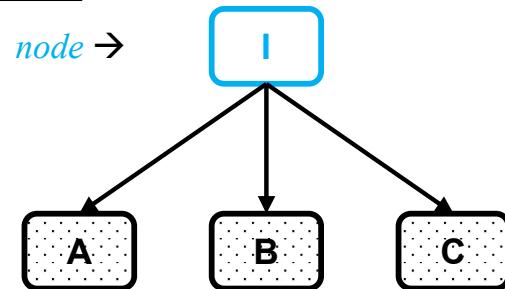
 add *child* to *frontier*

return failure

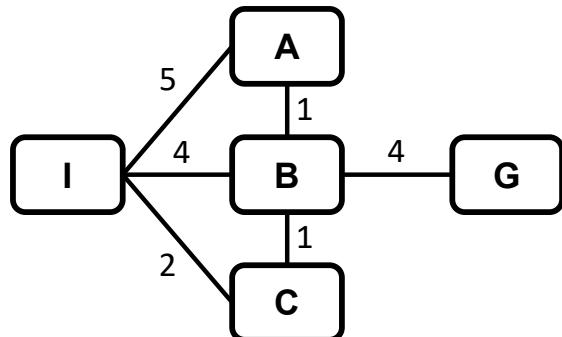
Frontier	Parent	I	I	I			
	Node	C	B	A			
	f(Node)	6	7	7			

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I	I			
	Node	C	B	A			
	f(Node)	6	7	7			

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

NOT EMPTY!

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

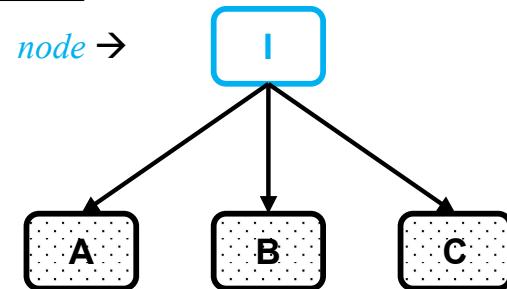
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

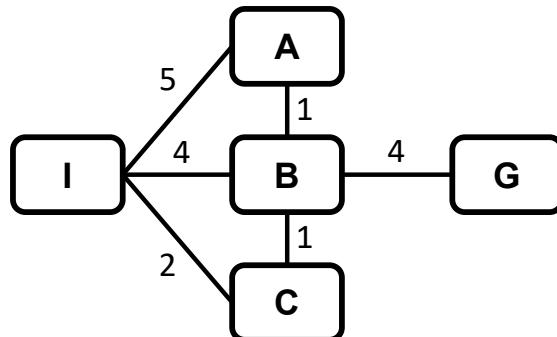
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

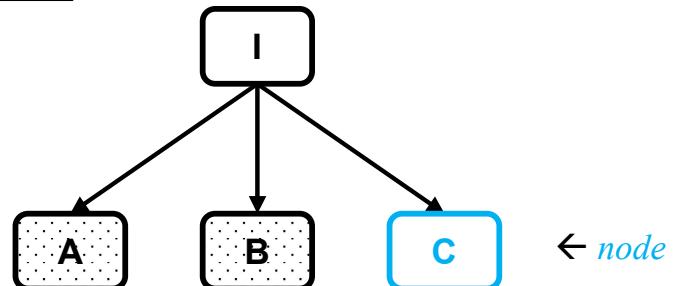
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

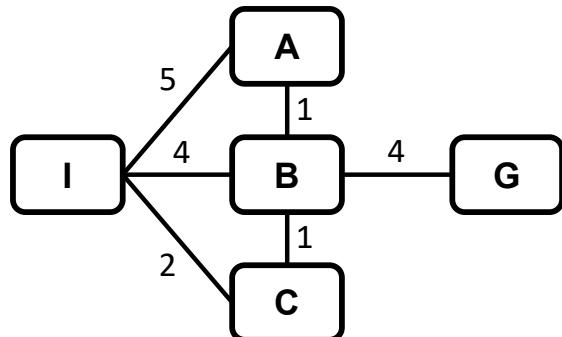
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

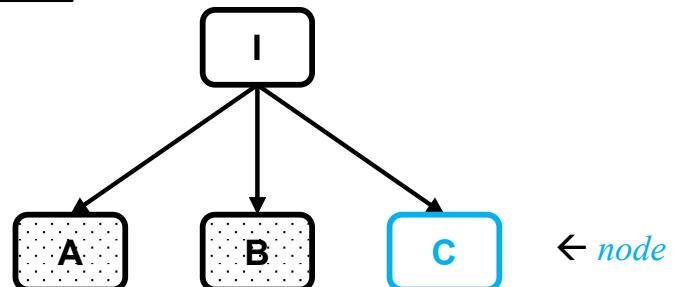
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

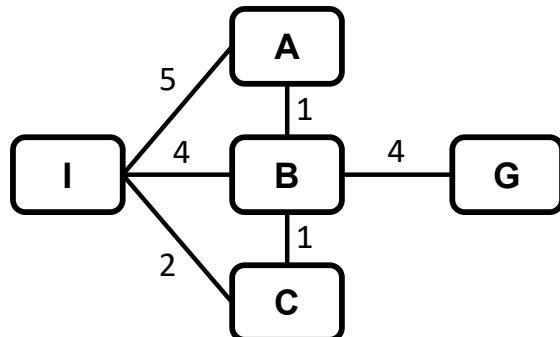
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node FALSE!*

for each *child* in EXPAND(*problem, node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

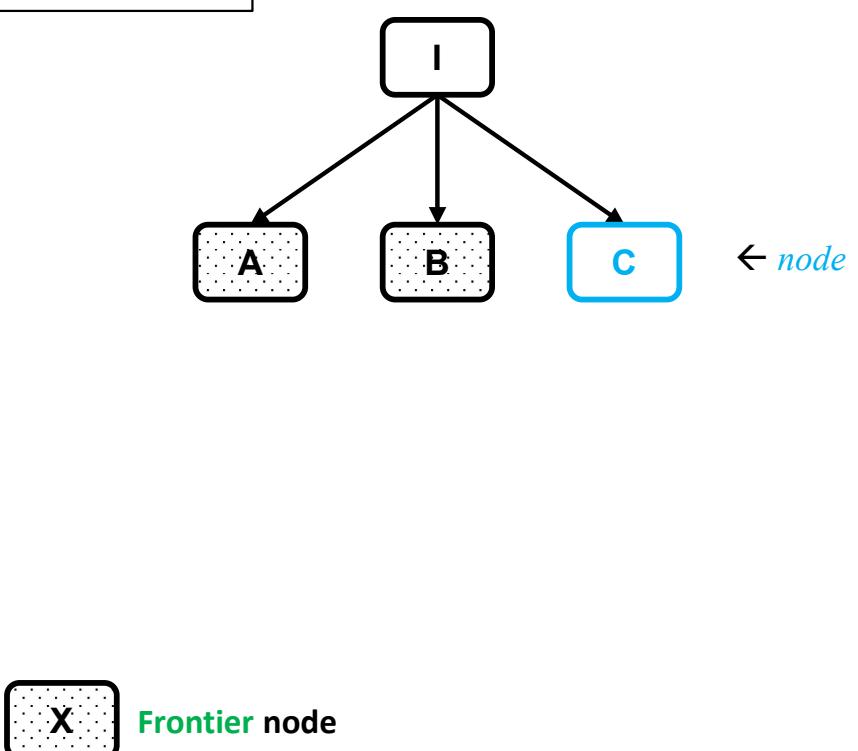
 add *child* to *frontier*

return failure

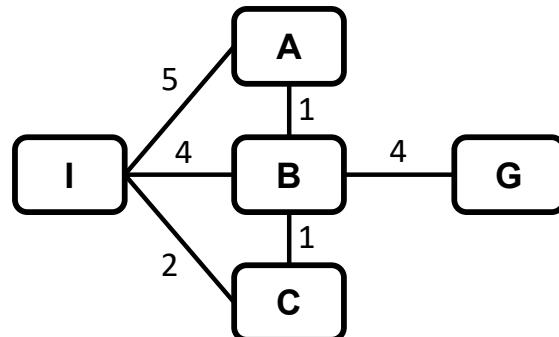
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

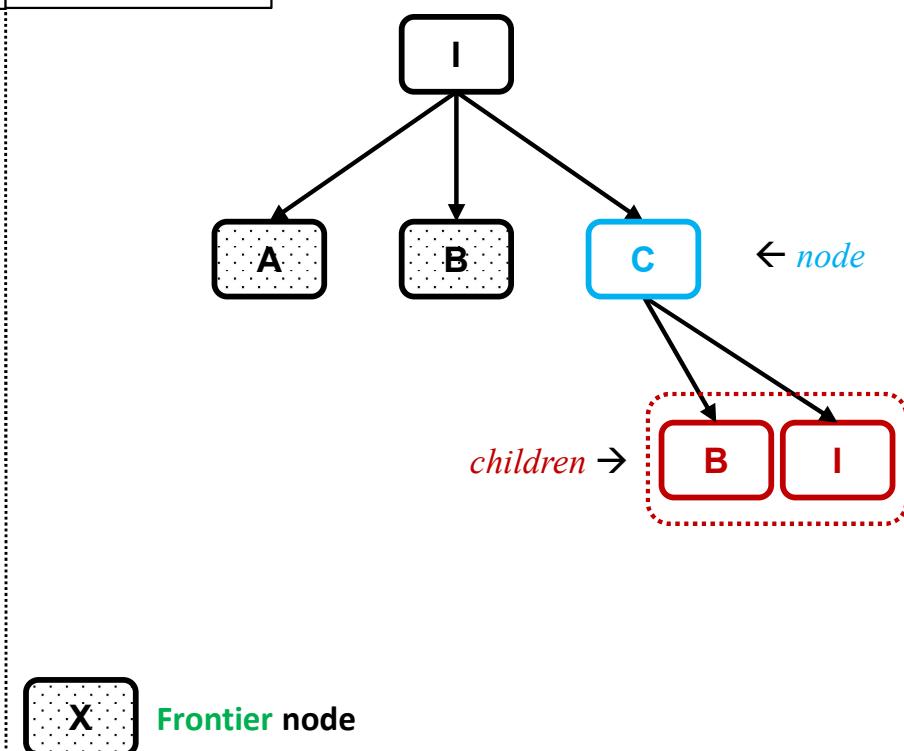
 add *child* to *frontier*

 return failure

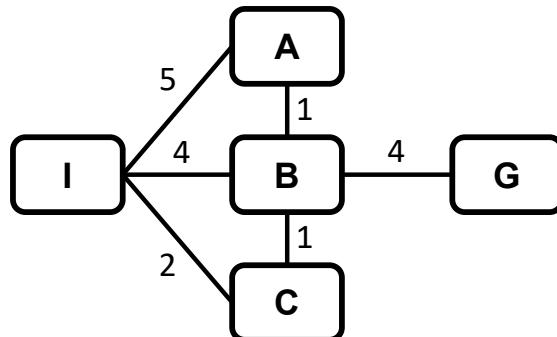
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].PATH-COST **then**

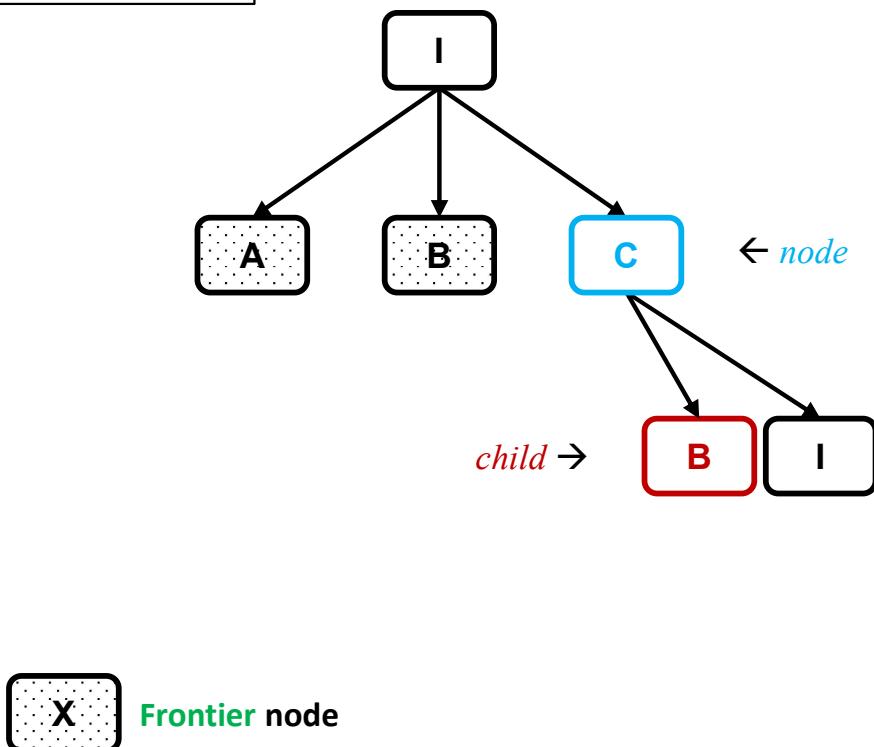
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

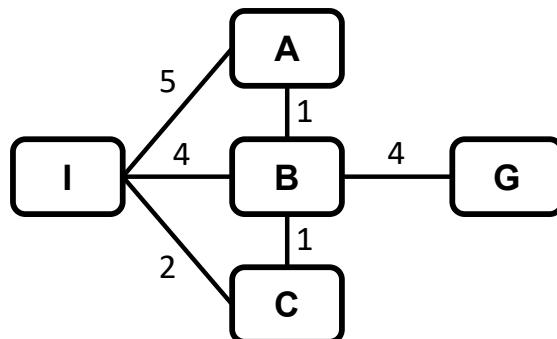
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				
Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* < *reached[s].PATH-COST* then

reached[s] \leftarrow *child*

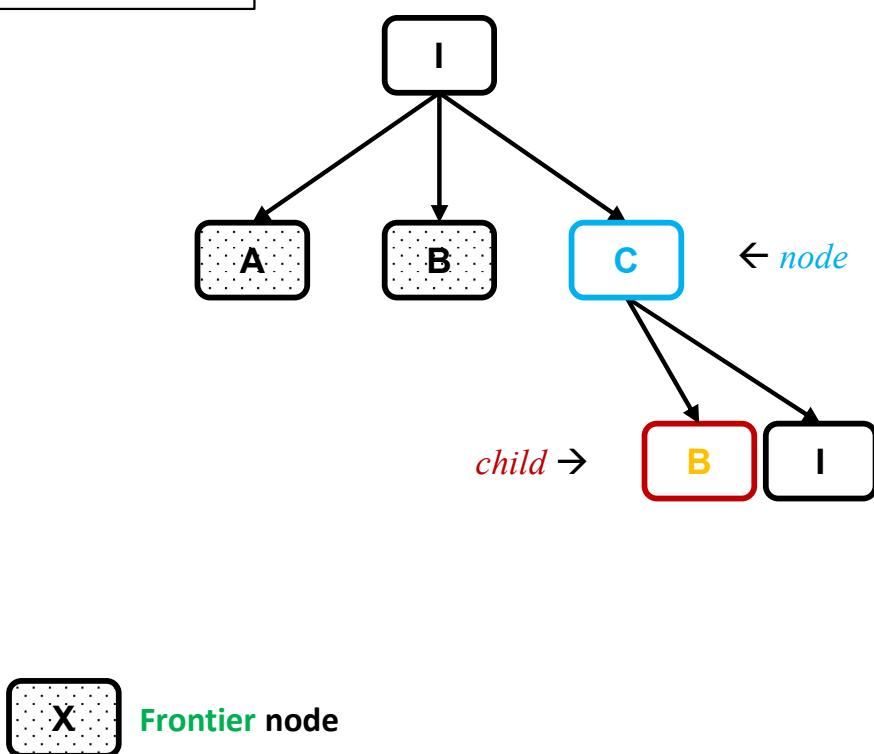
 add *child* to *frontier*

 return failure

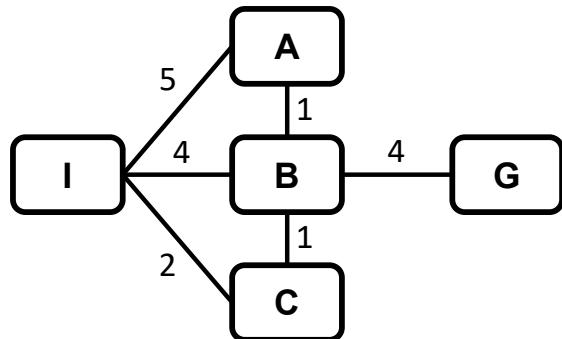
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

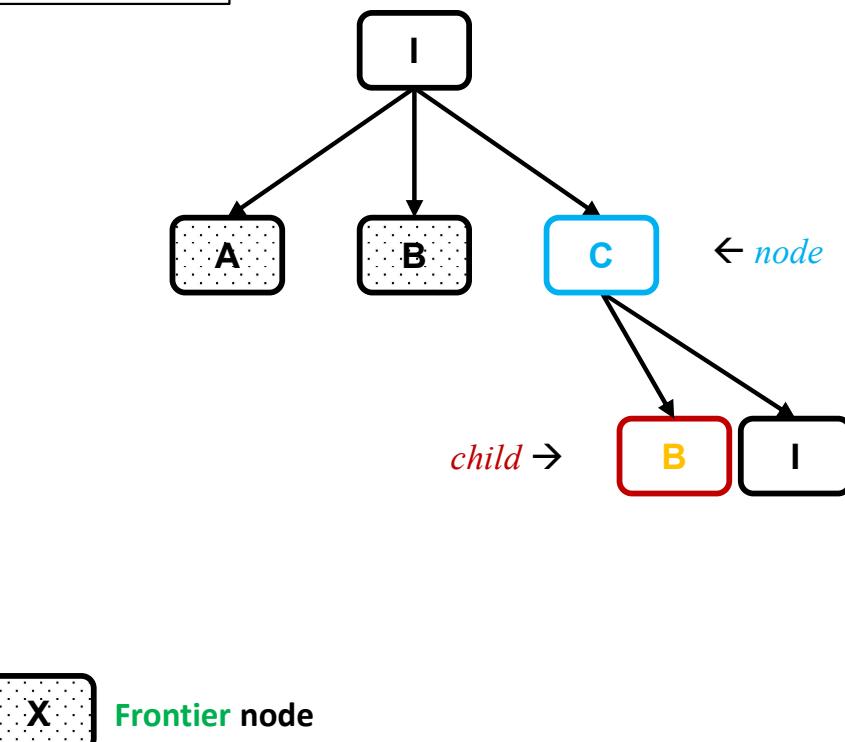
reached[s] \leftarrow *child*

 add *child* to *frontier*

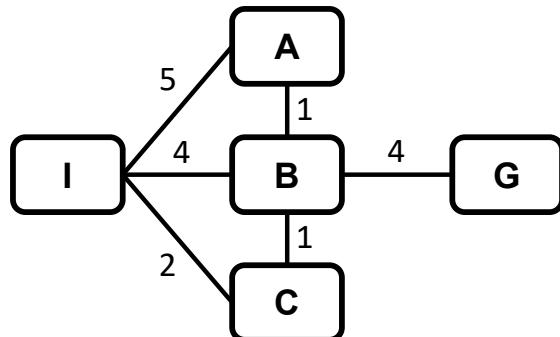
 return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	<i>f</i> (Node)	7	7				
Reached	Parent	---	I	I	I		
	Key/State	I	A	B	C		
	Path cost	0	5	4	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	A					
f(Node)	7	7					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

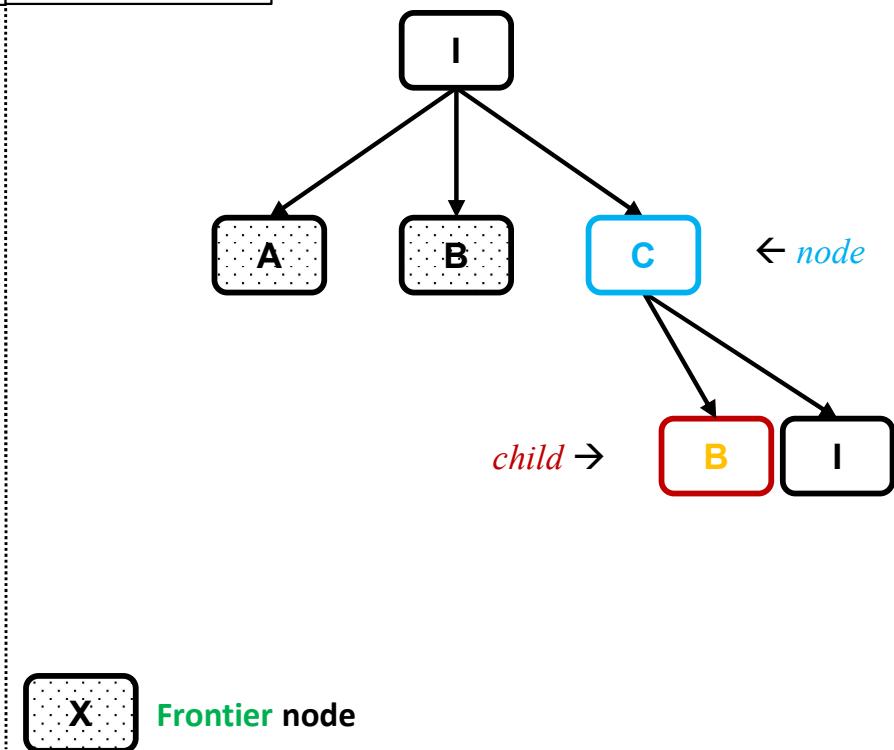
s \leftarrow *child.STATE*

 if *s* is not ~~reached~~ or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

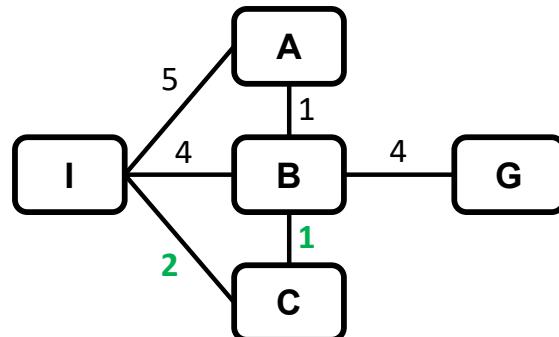
reached[s] \leftarrow *child*

 add *child* to *frontier*

 return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* < *reached*[*s*].*PATH-COST* then

reached[*s*] \leftarrow *child*

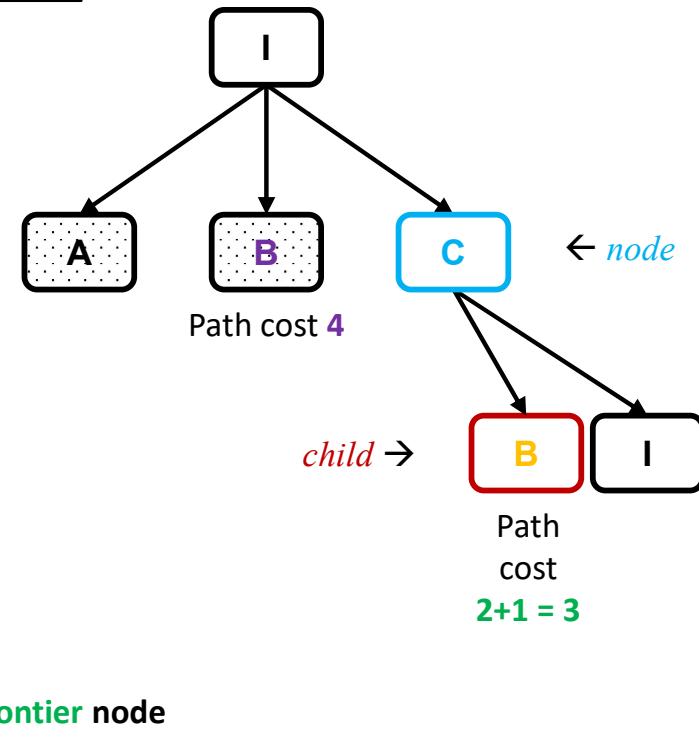
 add *child* to *frontier*

 return failure

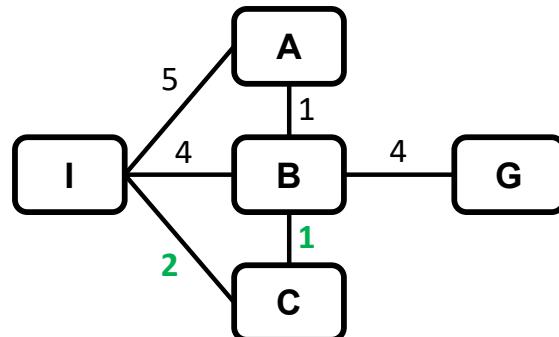
Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* < *reached*[*s*].*PATH-COST* then

reached[*s*] \leftarrow *child*

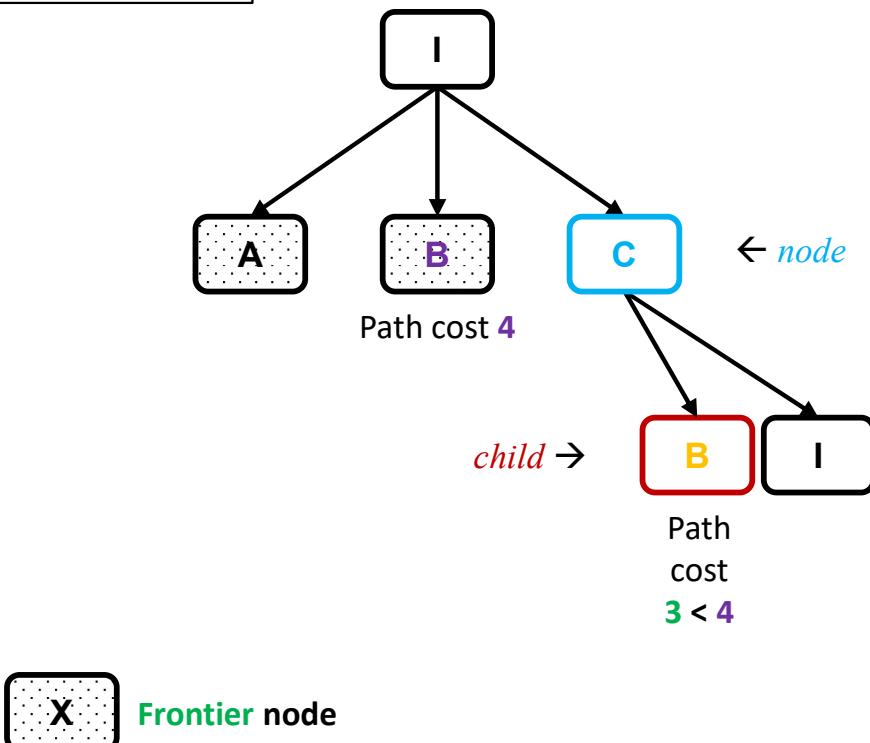
 add *child* to *frontier*

 return failure

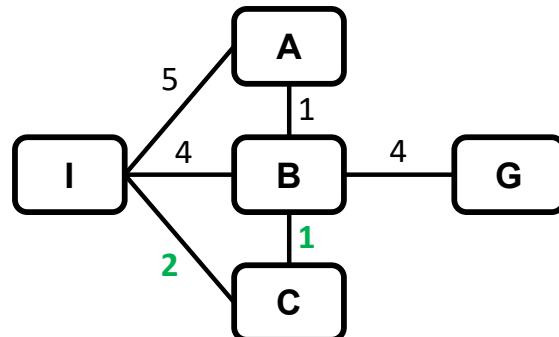
Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

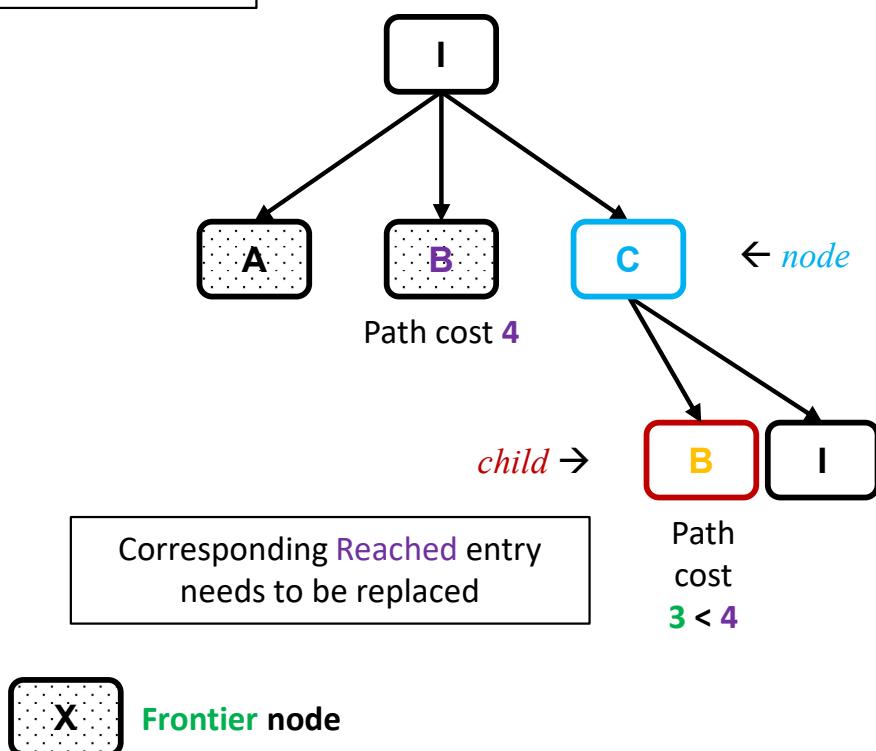
 add *child* to *frontier*

return failure

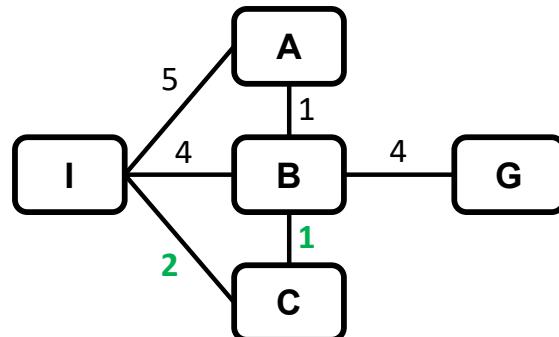
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I			
	Node	B	A			
	$f(\text{Node})$	7	7			

Reached	Parent	---	I	I	I	
	Key/State	I	A	B	C	
	Path cost	0	5	4	2	



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

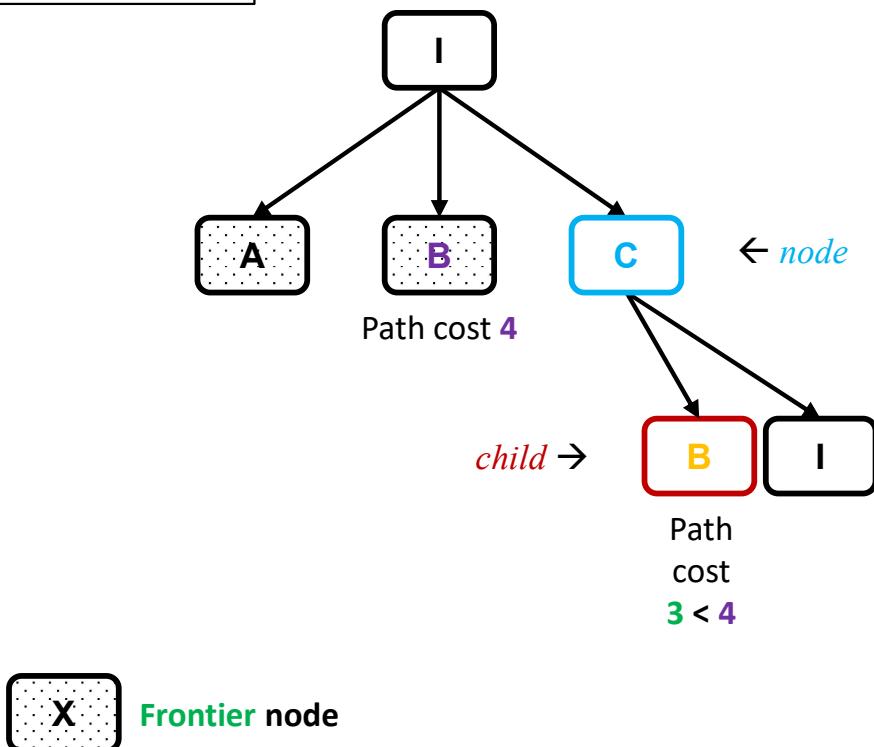
reached[s] \leftarrow *child*

 add *child* to *frontier*

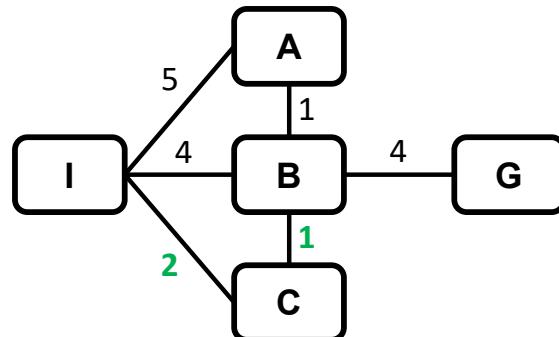
 return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	<i>f</i> (Node)	7	7				
Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

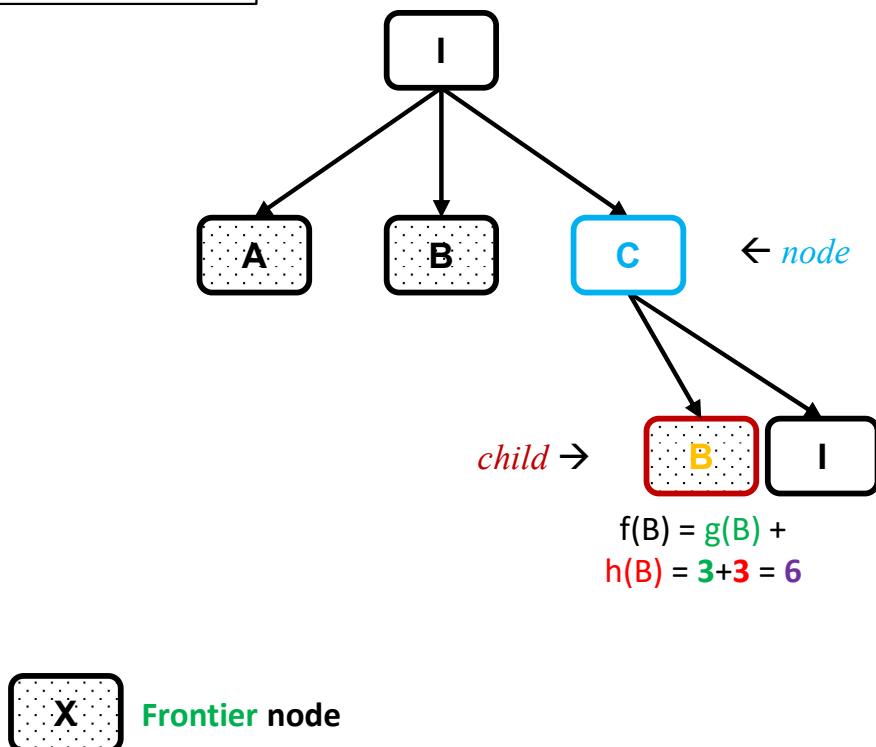
reached[s] \leftarrow *child*

 add *child* to *frontier*

 return failure

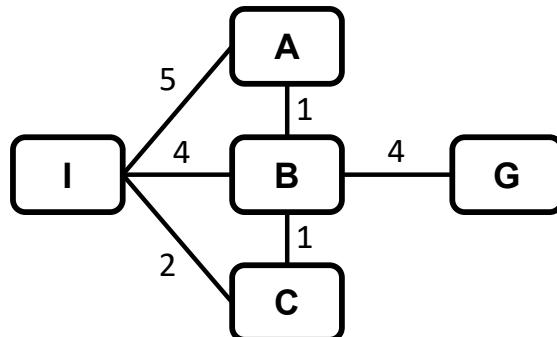
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	C	I	I			
	Node	B	B	A			
	<i>f</i> (Node)	6	7	7			
Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



$$f(B) = g(B) + h(B) = 3+3 = 6$$

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].PATH-COST **then**

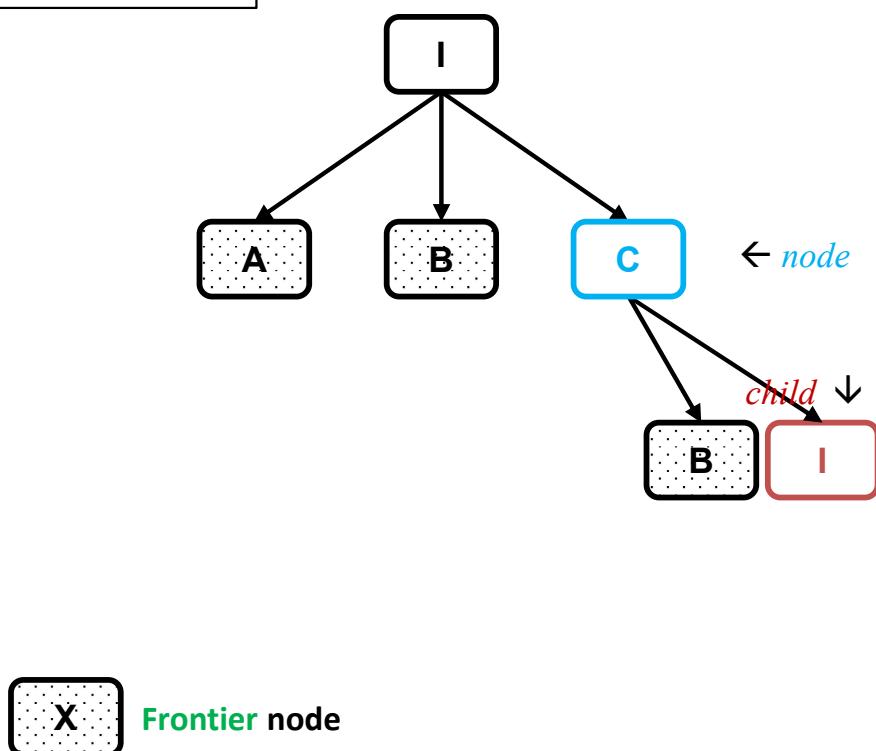
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

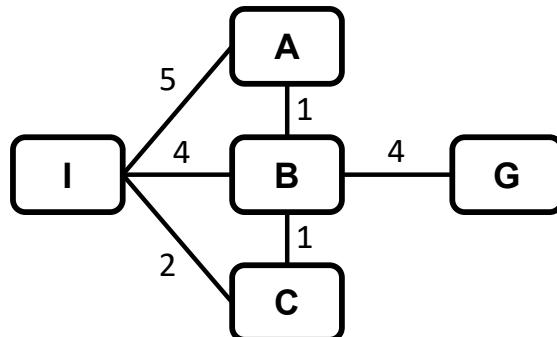
return failure

State Space Graph		Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	C	I	I			
	Node	B	B	A			
	f(Node)	6	7	7			
Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

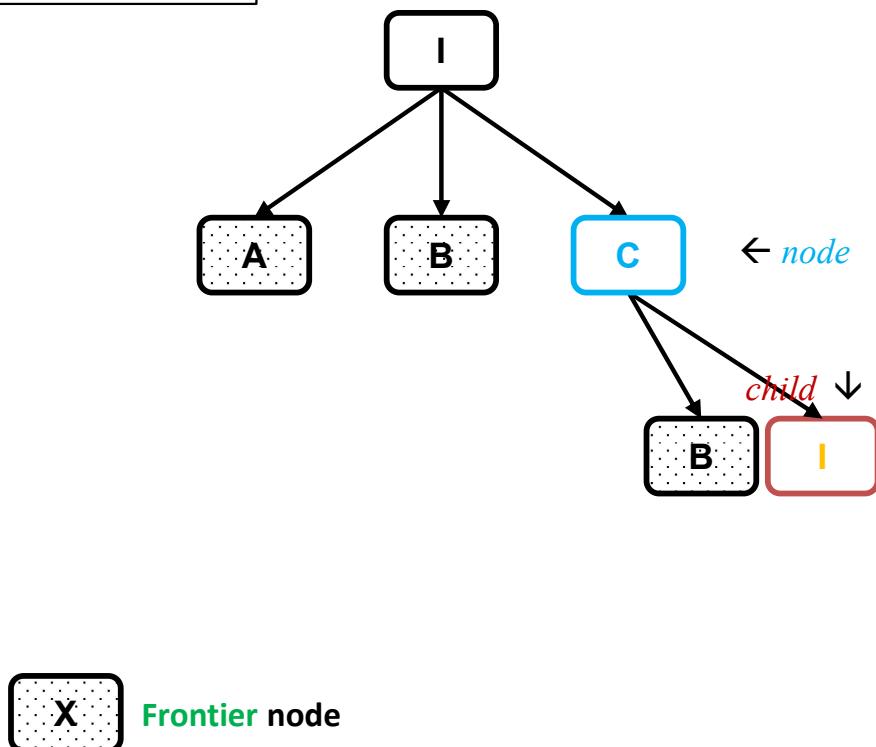
reached[s] \leftarrow *child*

 add *child* to *frontier*

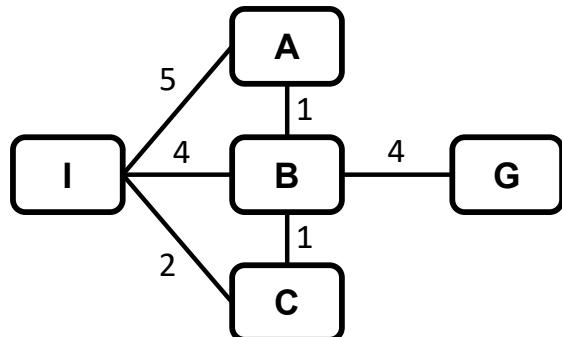
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	C	I	I			
	Node	B	B	A			
	f(Node)	6	7	7			
Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

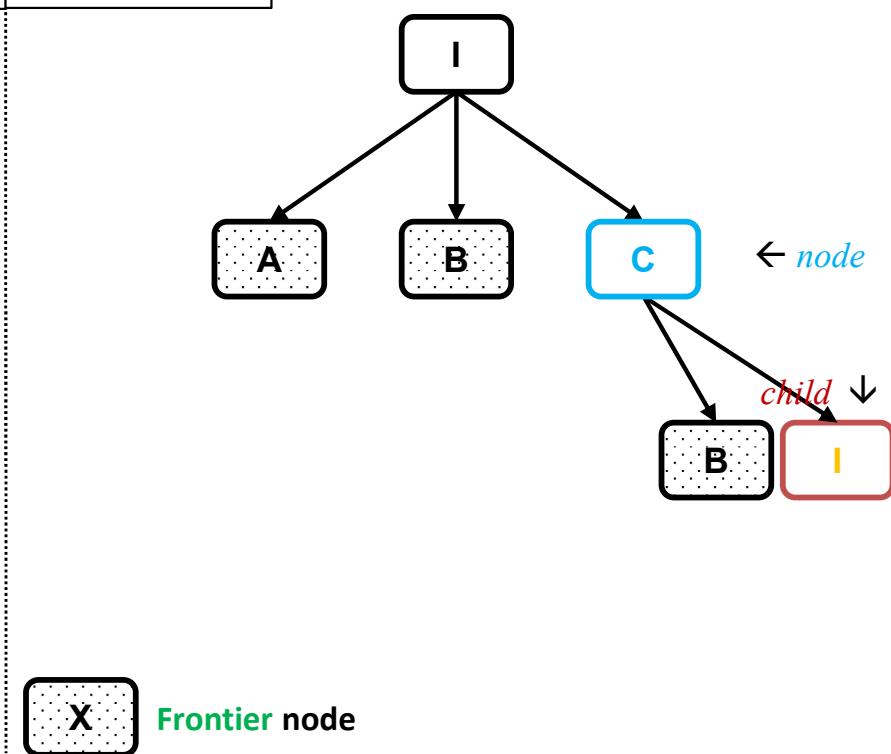
 add *child* to *frontier*

return failure

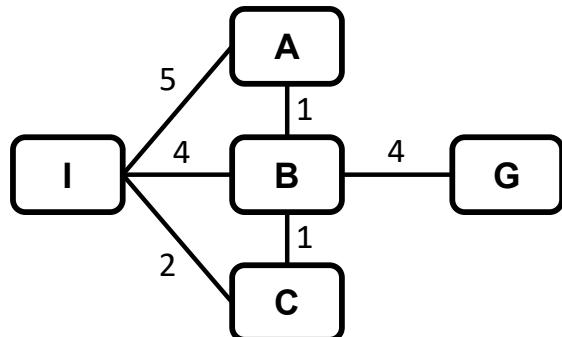
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	C	I	I			
	Node	B	B	A			
	f(Node)	6	7	7			

Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not ~~X~~ *reached* or *child*.PATH-COST < *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

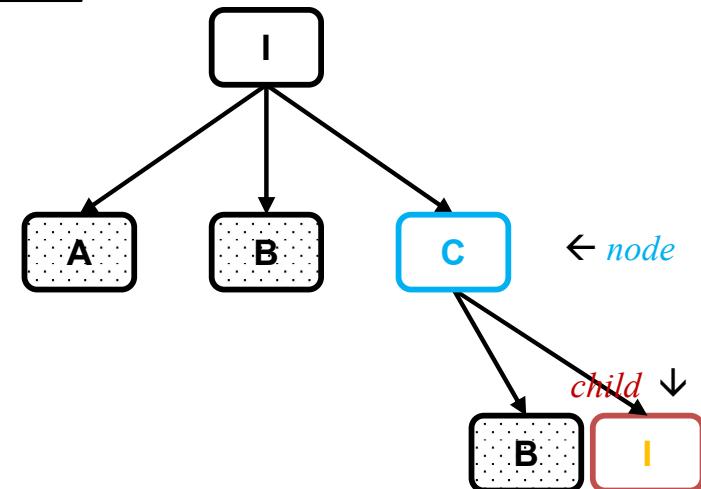
 add *child* to *frontier*

 return failure

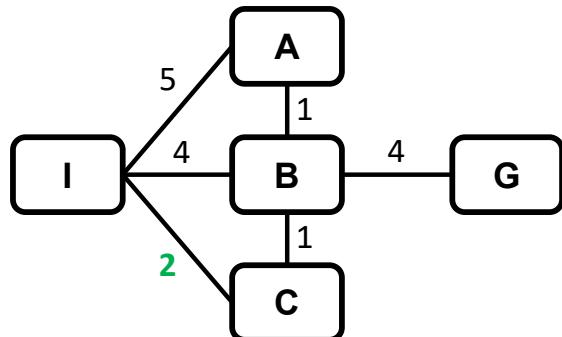
Frontier	Parent	C	I	I			
Node	B	B	A				
f(Node)	6	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

$s \leftarrow$ *child*.STATE

 if s is not ~~X~~ *reached* or *child*.PATH-COST $<$ *reached*[s].PATH-COST then

reached[s] \leftarrow *child*

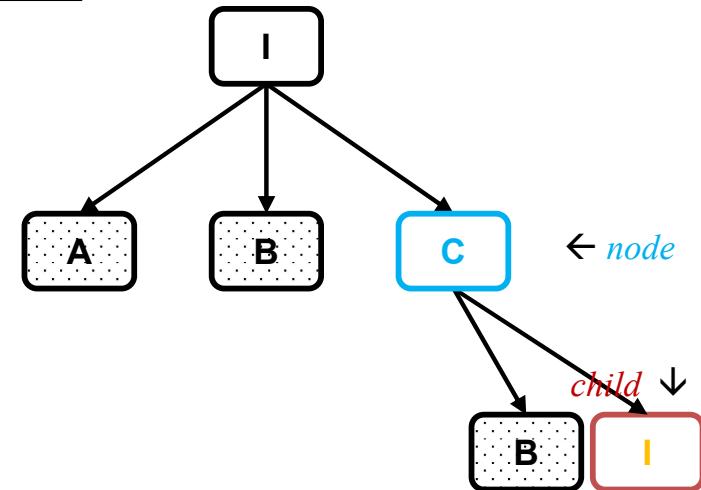
 add *child* to *frontier*

 return failure

Frontier	Parent	C	I	I			
Node	B	B	A				
<i>f</i> (Node)	6	7	7				

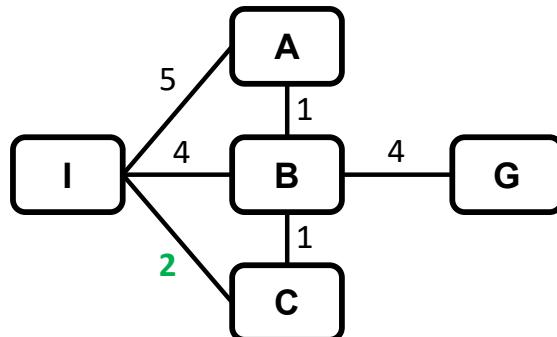
Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~< i>reached[*s*].PATH-COST~~ then

reached[*s*] \leftarrow *child*

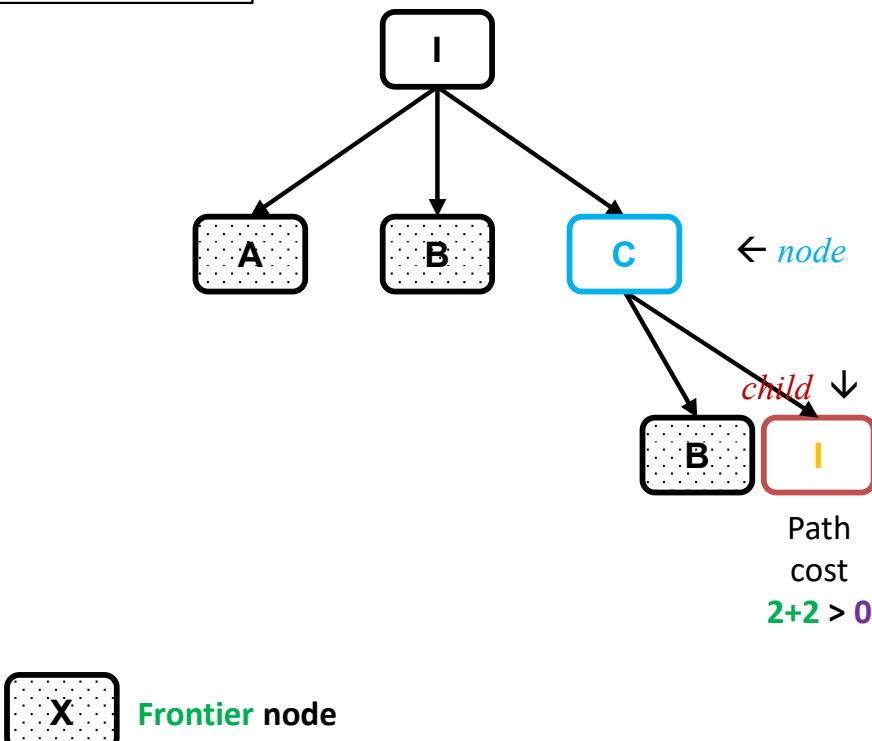
 add *child* to *frontier*

 return failure

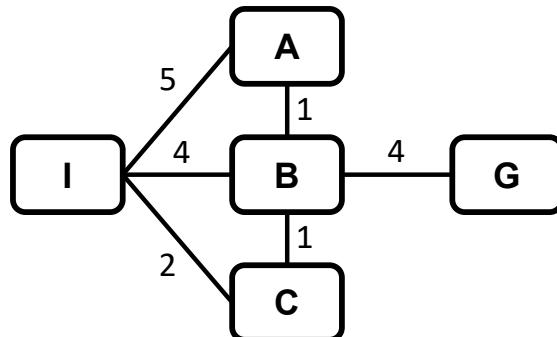
Frontier	Parent	C	I	I			
Node	B	B	A				
<i>f</i> (Node)	6	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	C	I	I			
Node	B	B	A				
f(Node)	6	7	7				

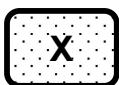
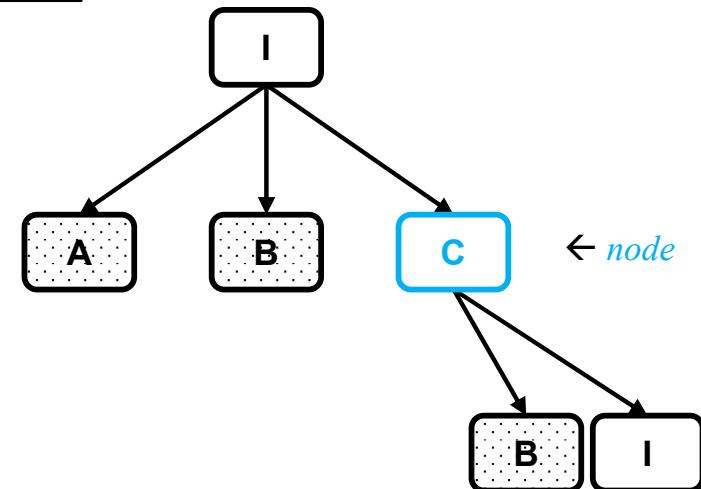
Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph

Frontier / Reached

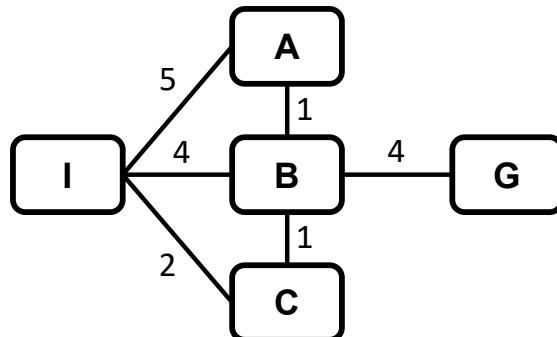
Algorithm

Search Tree



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	C	I	I			
Node	B	B	A				
$f(\text{Node})$	6	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, f)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by f , with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

NOT EMPTY!

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

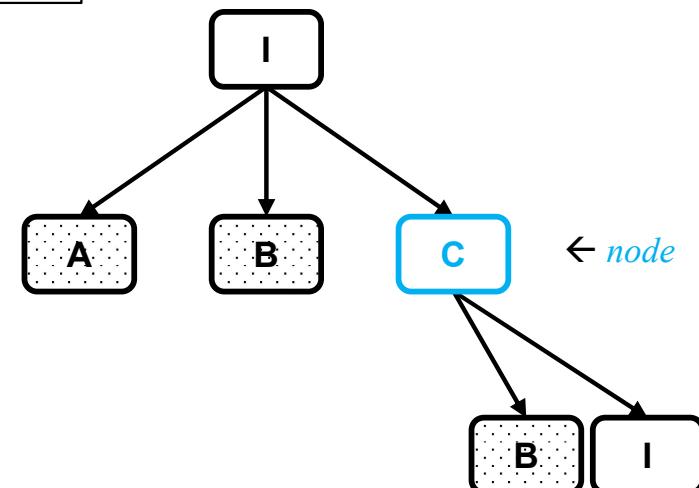
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

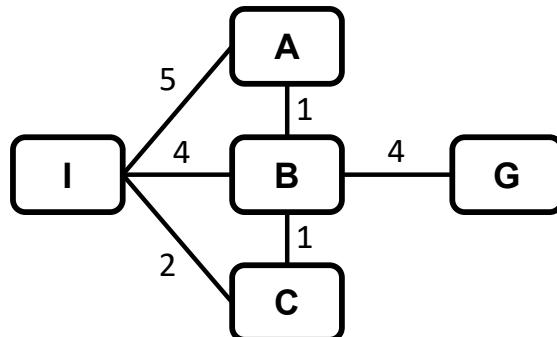
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

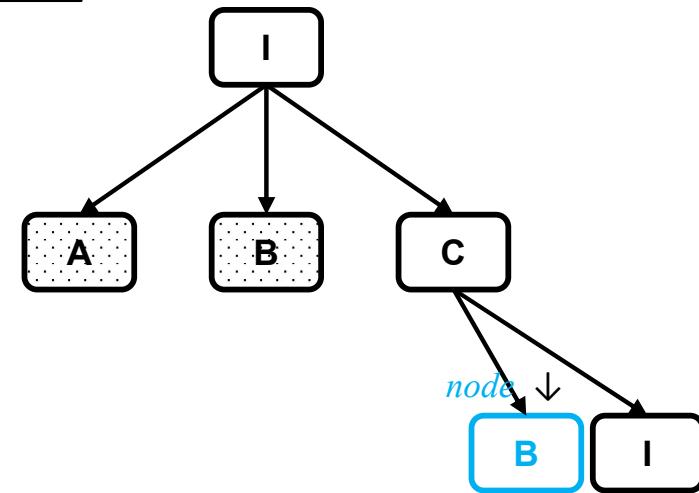
 add *child* to *frontier*

return failure

Frontier	Parent	I	I				
Node	B	A					
f(Node)	7	7					

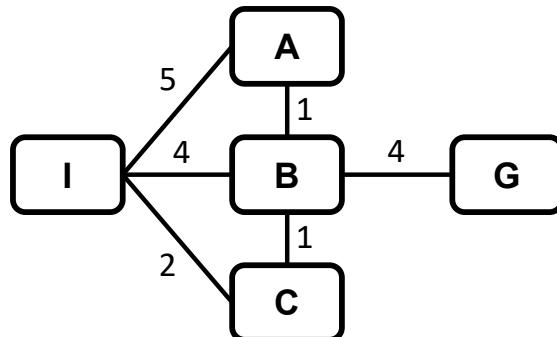
Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

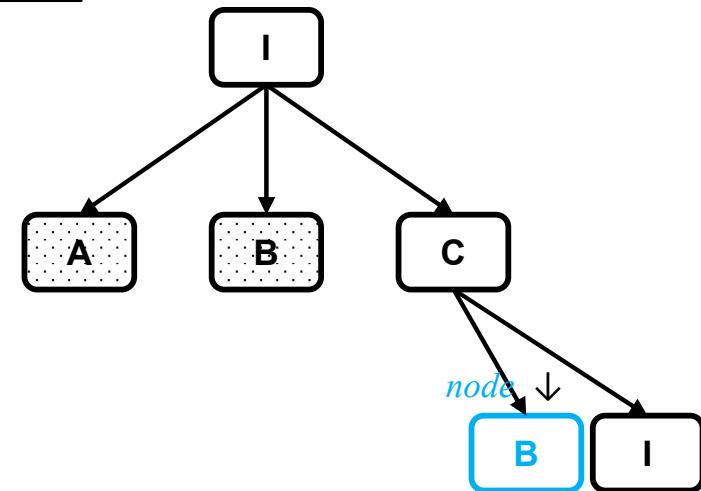
reached[s] \leftarrow *child*

 add *child* to *frontier*

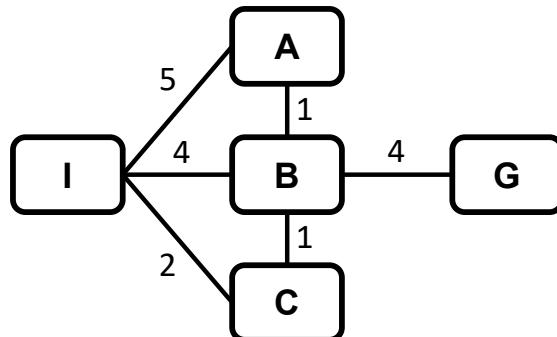
return failure

State Space Graph		Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				
Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node FALSE!*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

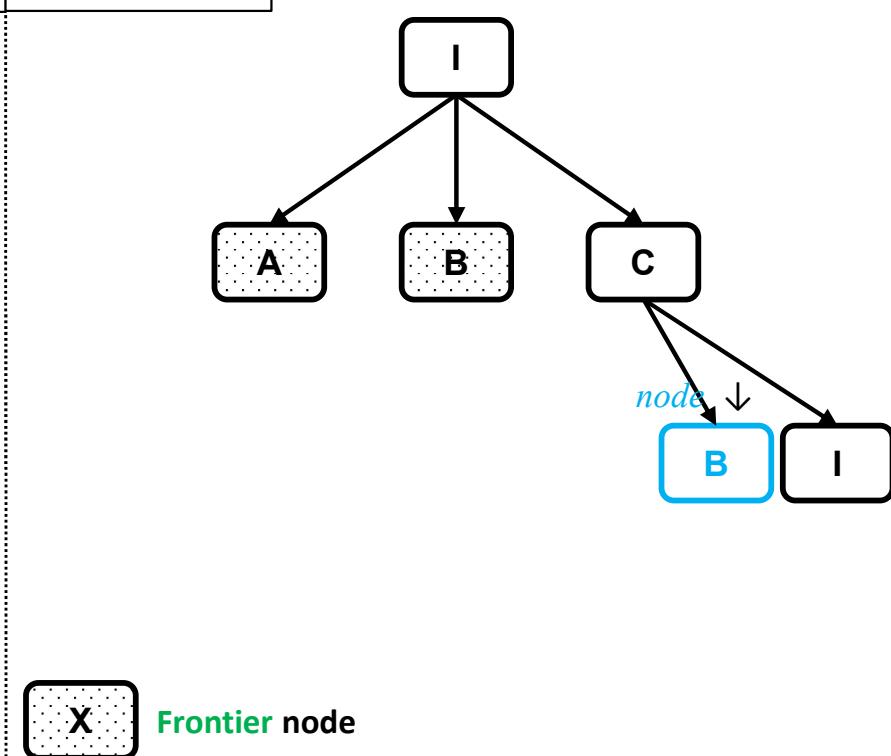
 add *child* to *frontier*

return failure

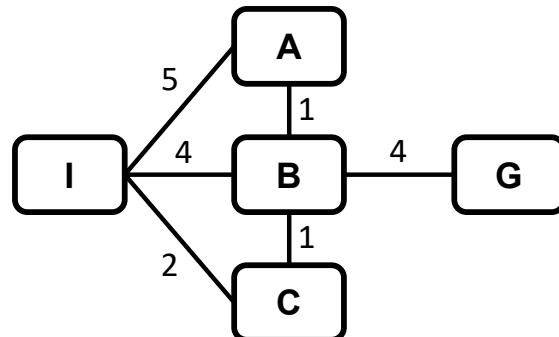
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL*(*node.STATE*) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

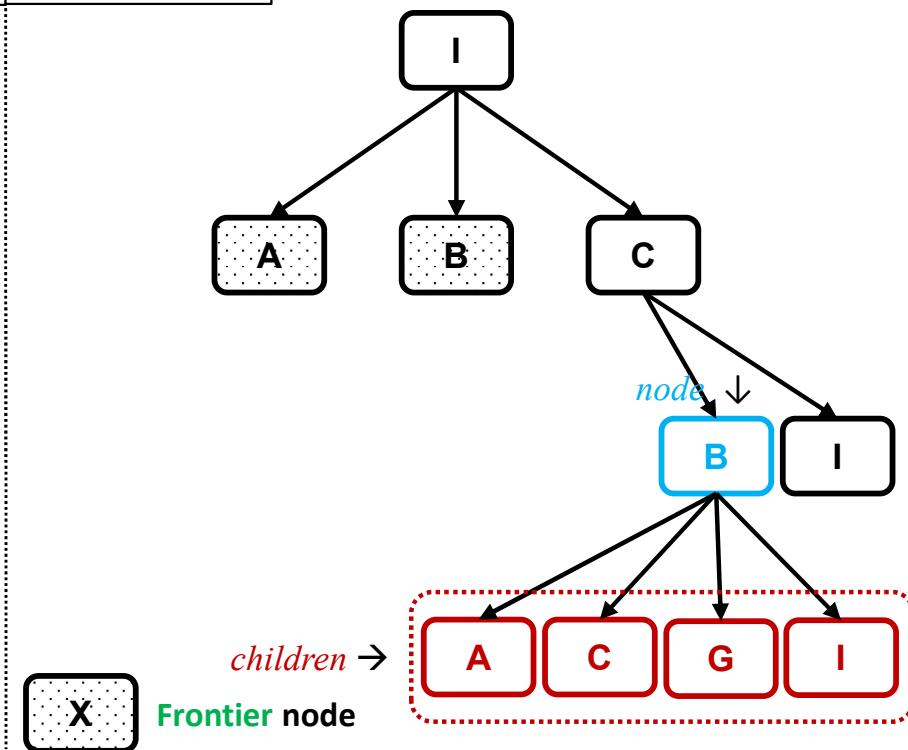
 add *child* to *frontier*

 return failure

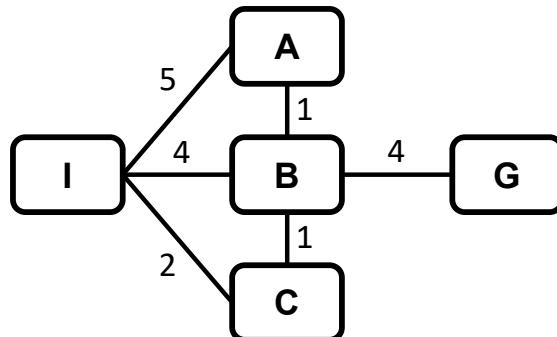
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL*(*node.STATE*) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

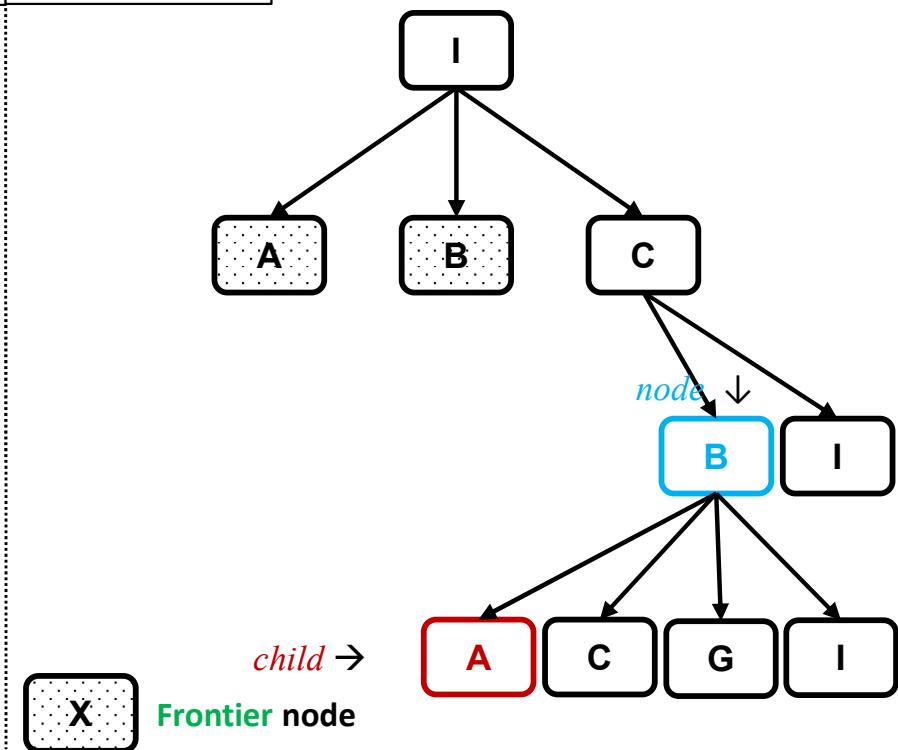
 add *child* to *frontier*

 return failure

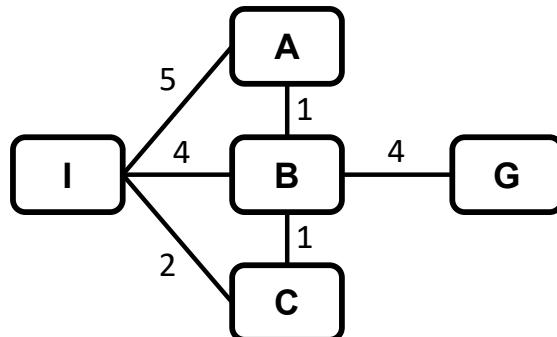
Frontier	Parent	I	I			
Node	B	A				
$f(\text{Node})$	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I			
Node	B	A				
$f(\text{Node})$	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

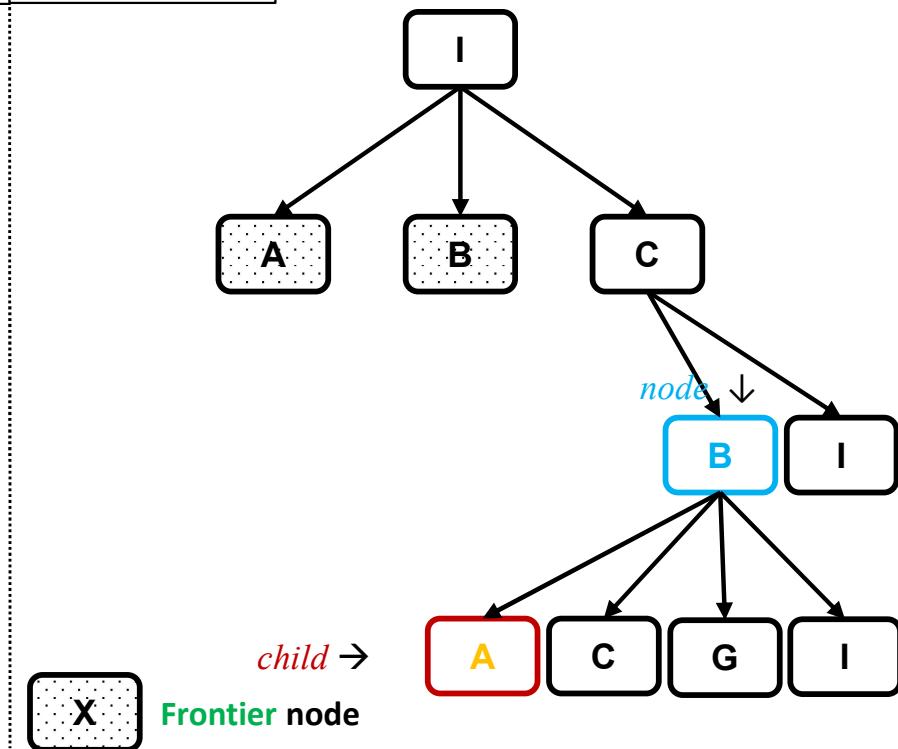
s \leftarrow *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

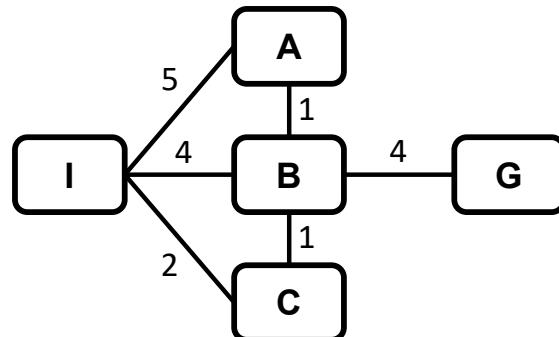
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

reached[s] \leftarrow *child*

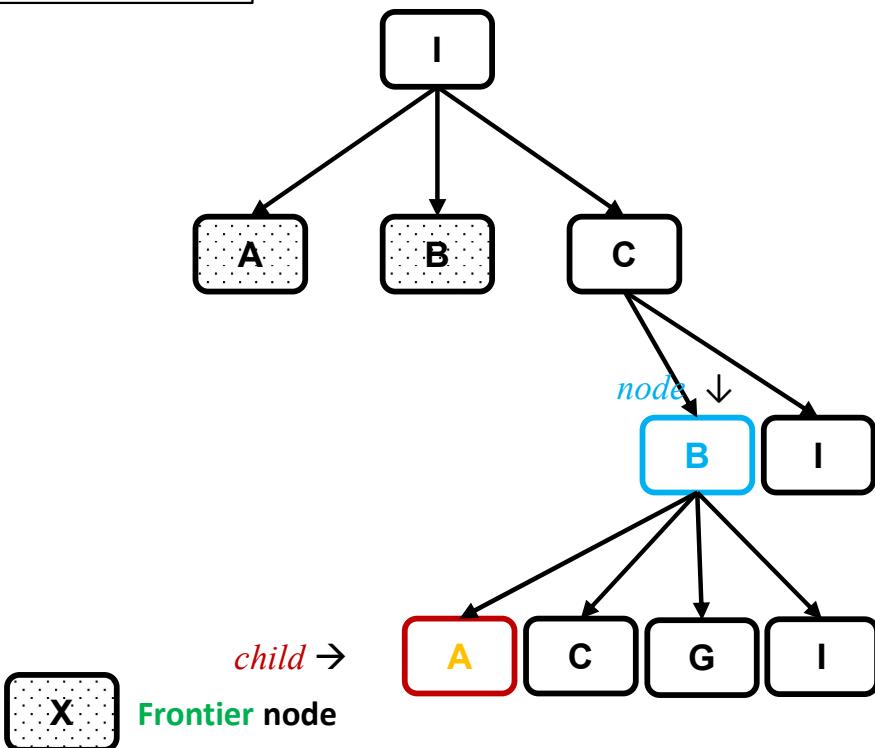
 add *child* to *frontier*

 return failure

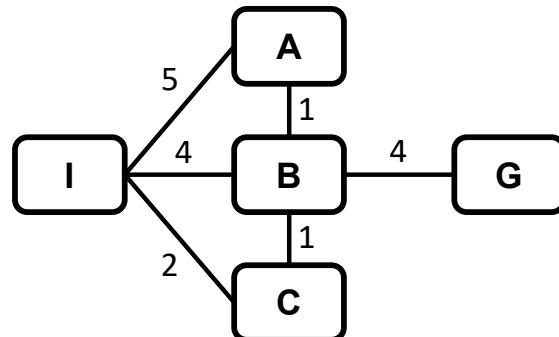
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not ~~X~~ *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

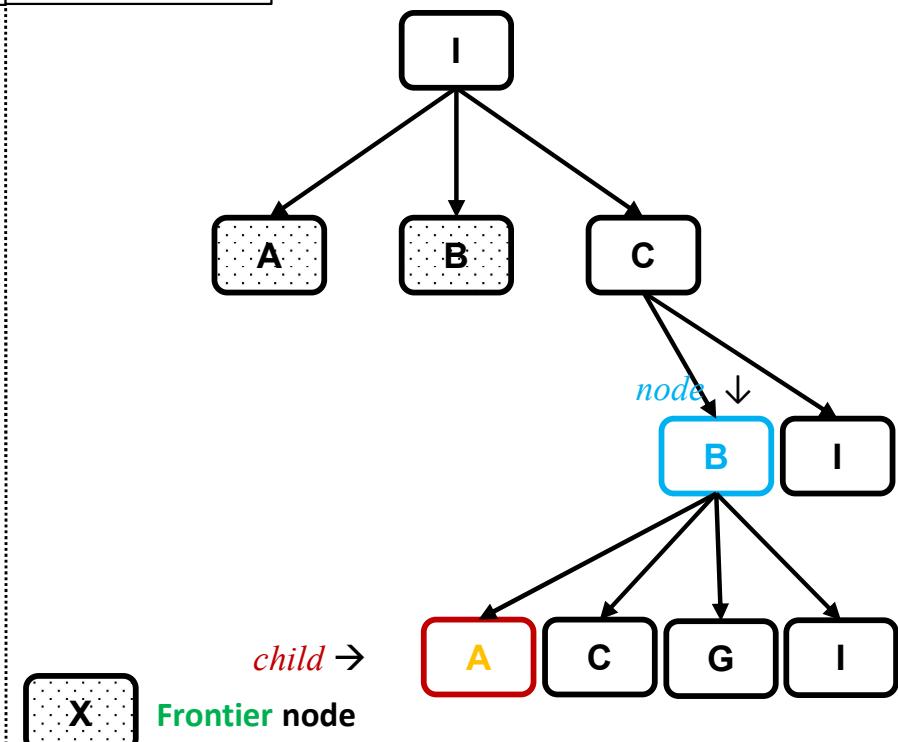
 add *child* to *frontier*

return failure

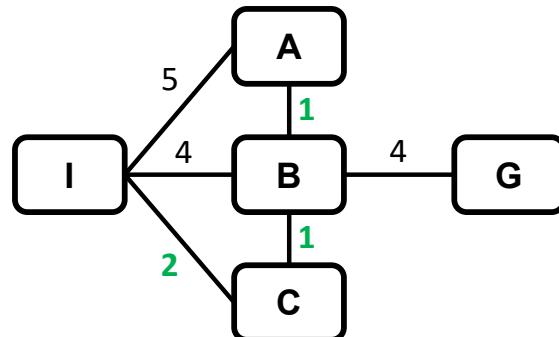
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<=~~ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

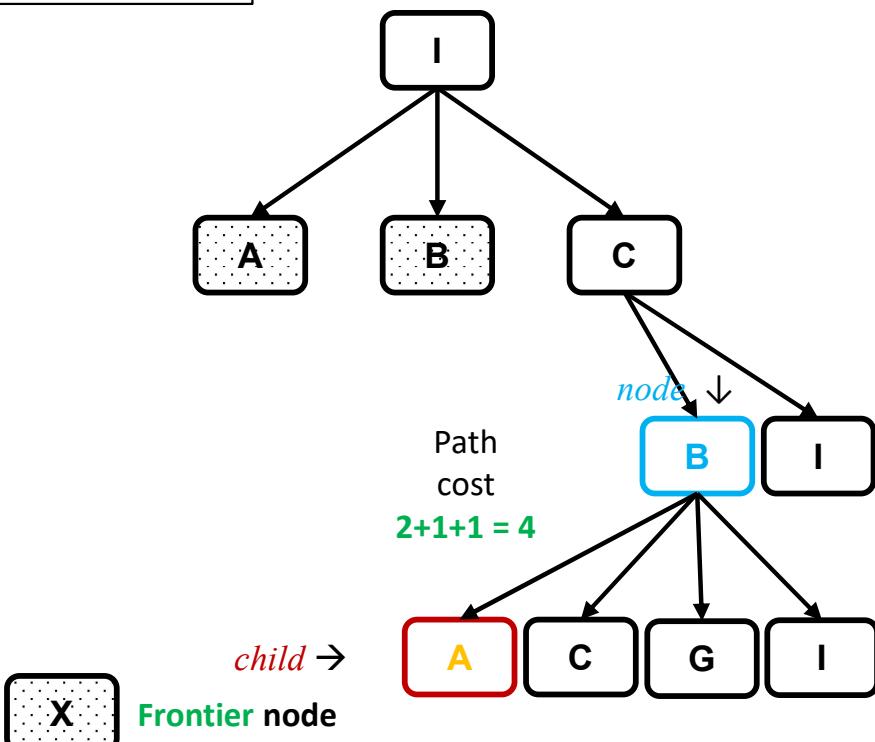
 add *child* to *frontier*

 return failure

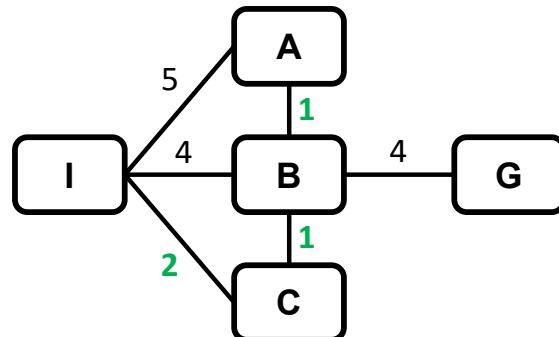
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* ~~<=~~ *reached[s].PATH-COST* then

reached[s] \leftarrow *child*

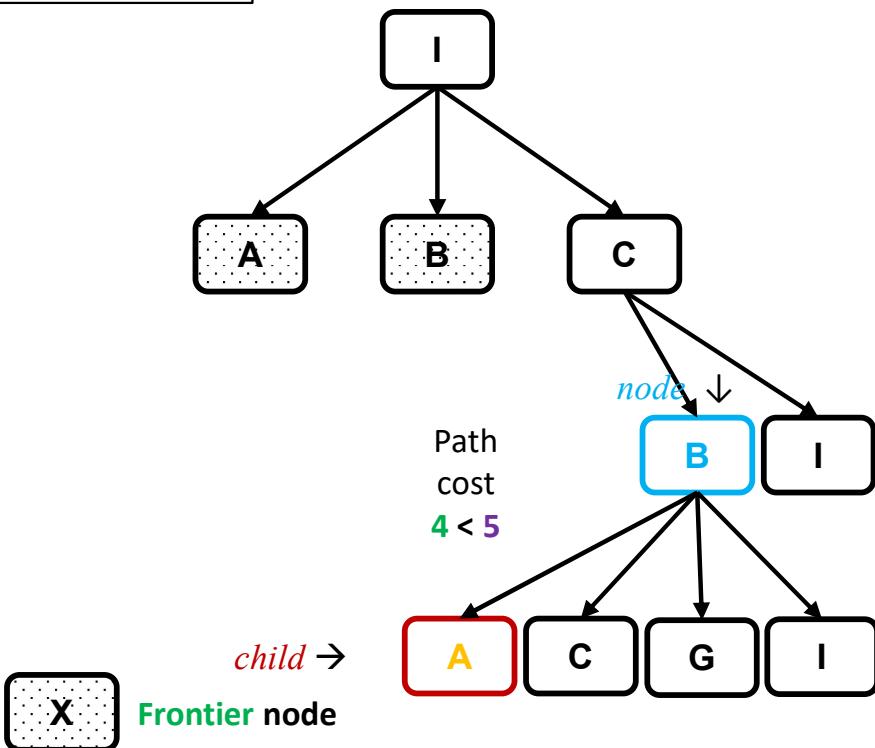
 add *child* to *frontier*

 return failure

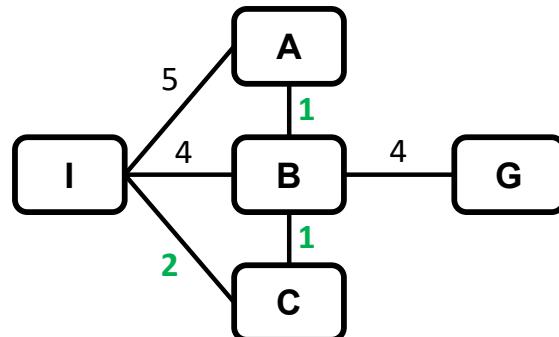
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* \leq *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

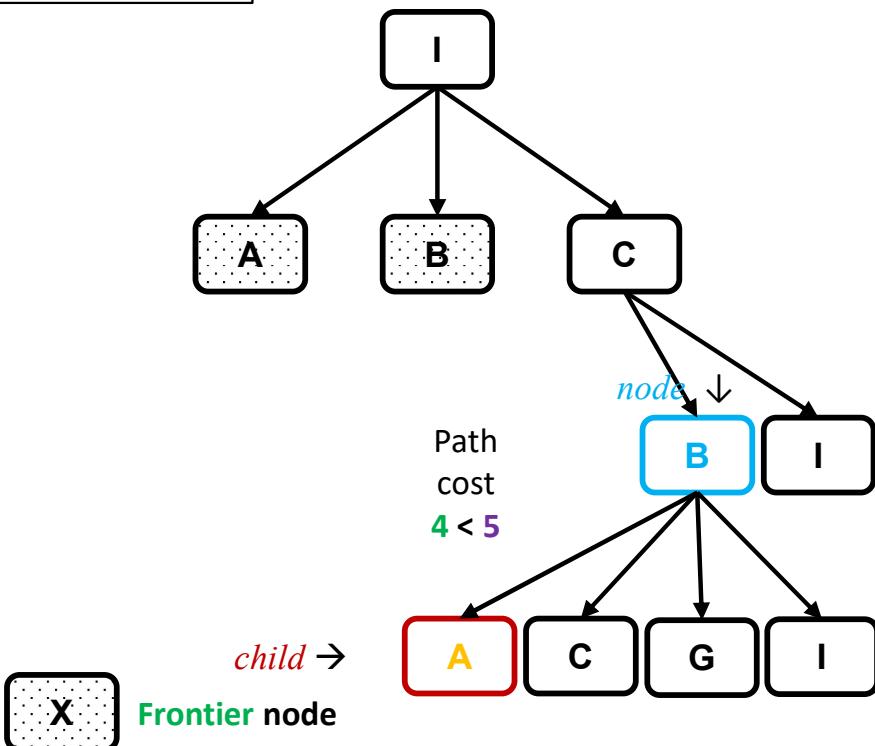
 add *child* to *frontier*

 return failure

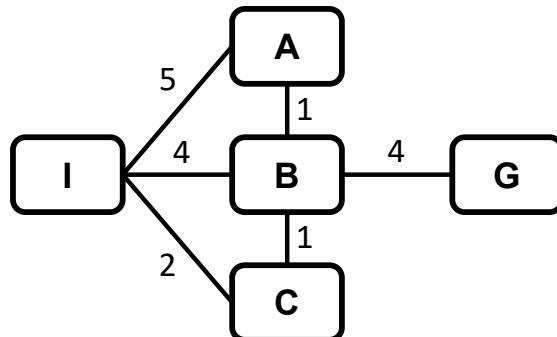
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

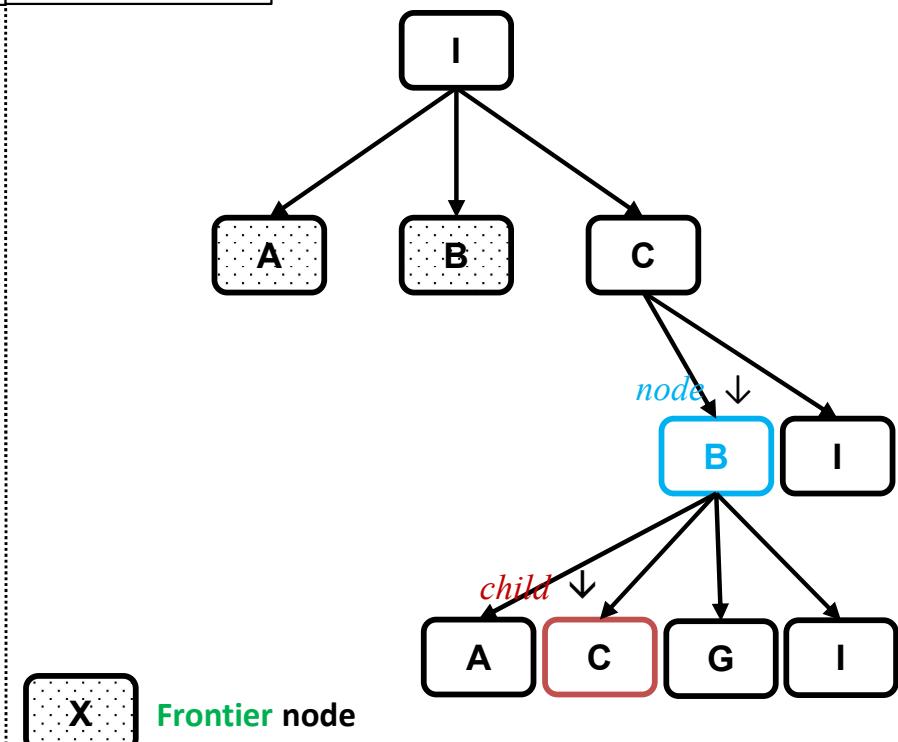
 add *child* to *frontier*

return failure

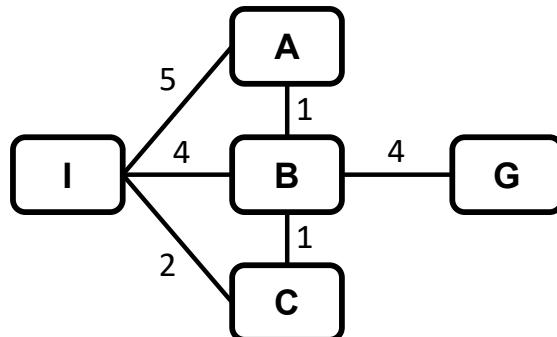
Frontier	Parent	I	I			
	Node	B	A			
	<i>f</i> (Node)	7	7			

Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

reached[s] \leftarrow *child*

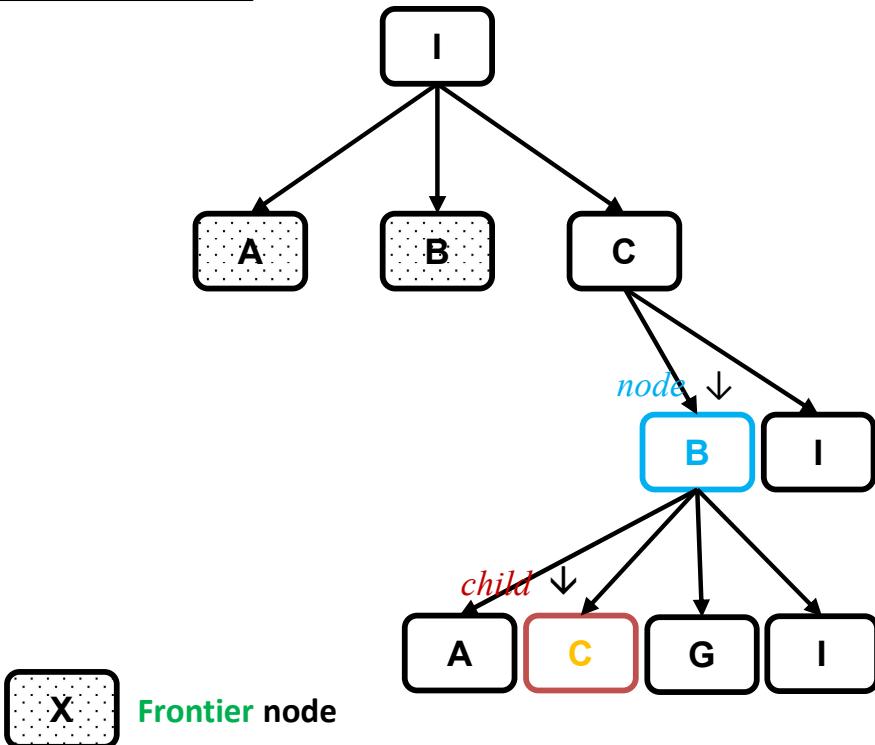
 add *child* to *frontier*

 return failure

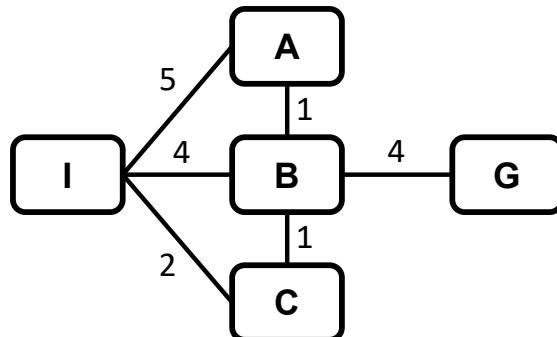
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

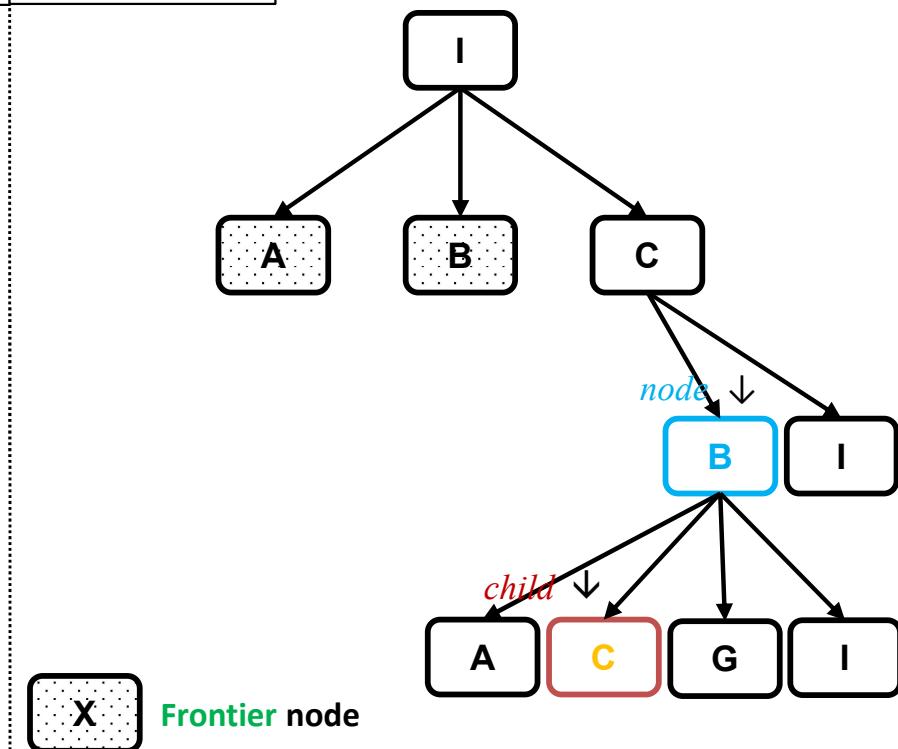
 add *child* to *frontier*

return failure

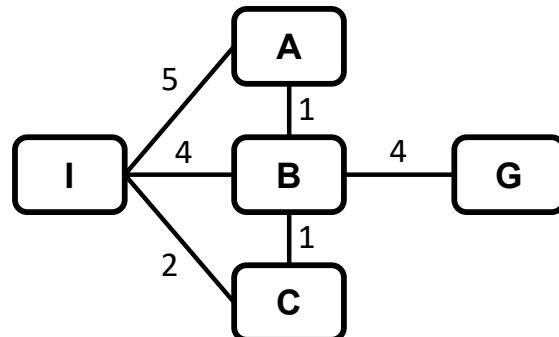
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	I	I			
	Node	B	A			
	f(Node)	7	7			

Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not ~~X~~ **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

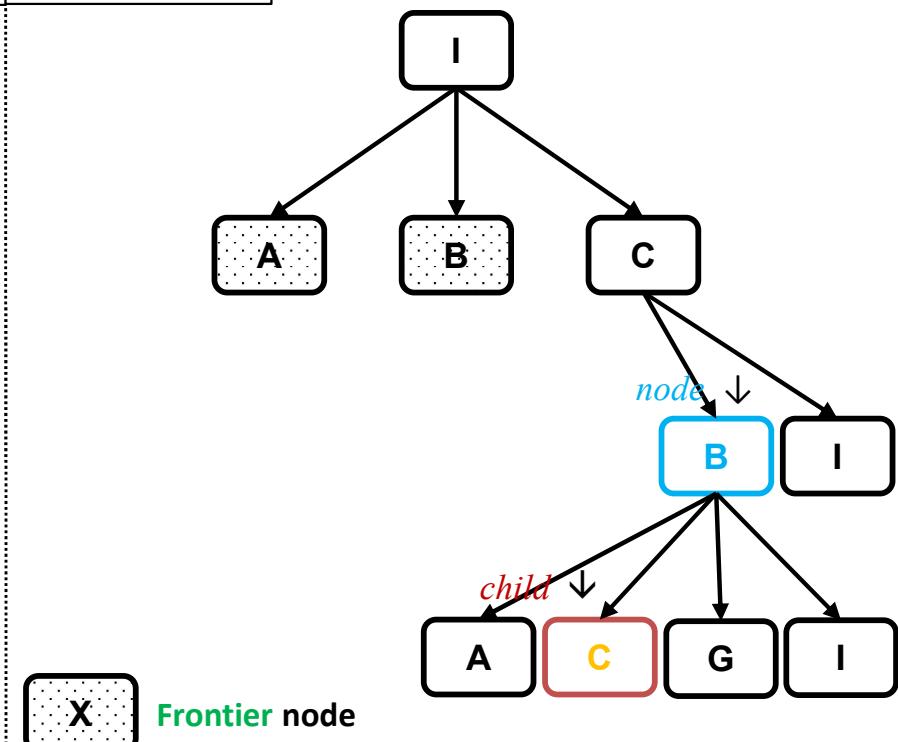
 add *child* to *frontier*

return failure

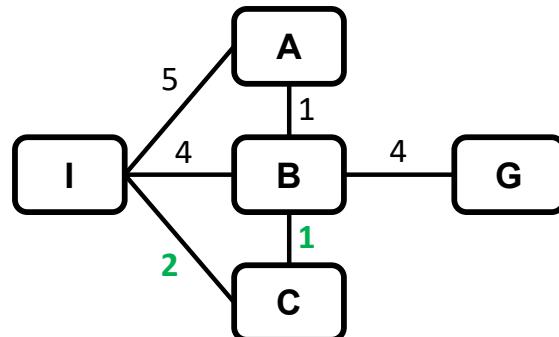
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST ~~<=~~ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

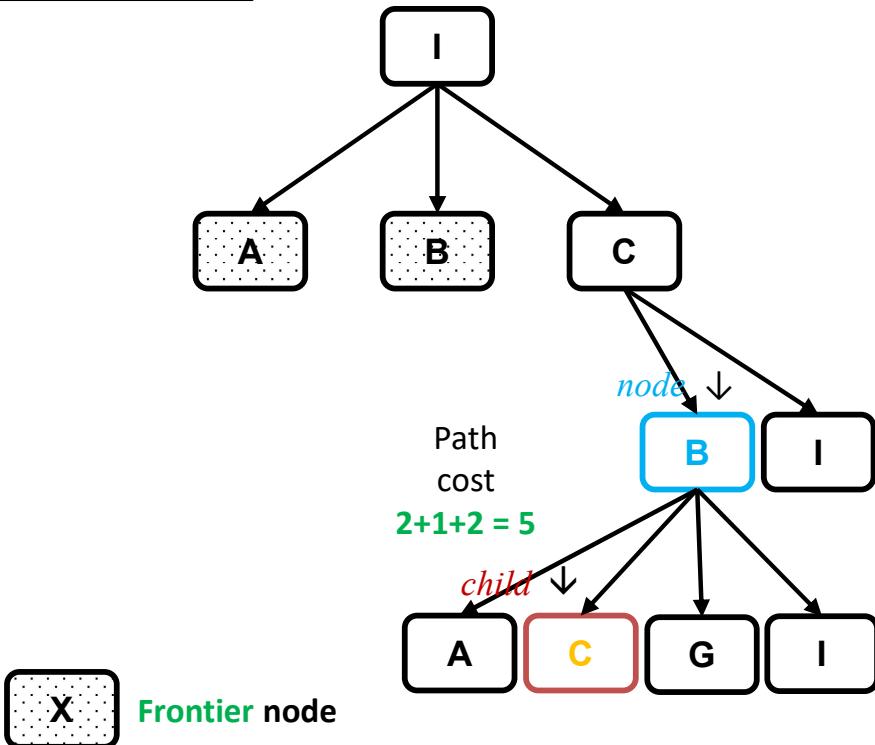
 add *child* to *frontier*

 return failure

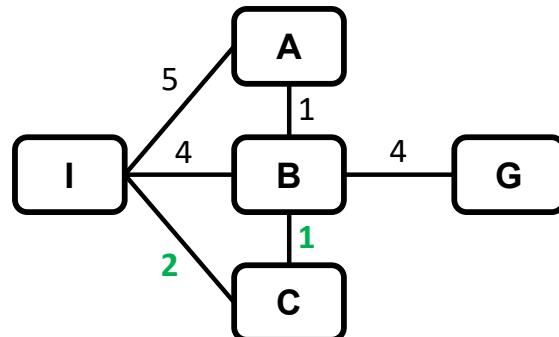
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* ~~<=~~ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

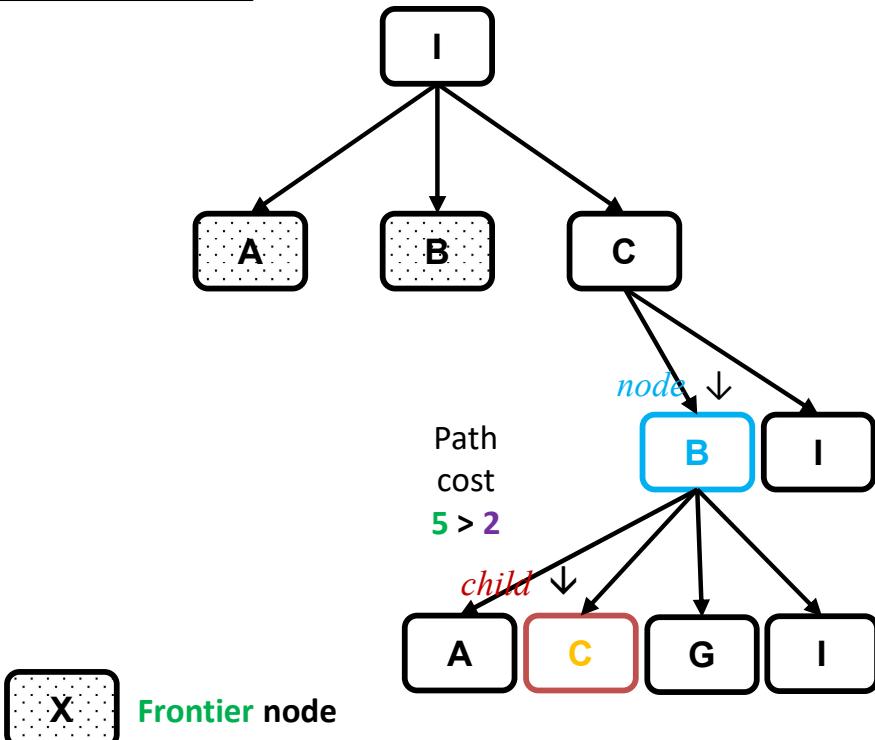
 add *child* to *frontier*

return failure

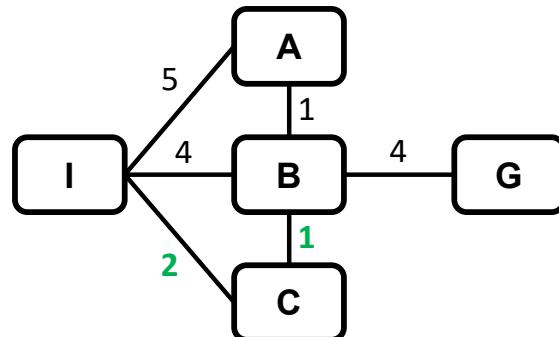
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow child

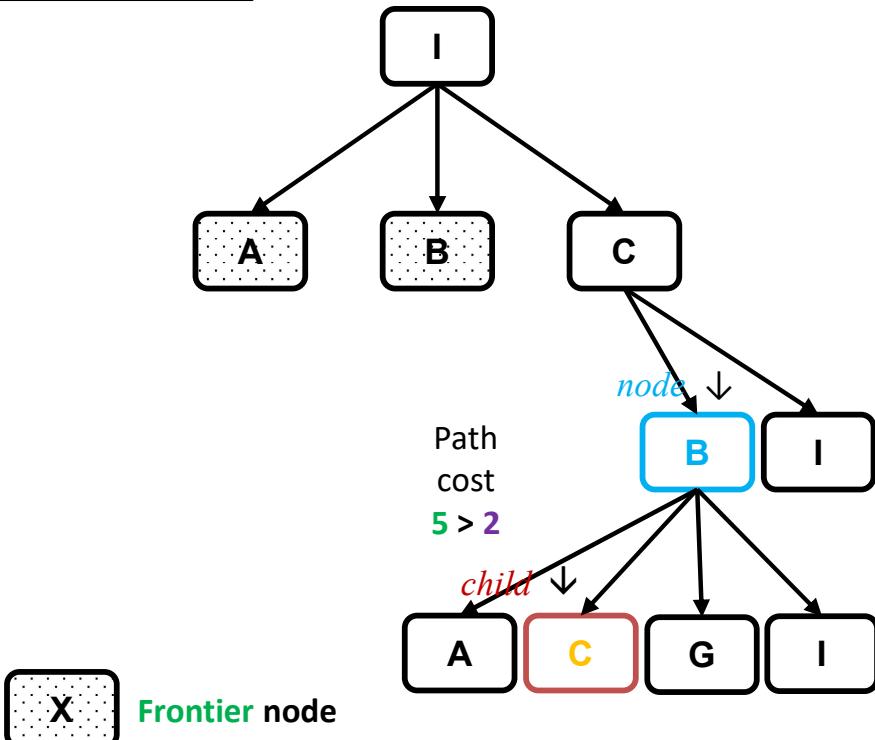
 add *child* to *frontier*

return failure

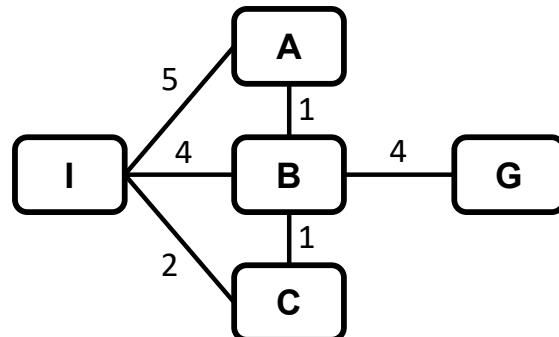
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

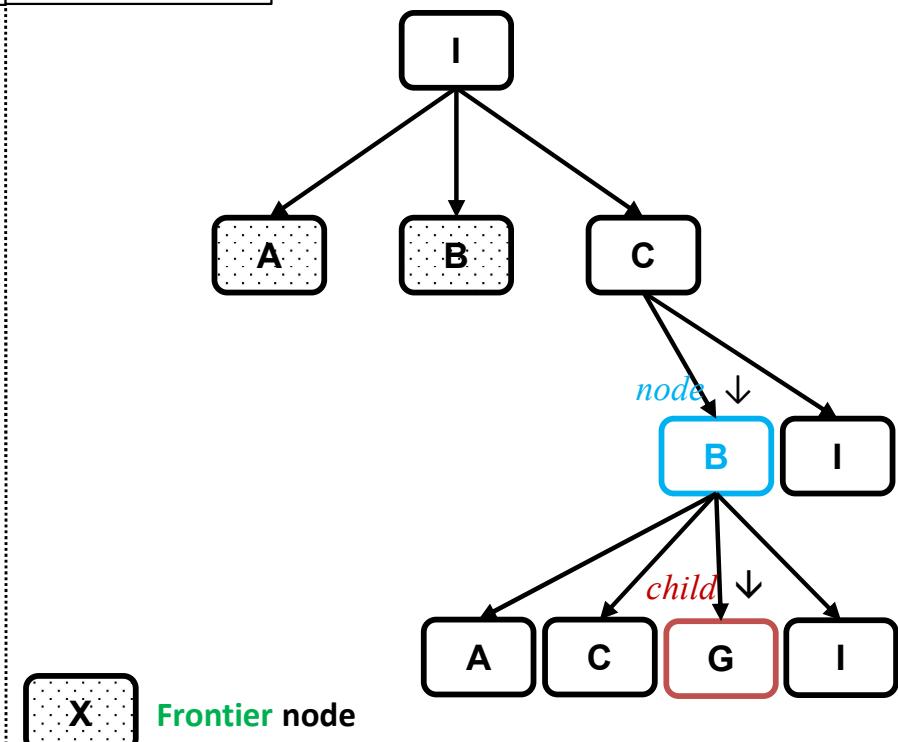
 add *child* to *frontier*

return failure

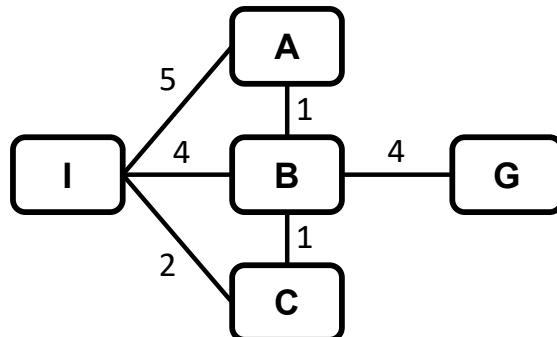
Frontier	Parent	I	I			
	Node	B	A			
	<i>f</i> (Node)	7	7			

Reached	Parent	---	I	C	I		
	Key/State	I	A	B	C		
	Path cost	0	5	3	2		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

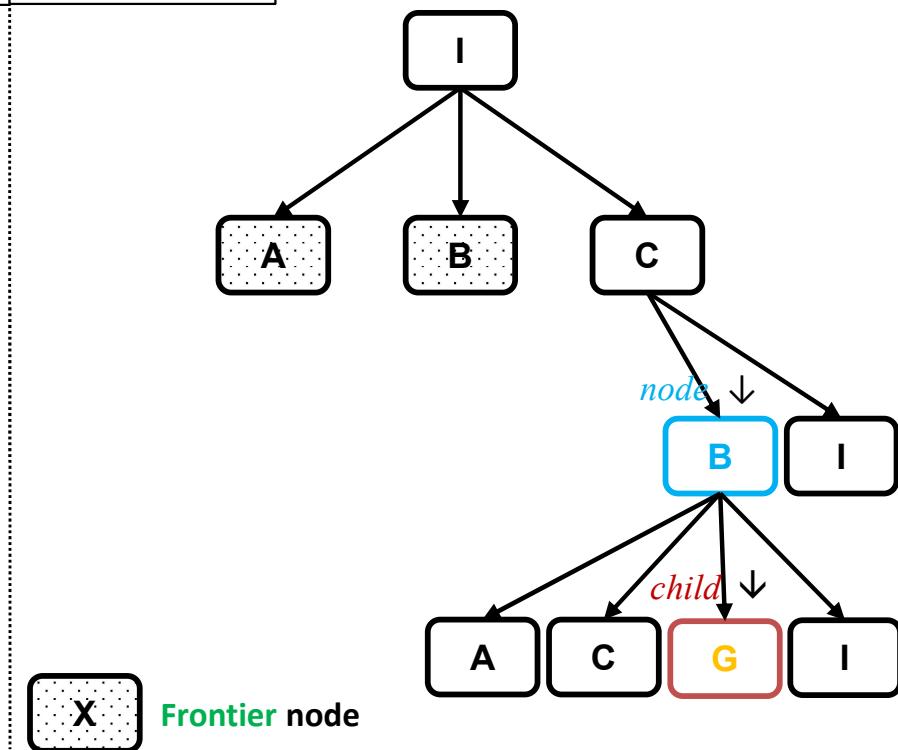
 add *child* to *frontier*

return failure

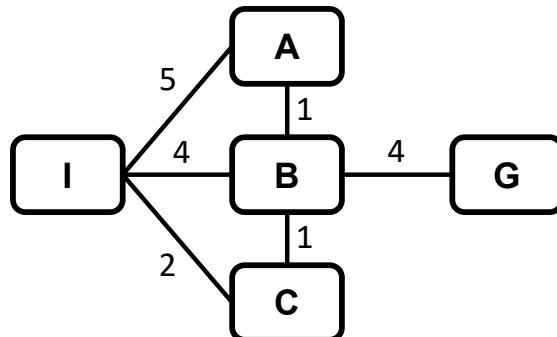
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	I	I			
	Node	B	A			
	f(Node)	7	7			

Reached	Parent	---	I	C	I	
	Key/State	I	A	B	C	
	Path cost	0	5	3	2	



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem.IS-GOAL(node.STATE)* then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child.STATE*

 if *s* is not in *reached* or *child.PATH-COST* $<$ *reached[s].PATH-COST* then

reached[s] \leftarrow *child*

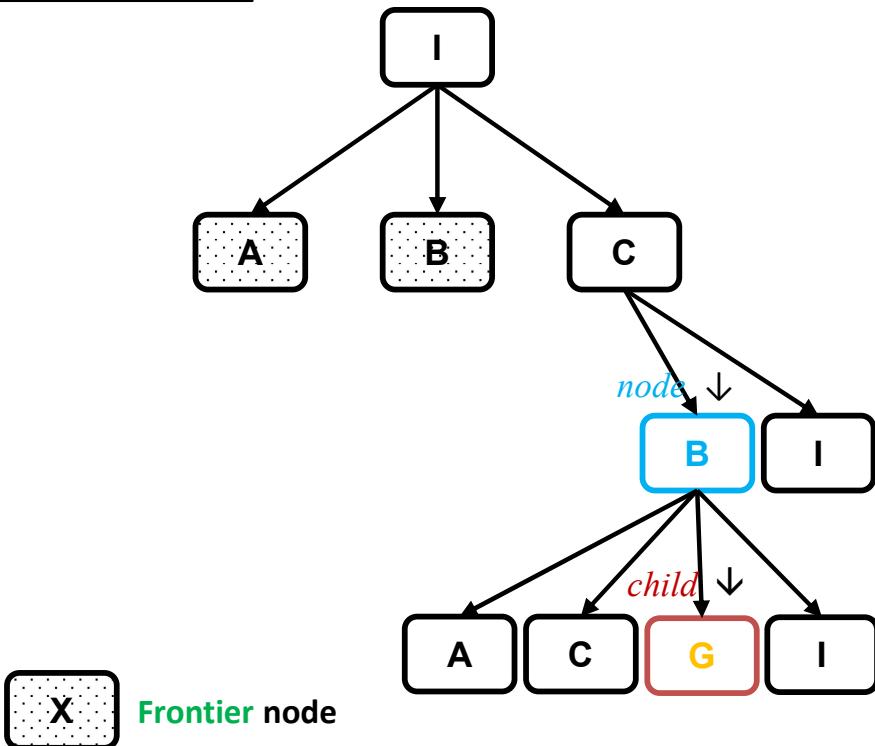
 add *child* to *frontier*

 return failure

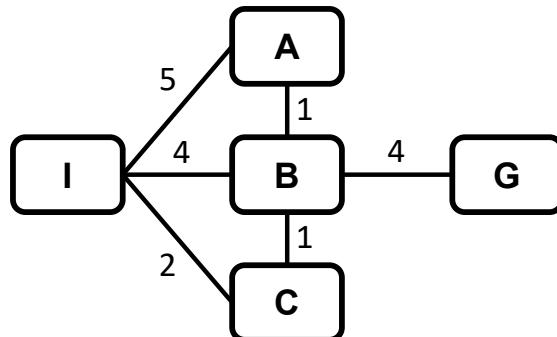
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

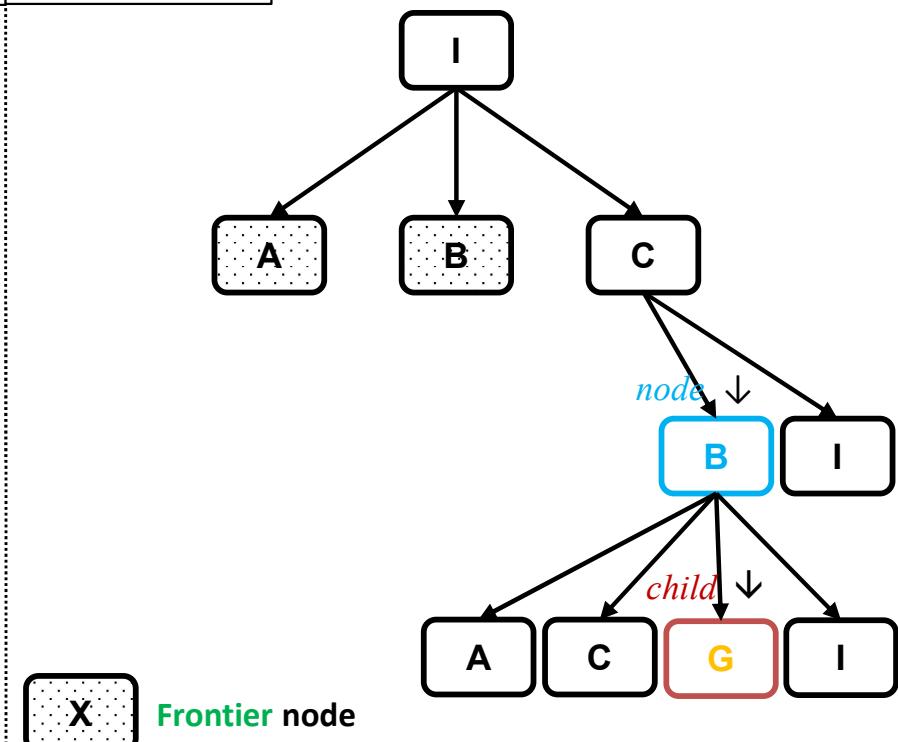
 add *child* to *frontier*

return failure

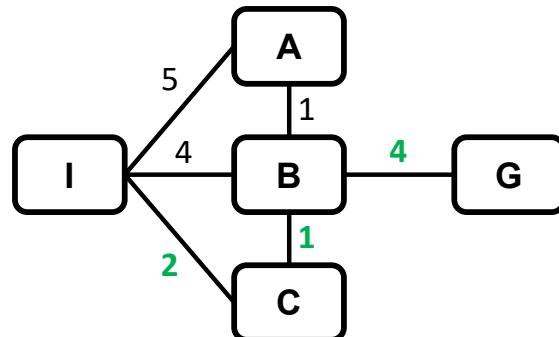
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

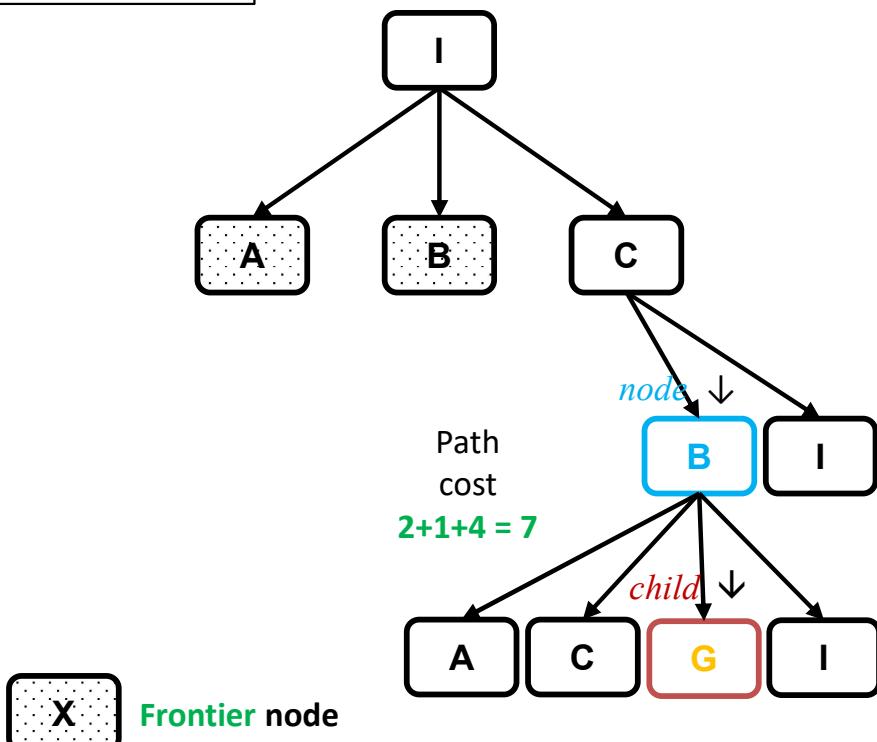
reached[s] \leftarrow *child*

 add *child* to *frontier*

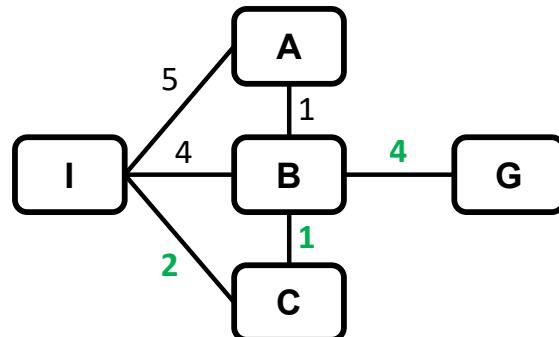
return failure

State Space Graph		Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I			
	Node	B	A			
	<i>f</i> (Node)	7	7			
Reached	Parent	---	I	C	I	
	Key/State	I	A	B	C	
	Path cost	0	5	3	2	



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	I	I				
Node	B	A					
$f(\text{Node})$	7	7					

Reached	Parent	---	I	C	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	3	2	7			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

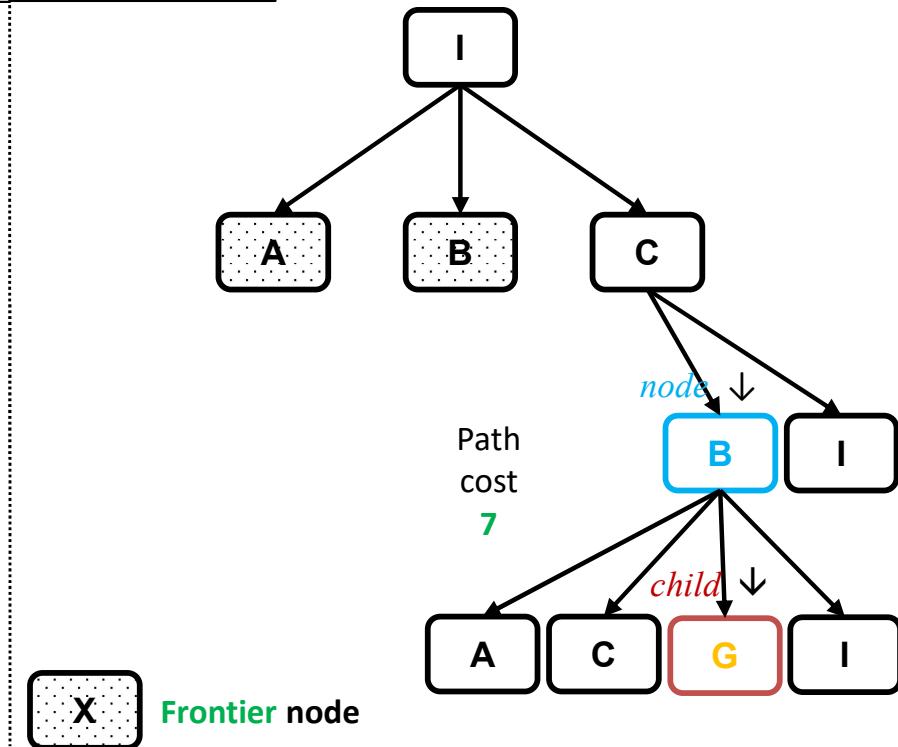
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

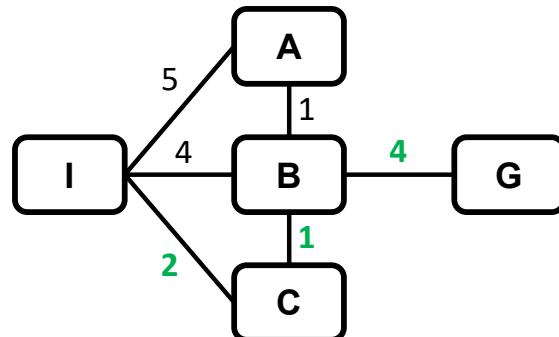
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I			
Node	G		B	A			
f(Node)	7		7	7			

Reached	Parent	---	I	C	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	3	2	7		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node*

 for each *child* in EXPAND(*problem*, *node*) do

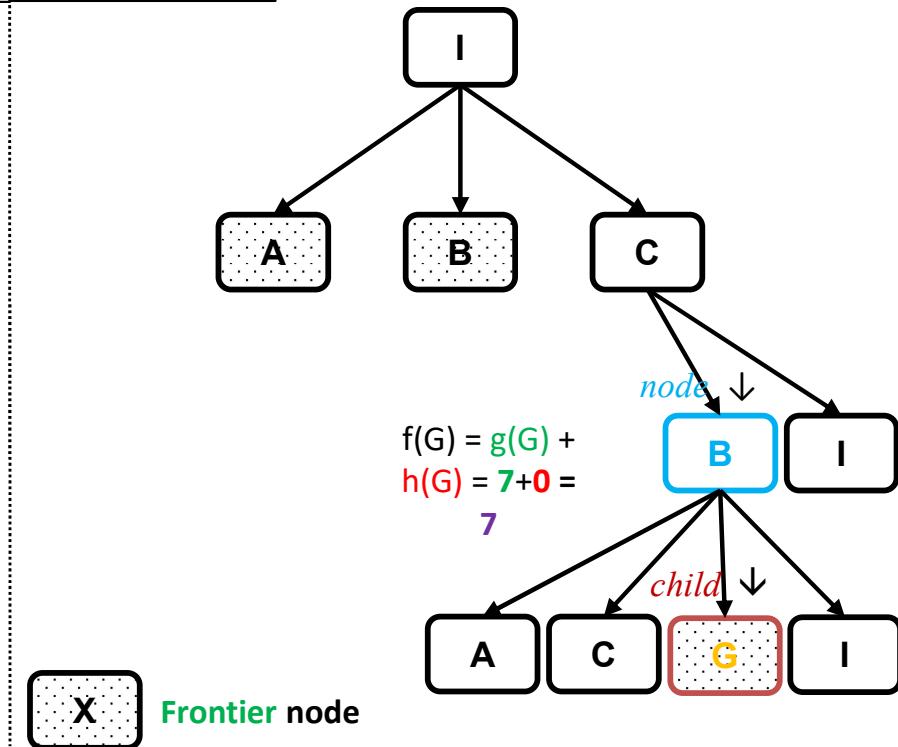
s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

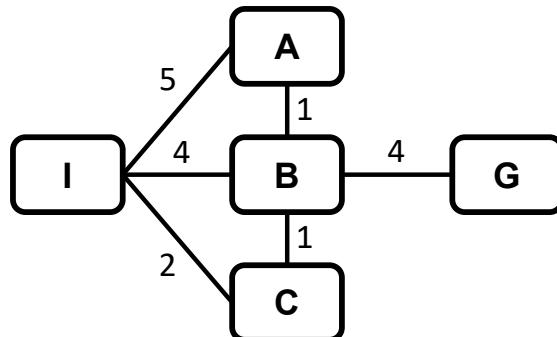
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

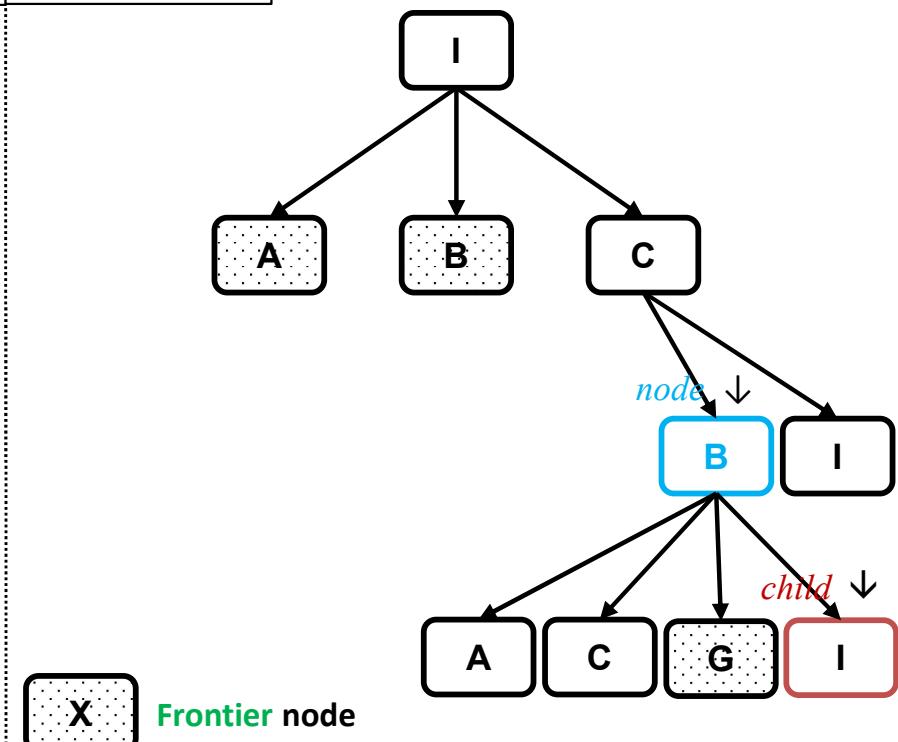
 add *child* to *frontier*

return failure

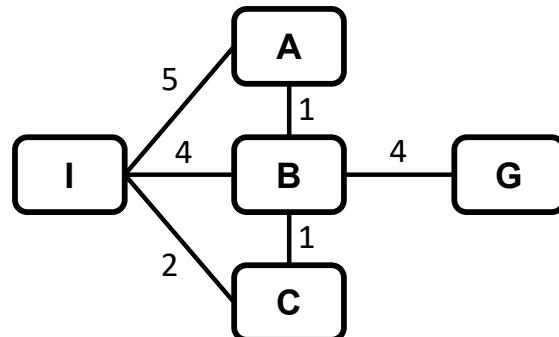
Frontier	Parent	B	I	I			
Node	G	B	A				
f(Node)	7	7	7				

Reached	Parent	---	I	C	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	3	2	7		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I			
Node	G	B	A				
f(Node)	7	7	7				

Reached	Parent	---	I	C	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	3	2	7			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

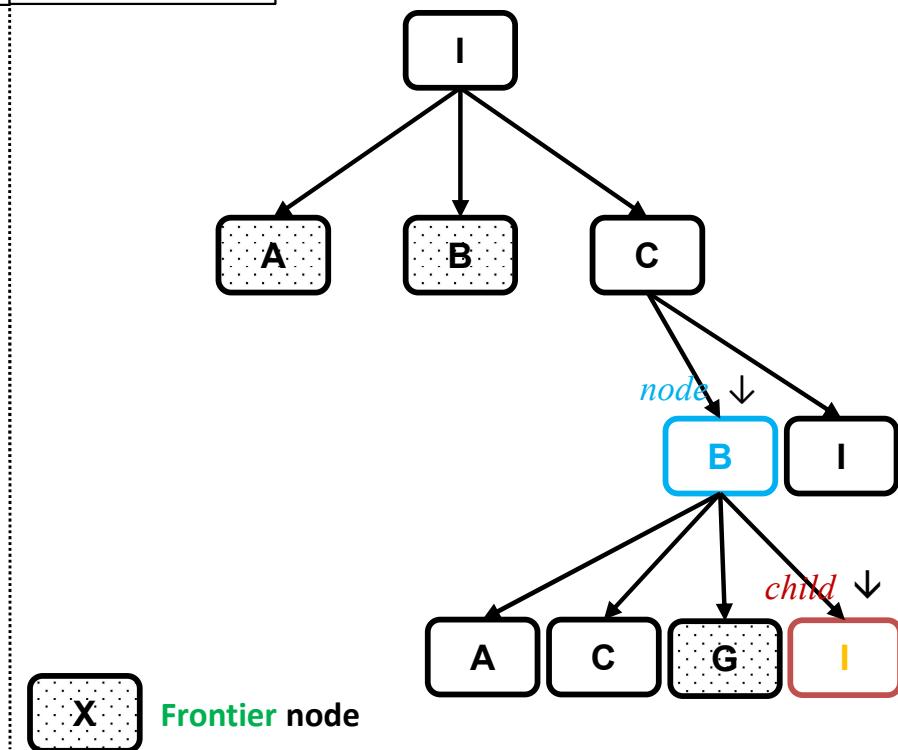
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

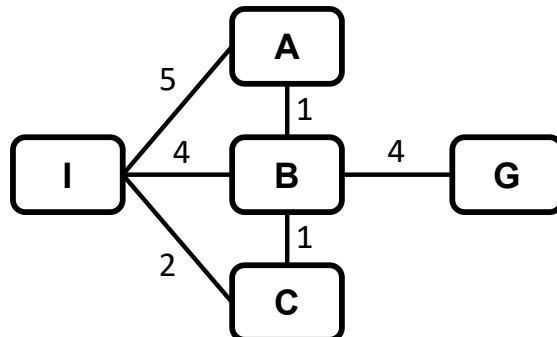
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

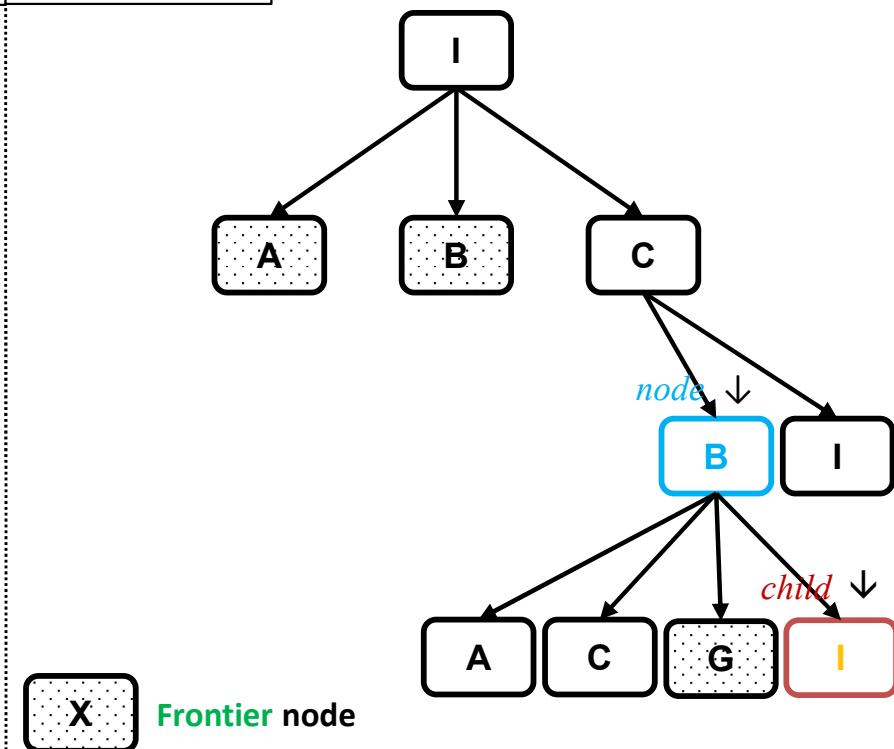
 add *child* to *frontier*

return failure

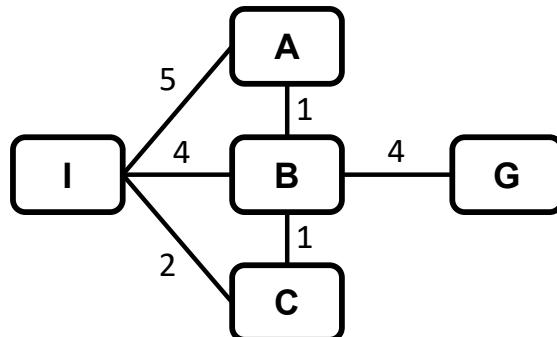
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	B	I	I			
	Node	G	B	A			
	<i>f</i> (Node)	7	7	7			

Reached	Parent	---	I	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	5	3	2	7		



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I			
Node	G	B	A				
f(Node)	7	7	7				

Reached	Parent	---	I	C	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	3	2	7			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(*STATE*=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL*(*node.STATE*) **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

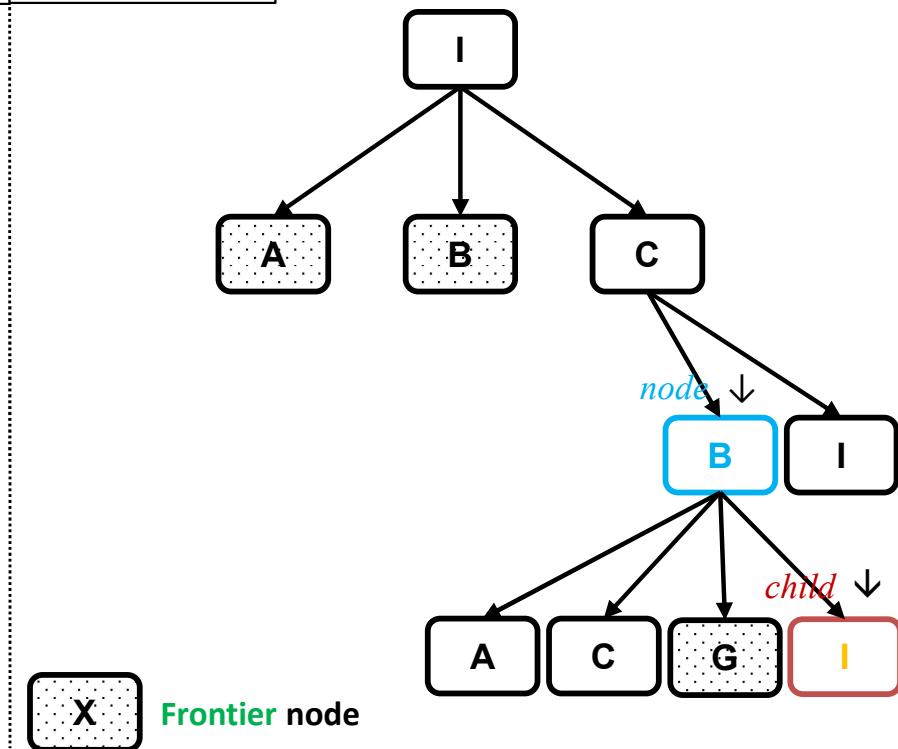
s \leftarrow *child.STATE*

if *s* is not ~~X~~ *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

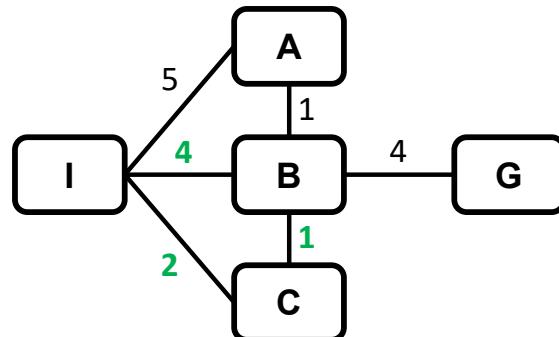
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I			
Node	G	B	A				
f(Node)	7	7	7				

Reached	Parent	---	I	C	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	3	2	7		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
node ← NODE(STATE=problem.INITIAL)
```

```
frontier ← a priority queue ordered by f, with node as an element
```

```
reached ← a lookup table, with one entry key problem.INITIAL and value node
```

```
while not IS-EMPTY(frontier) do
```

```
    node ← POP(frontier)
```

```
    if problem.IS-GOAL(node.STATE) then return node
```

```
    for each child in EXPAND(problem, node) do
```

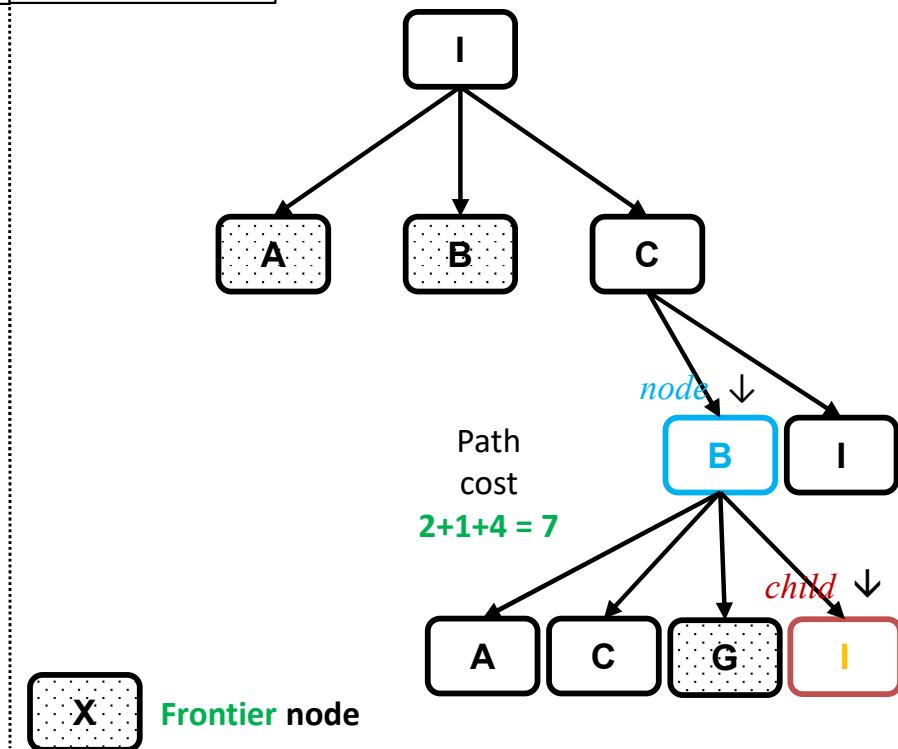
```
        s ← child.STATE
```

```
        if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
```

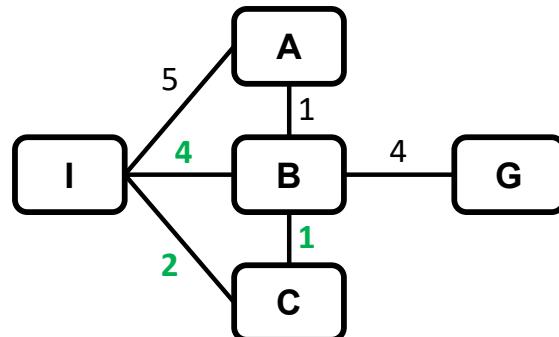
```
            reached[s] ← child
```

```
            add child to frontier
```

```
return failure
```



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I			
Node	G	B	A				
f(Node)	7	7	7				

Reached	Parent	---	I	C	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	3	2	7			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

```
function BEST-FIRST-SEARCH(problem, f)
```

```
node ← NODE(STATE=problem.INITIAL)
```

```
frontier ← a priority queue ordered by  $f$ , with  $node$  as an element
```

```
reached ← a lookup table, with one entry key problem.INITIAL and value  $node$ 
```

```
while not IS-EMPTY(frontier) do
```

```
node ← POP(frontier)
```

```
if problem.IS-GOAL(node.STATE) then return node
```

```
for each child in EXPAND(problem, node) do
```

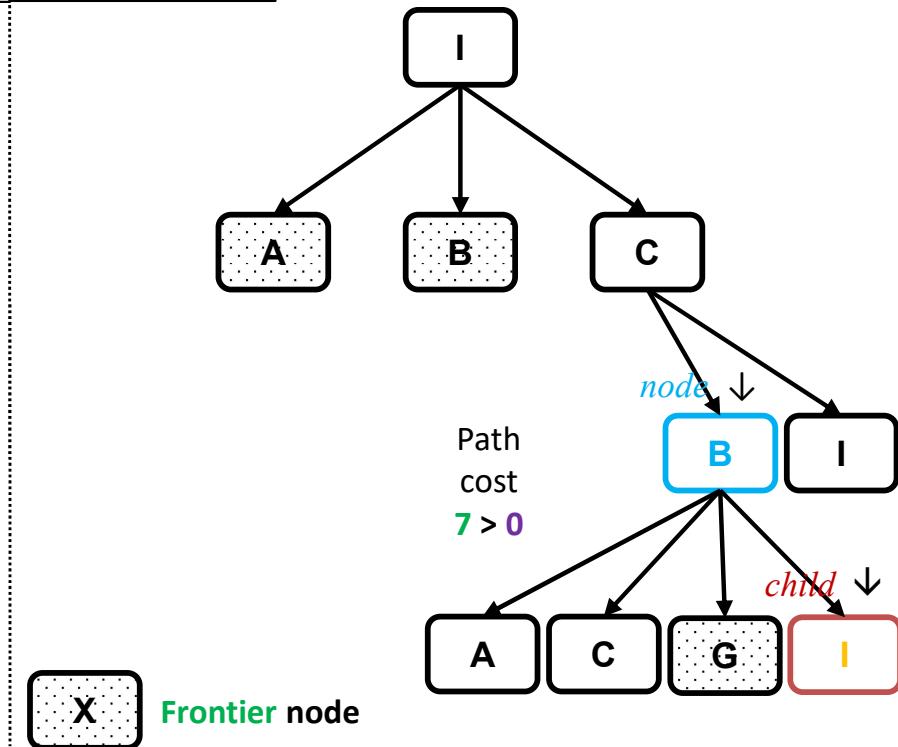
```
   $s \leftarrow child.STATE$ 
```

```
  if  $s$  is not in reached or child.PATH-COST < reached[ $s$ ].PATH-COST then
```

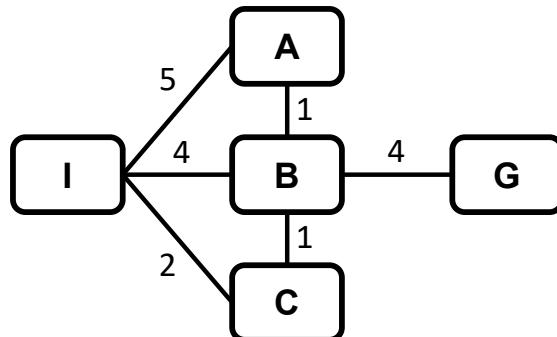
```
    reached[ $s$ ] ← child
```

```
    add child to frontier
```

```
return failure
```



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I			
Node	G	B	A				
f(Node)	7	7	7				

Reached	Parent	---	I	C	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	3	2	7		

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

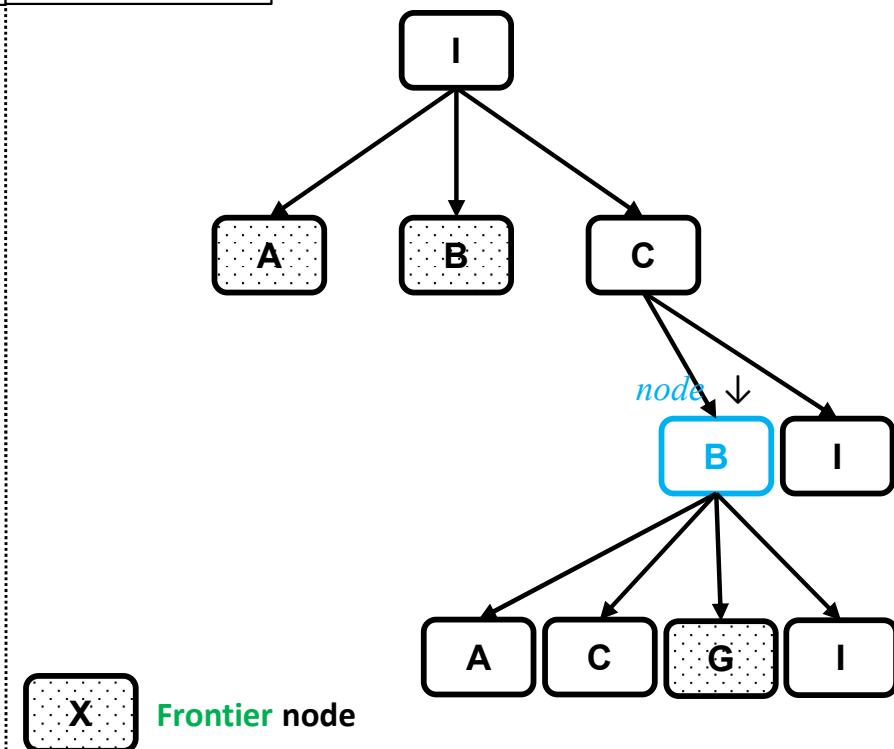
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

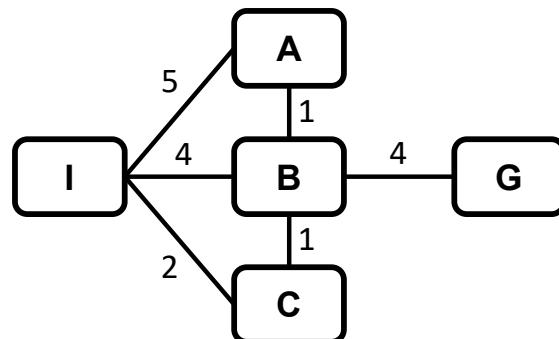
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I			
Node	G	B	A				
f(Node)	7	7	7				

Reached	Parent	---	I	C	I	B		
Key/State	I	A	B	C	G			
Path cost	0	5	3	2	7			

State Space Graph	Frontier / Reached
Algorithm	Search Tree

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

NOT EMPTY!

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

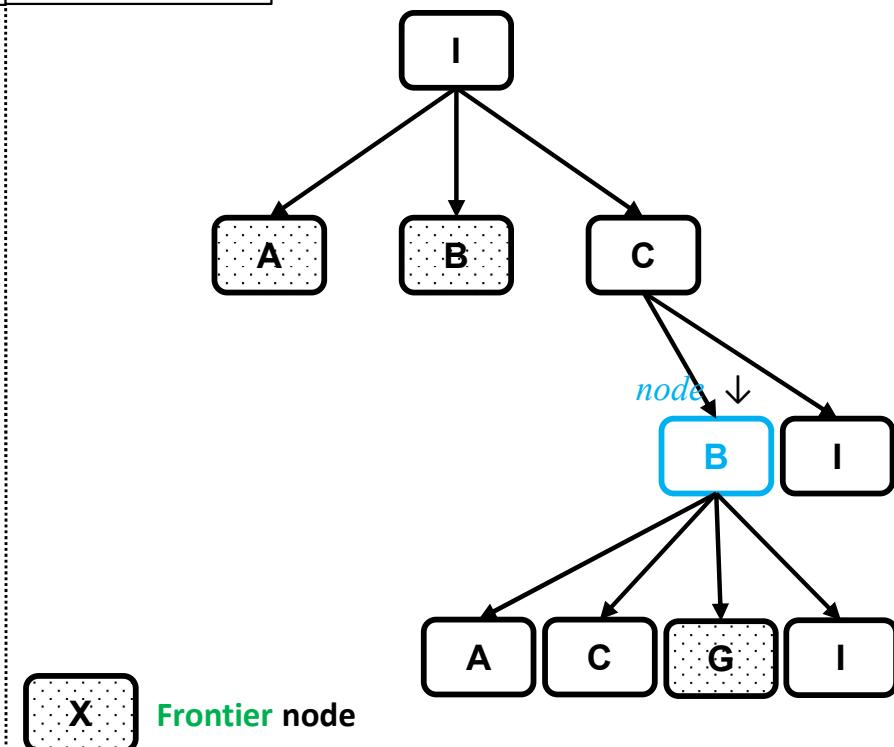
s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

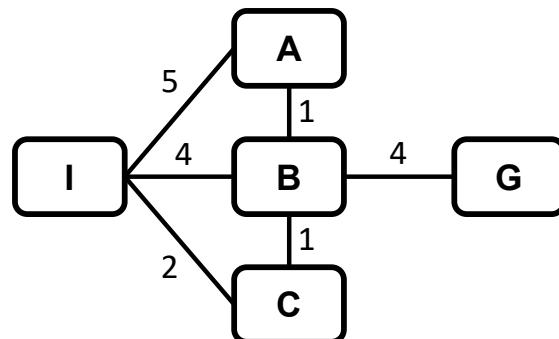
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

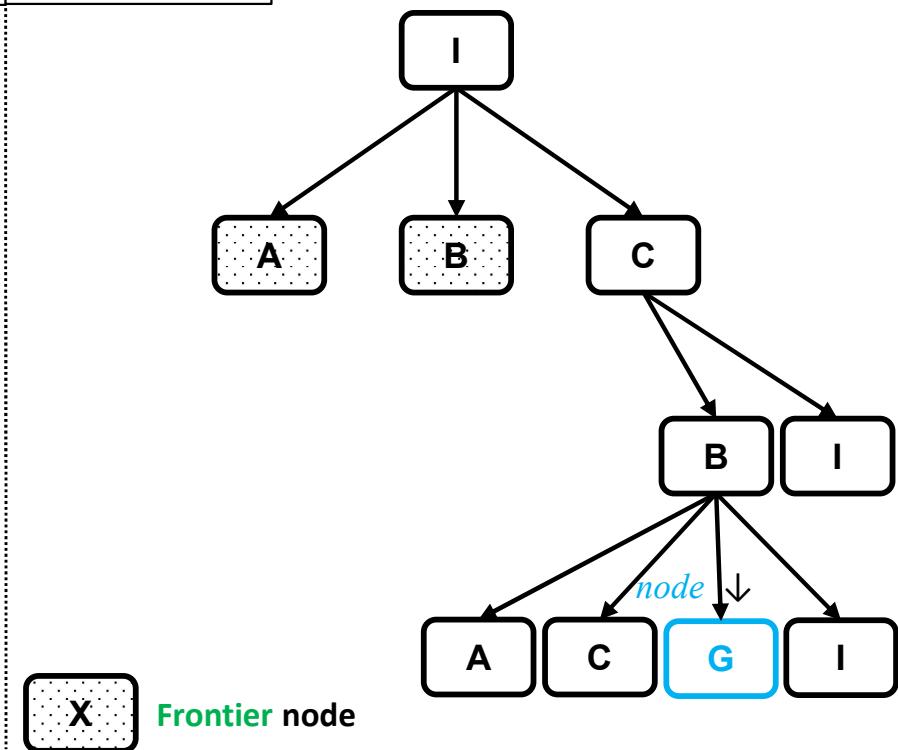
 add *child* to *frontier*

return failure

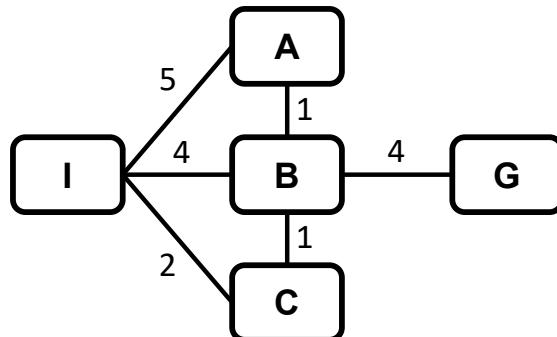
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	3	2	7		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node*

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

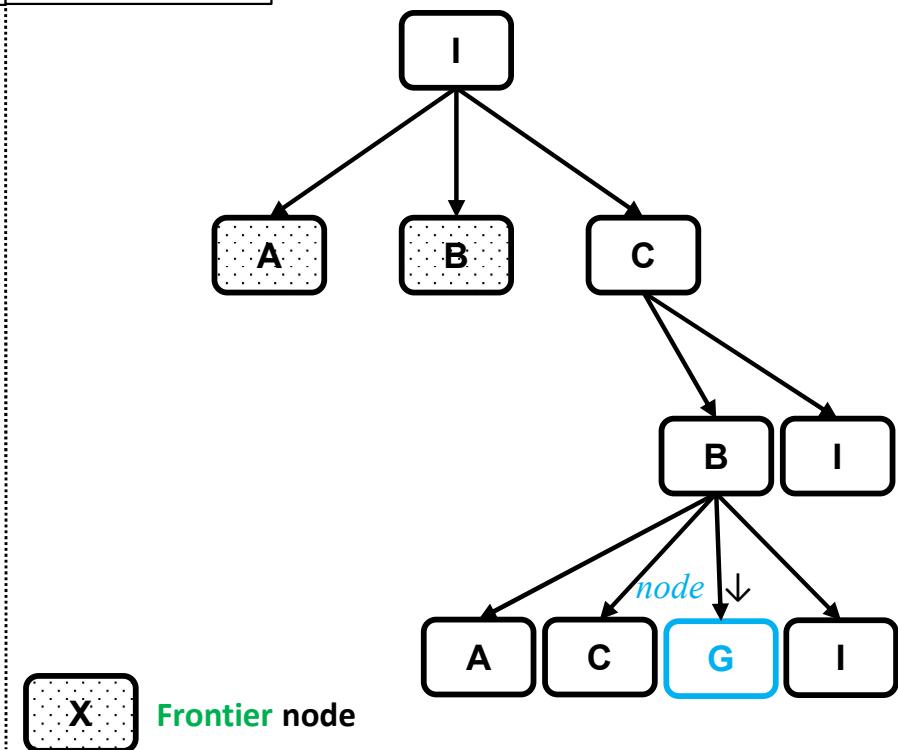
 add *child* to *frontier*

return failure

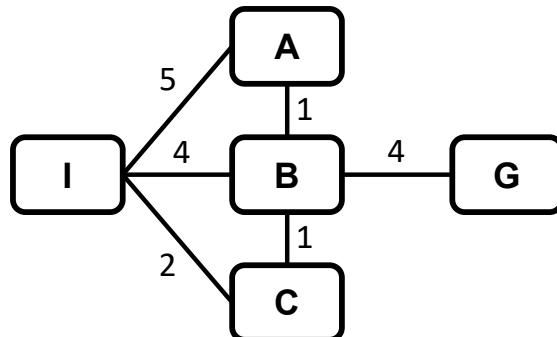
Frontier	Parent	I	I				
Node	B	A					
f(Node)	7	7					

Reached	Parent	---	I	C	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	3	2	7		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem.INITIAL*)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem.INITIAL* and value *node*

while not IS-EMPTY(*frontier*) **do**

node \leftarrow POP(*frontier*)

if *problem.IS-GOAL(node.STATE)* **then return** *node* **TRUE!**

for each *child* in EXPAND(*problem*, *node*) **do**

s \leftarrow *child.STATE*

if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

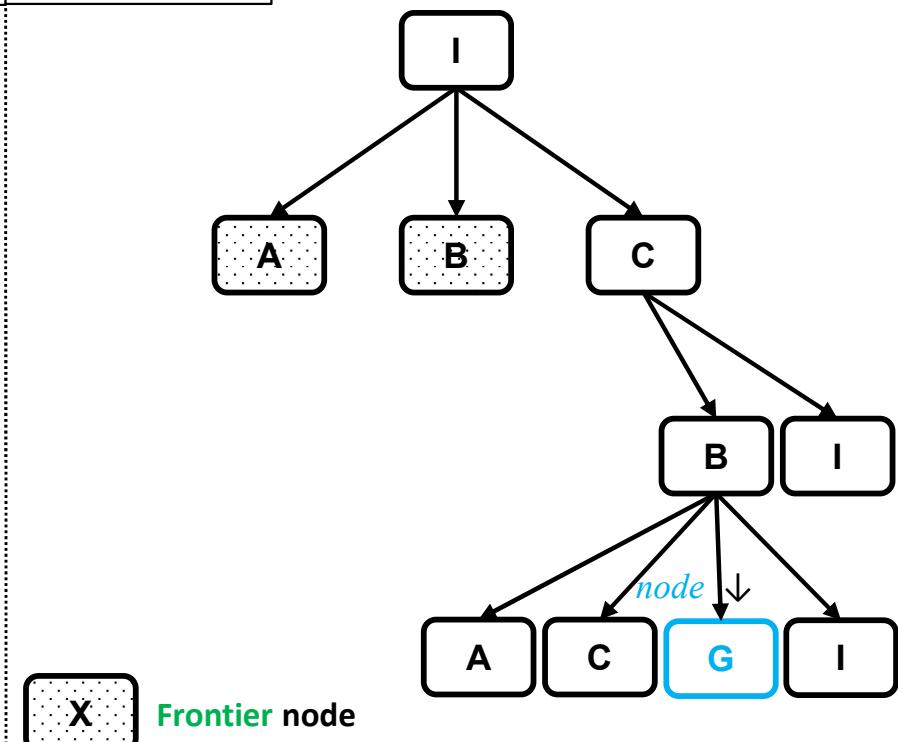
 add *child* to *frontier*

return failure

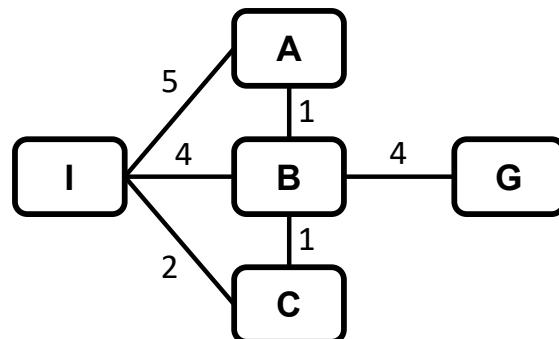
Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

Reached	Parent	---	I	C	I	B	
Key/State	I	A	B	C	G		
Path cost	0	5	3	2	7		

State Space Graph	Frontier / Reached
Algorithm	Search Tree



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

function BEST-FIRST-SEARCH(*problem*, *f*)

node \leftarrow NODE(STATE=*problem*.INITIAL)

frontier \leftarrow a priority queue ordered by *f*, with *node* as an element

reached \leftarrow a lookup table, with one entry key *problem*.INITIAL and value *node*

 while not IS-EMPTY(*frontier*) do

node \leftarrow POP(*frontier*)

 if *problem*.IS-GOAL(*node*.STATE) then return *node* TRUE!

 for each *child* in EXPAND(*problem*, *node*) do

s \leftarrow *child*.STATE

 if *s* is not in *reached* or *child*.PATH-COST $<$ *reached*[*s*].PATH-COST then

reached[*s*] \leftarrow *child*

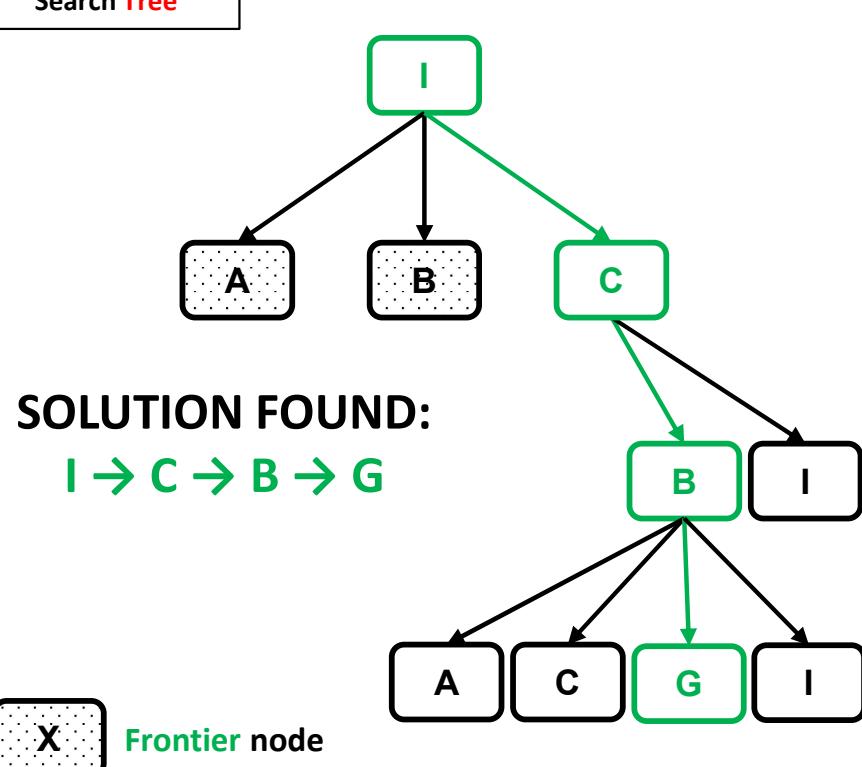
 add *child* to *frontier*

 return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

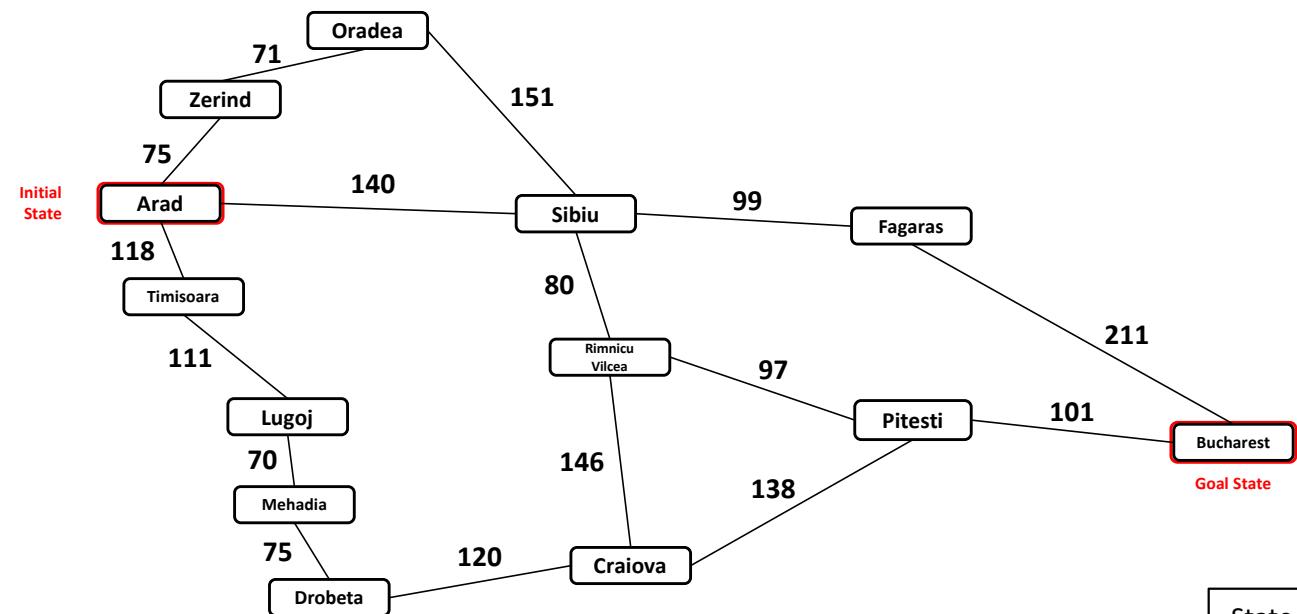
Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

Reached	Parent	---	I	C	I	B	
	Key/State	I	A	B	C	G	
	Path cost	0	5	3	2	7	



Additional A* and GBFS Examples

Romanian Roadtrip: Greedy Best First



- State Visited / expanded tree node
- State Frontier node
- State NOT a Frontier node

Arad

$f(\text{Arad}) = \text{h}(\text{Arad})$

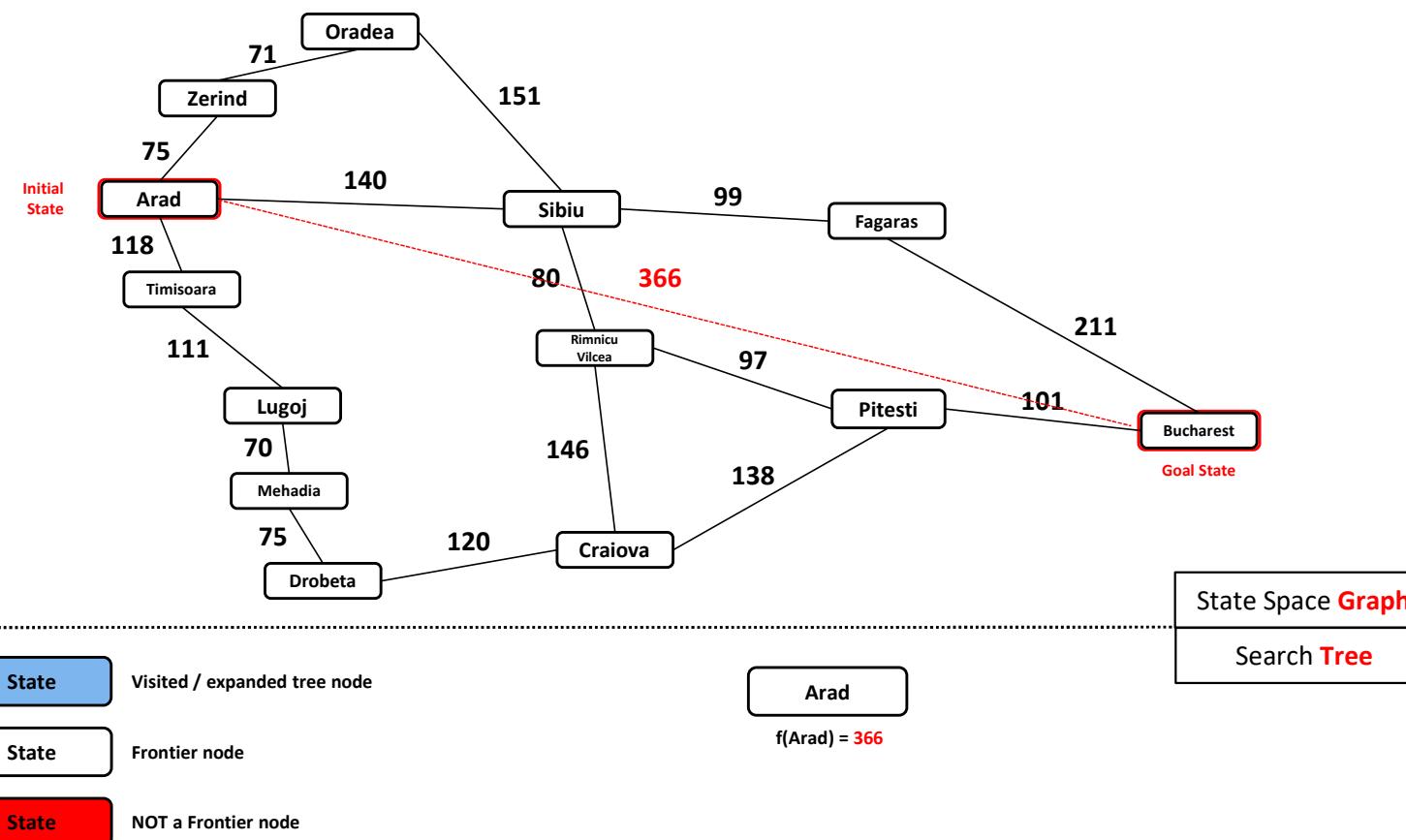
State Space **Graph**

Search Tree

Straight-line distance to Bucharest ($\text{h}(\text{State})$):

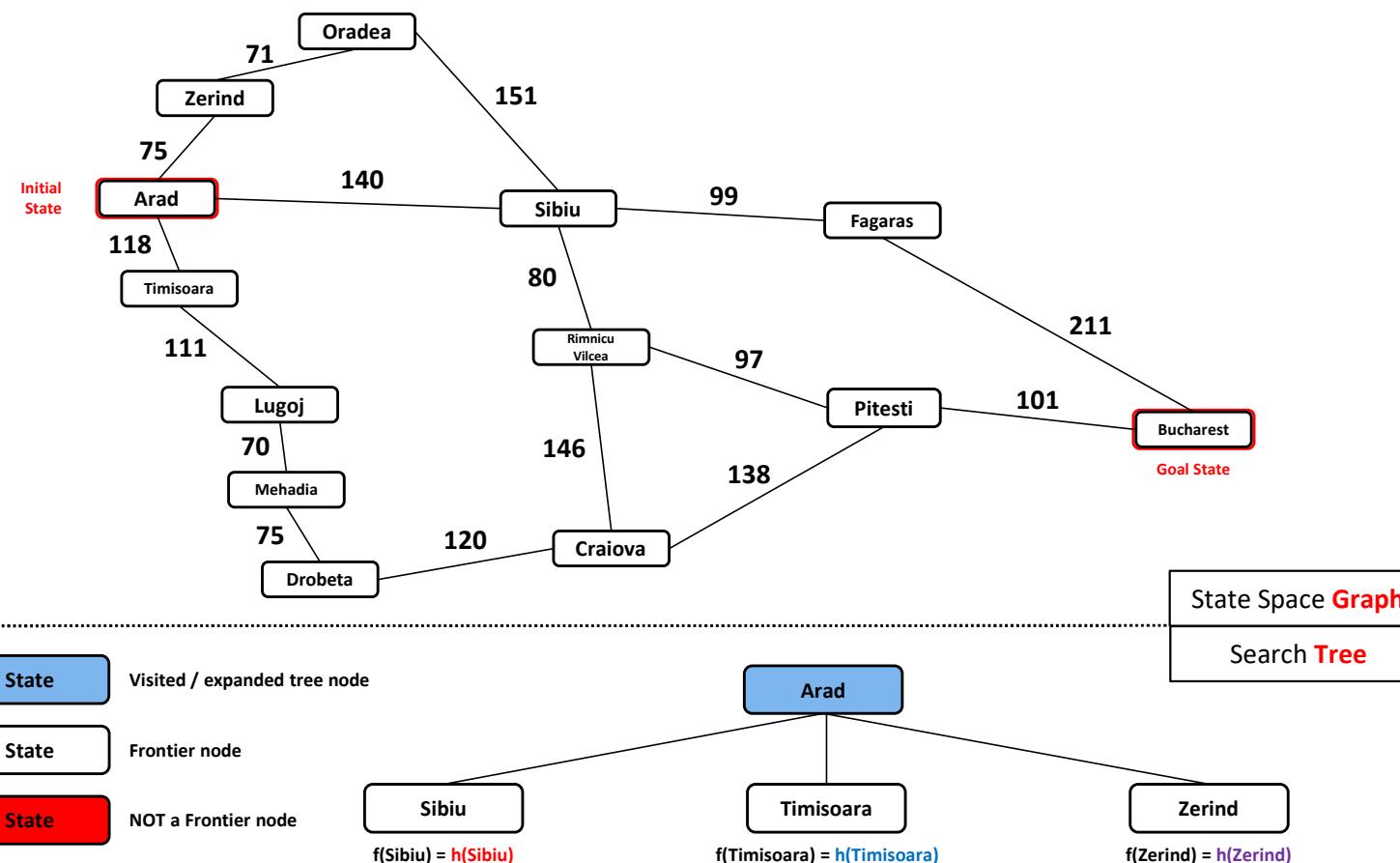
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



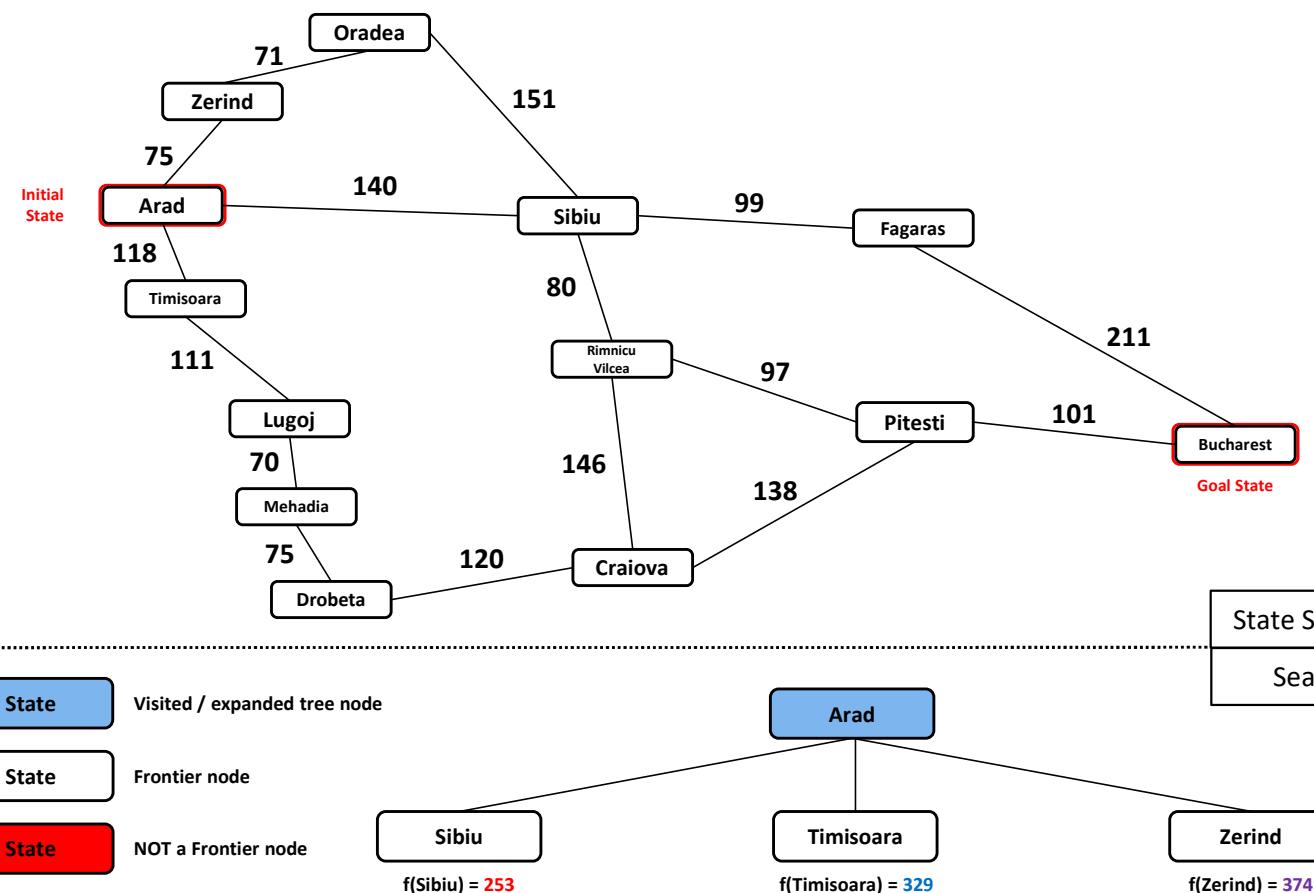
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



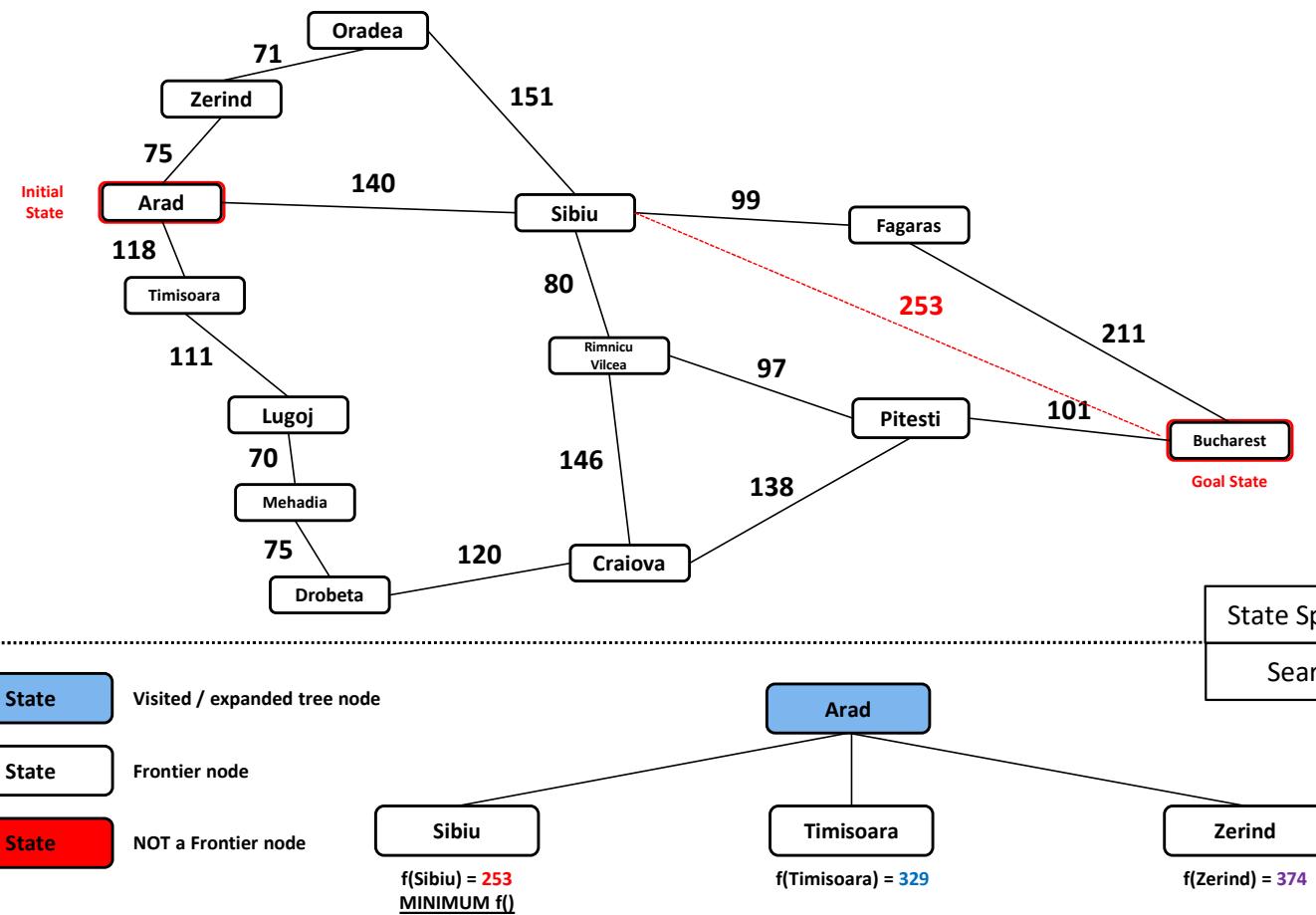
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



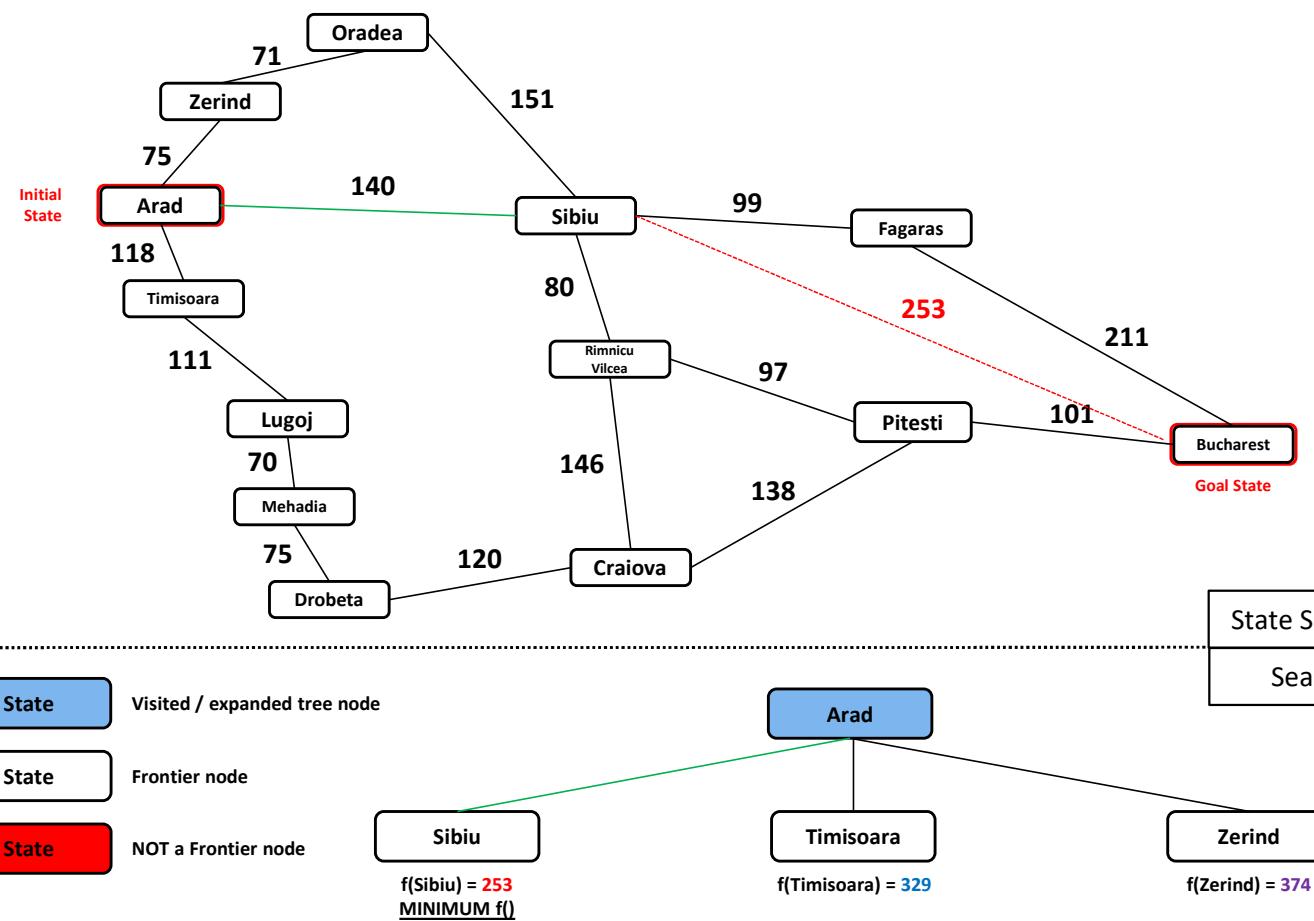
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



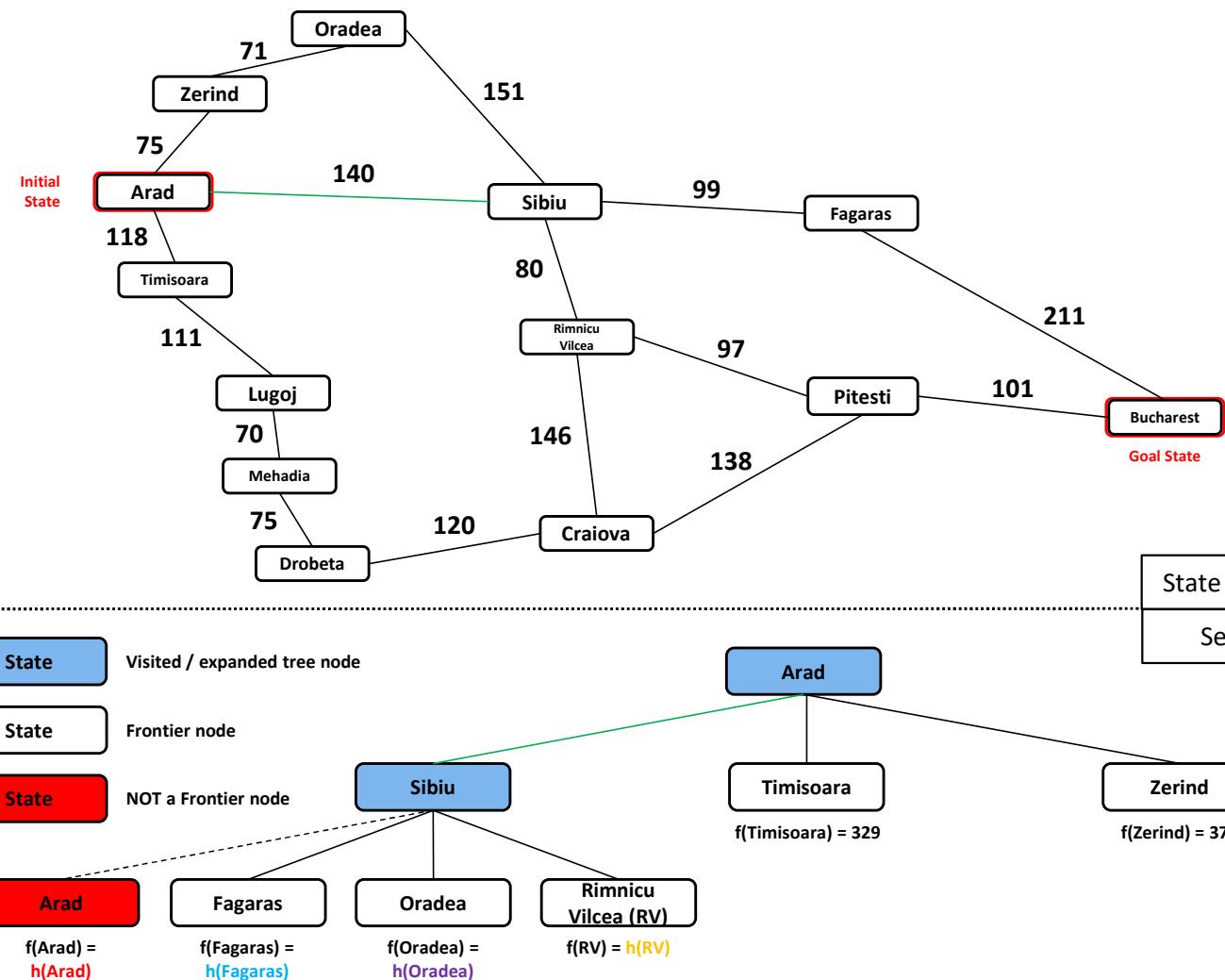
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

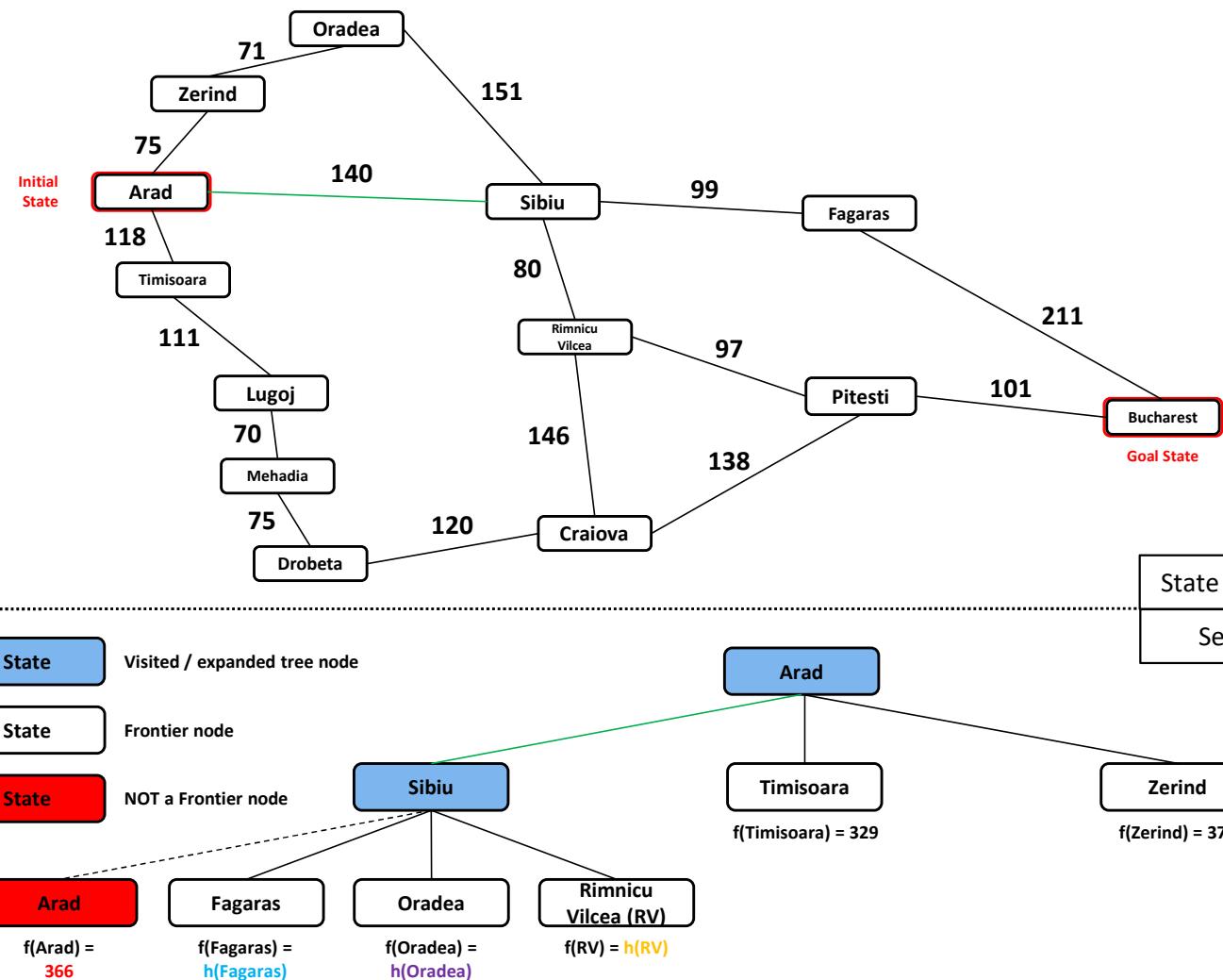
Romanian Roadtrip: Greedy Best First



Straight-line distance to Bucharest ($h(\text{State})$):

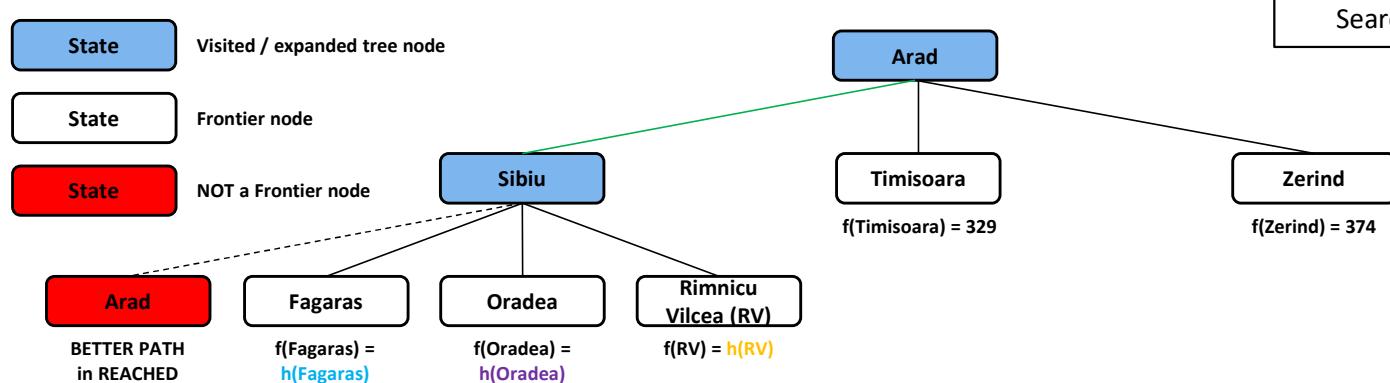
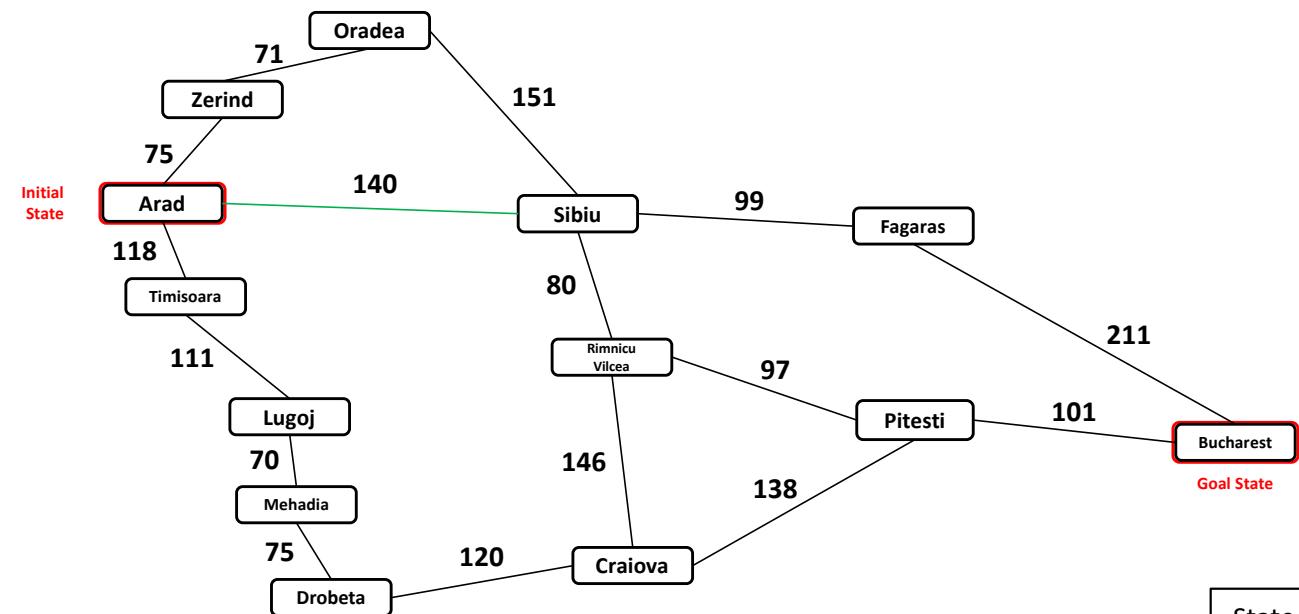
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



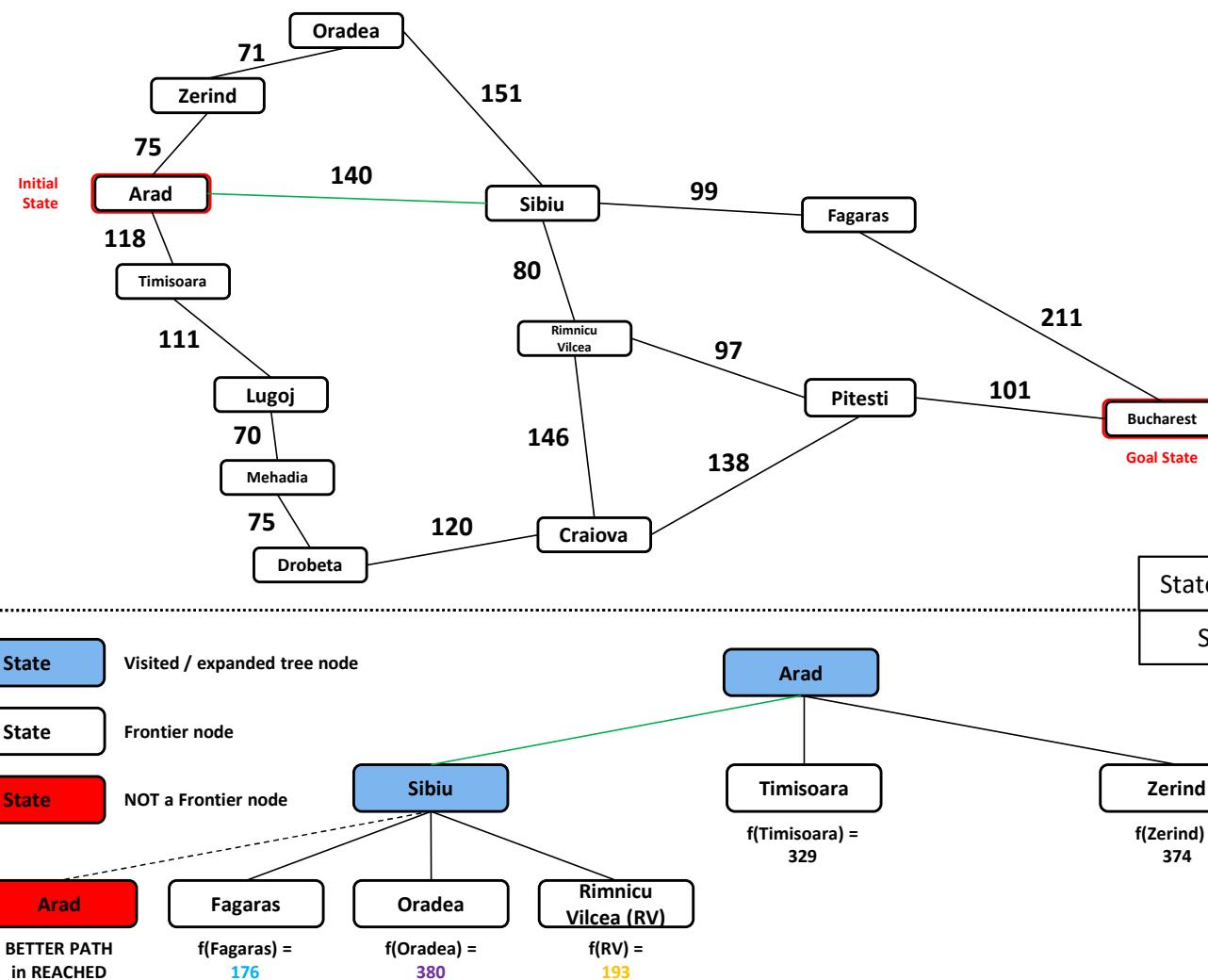
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



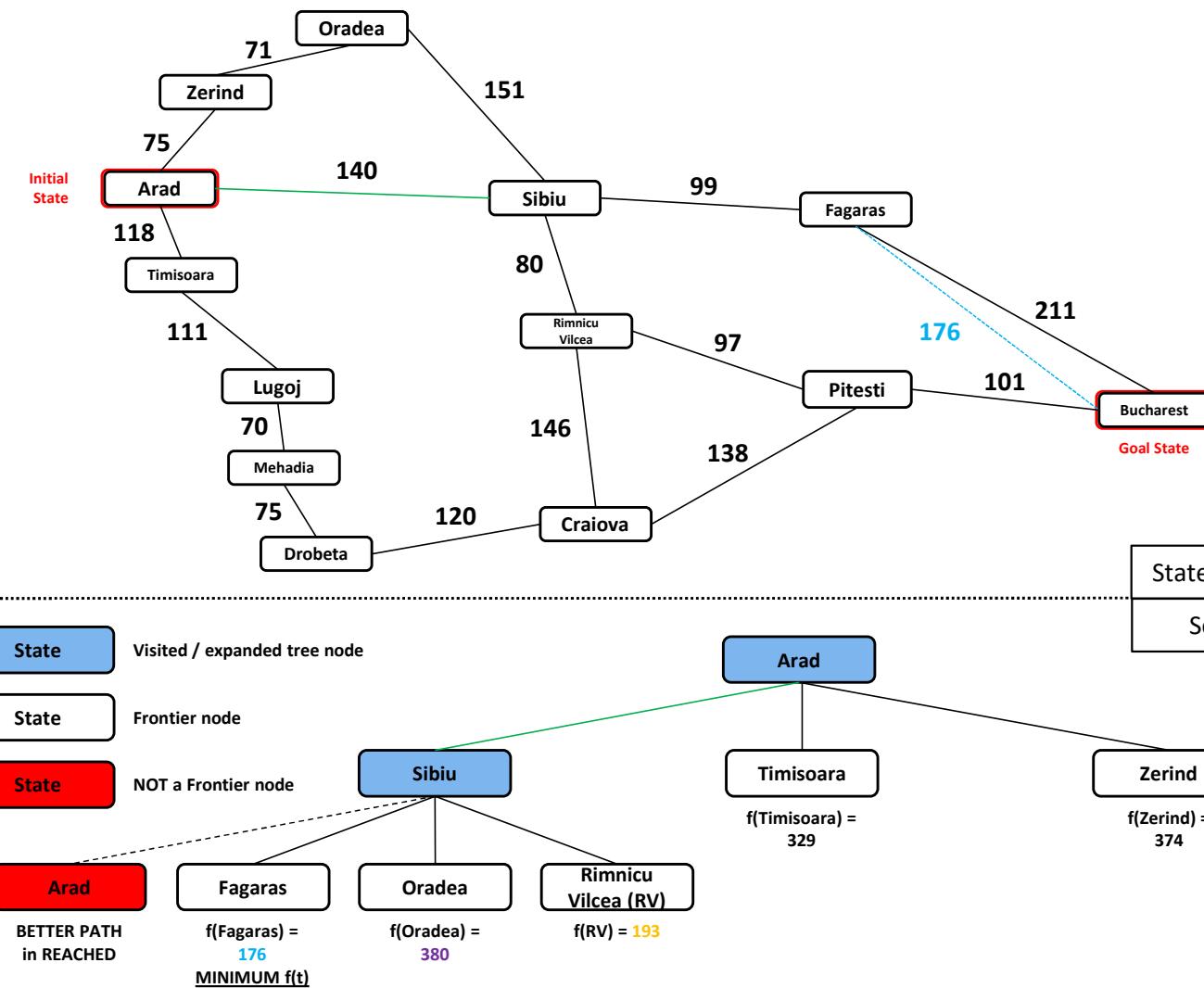
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

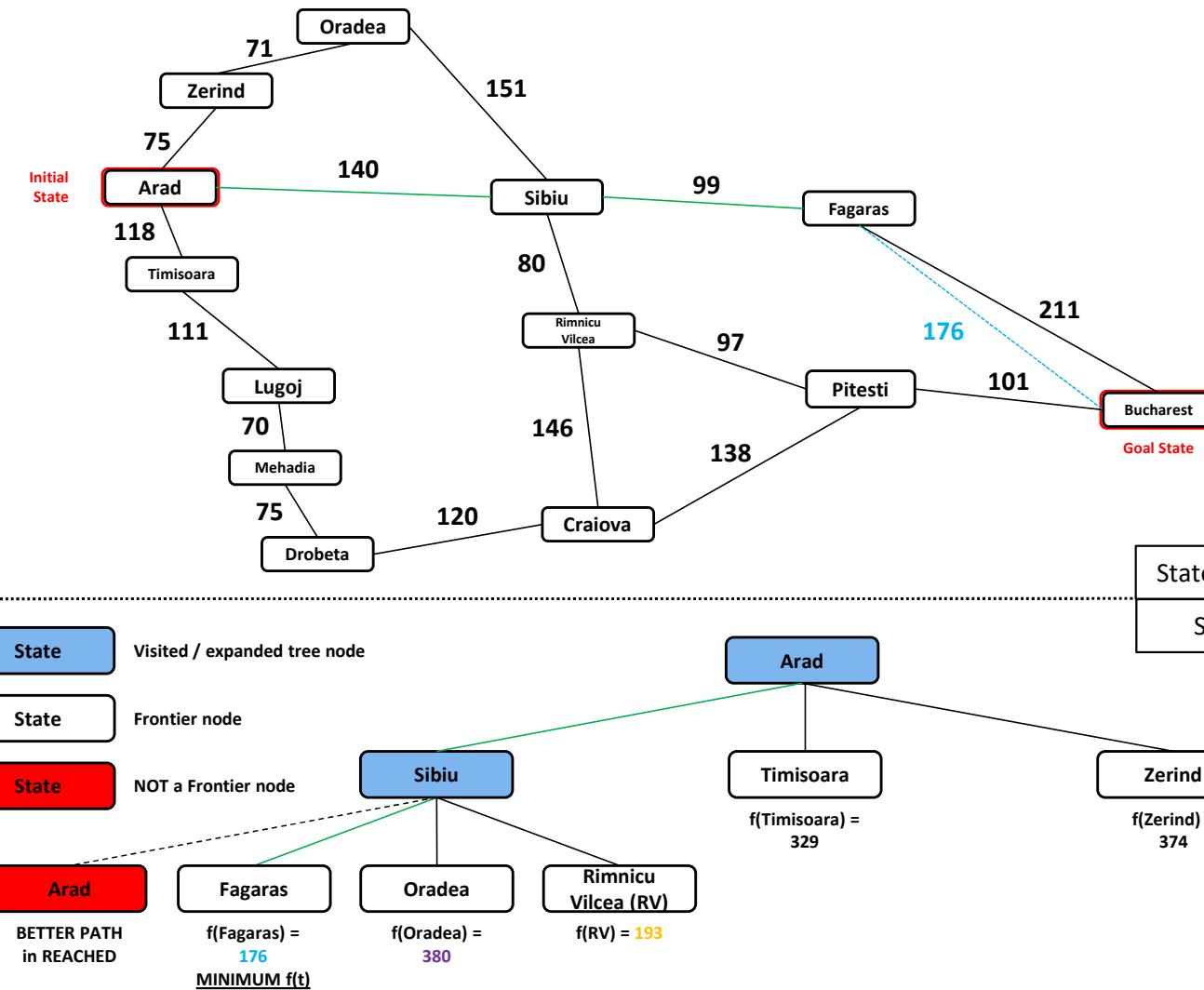
Romanian Roadtrip: Greedy Best First



Straight-line distance to Bucharest ($h(\text{State})$):

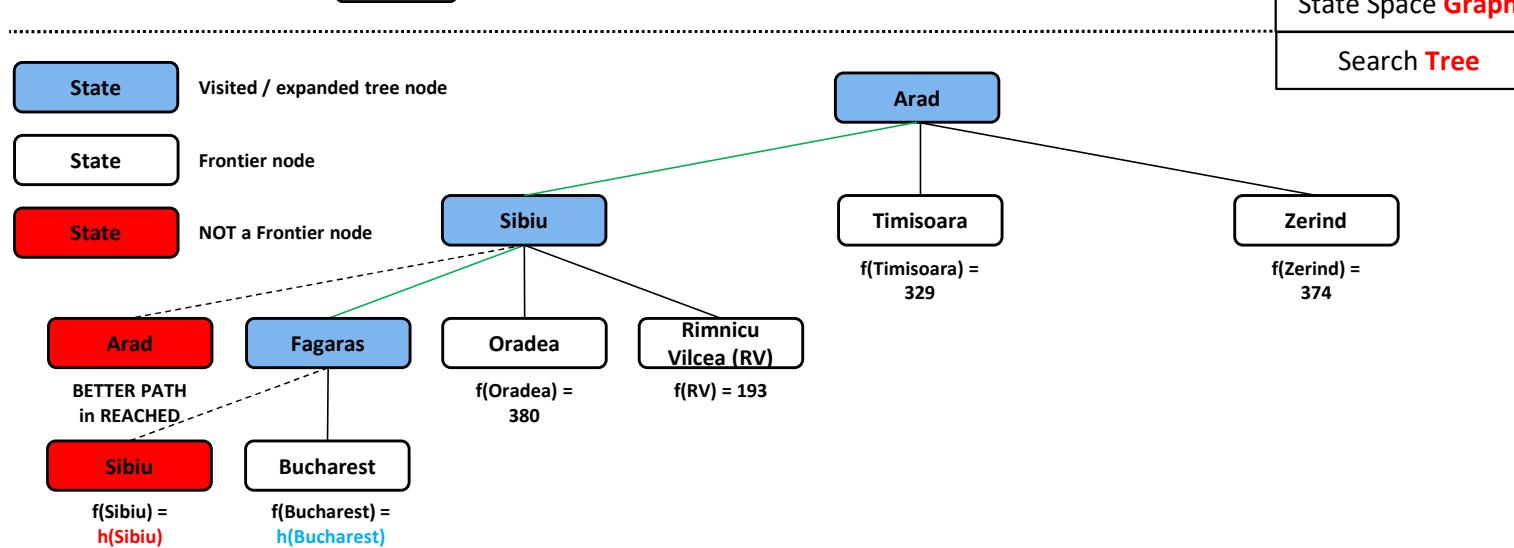
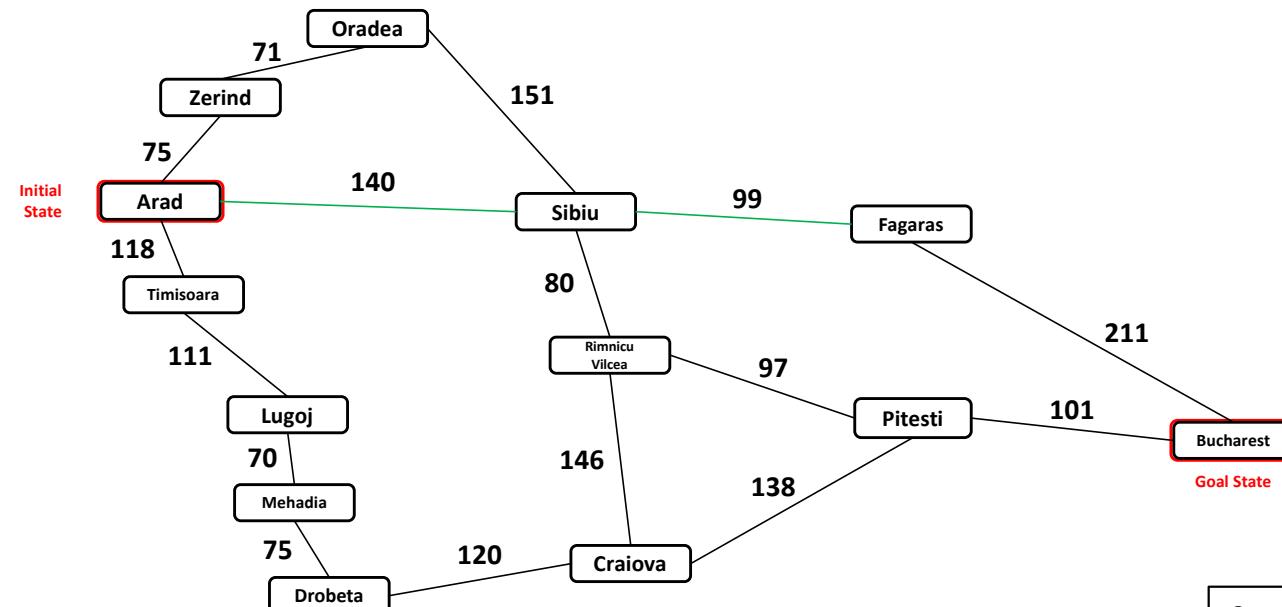
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



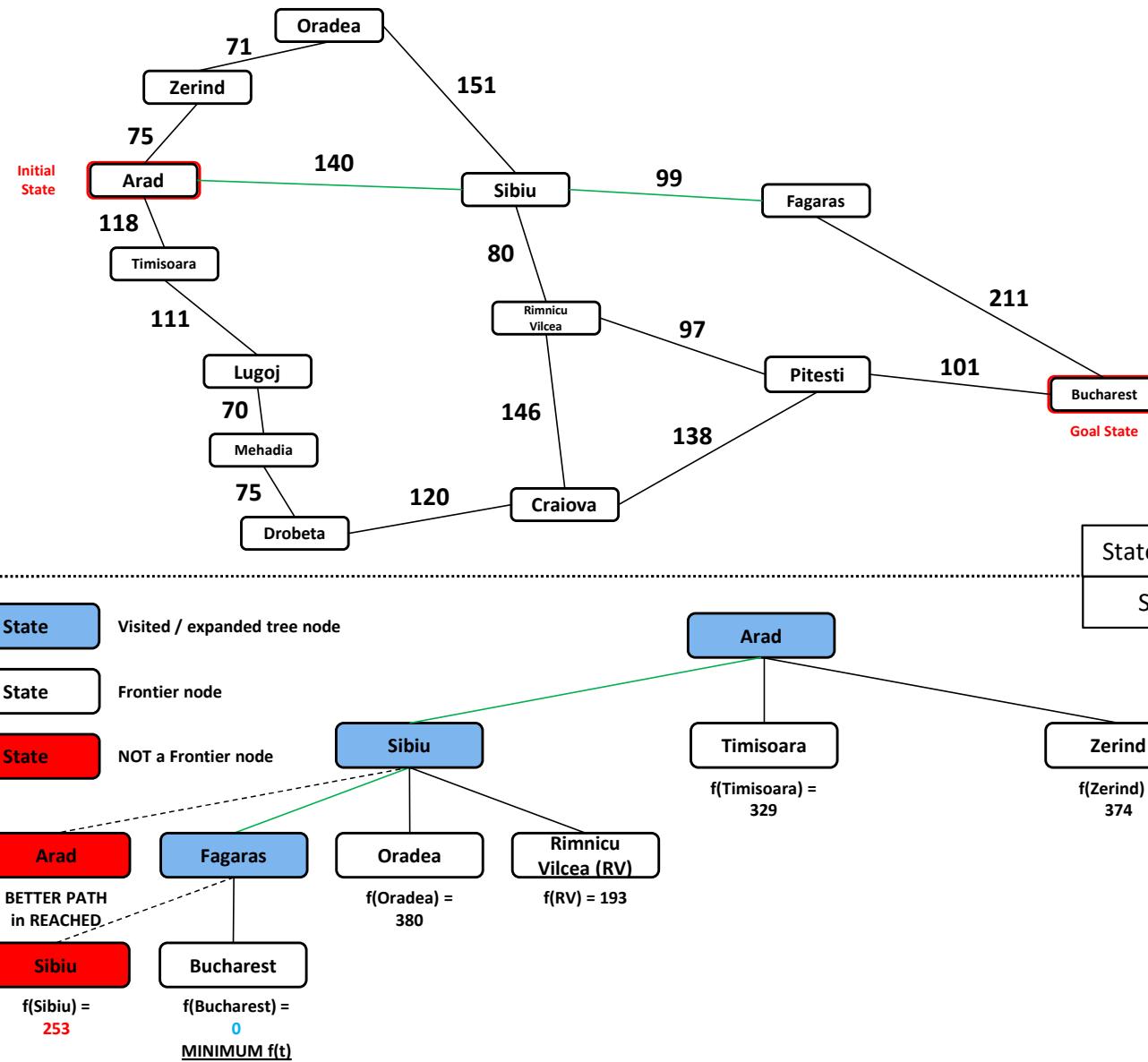
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



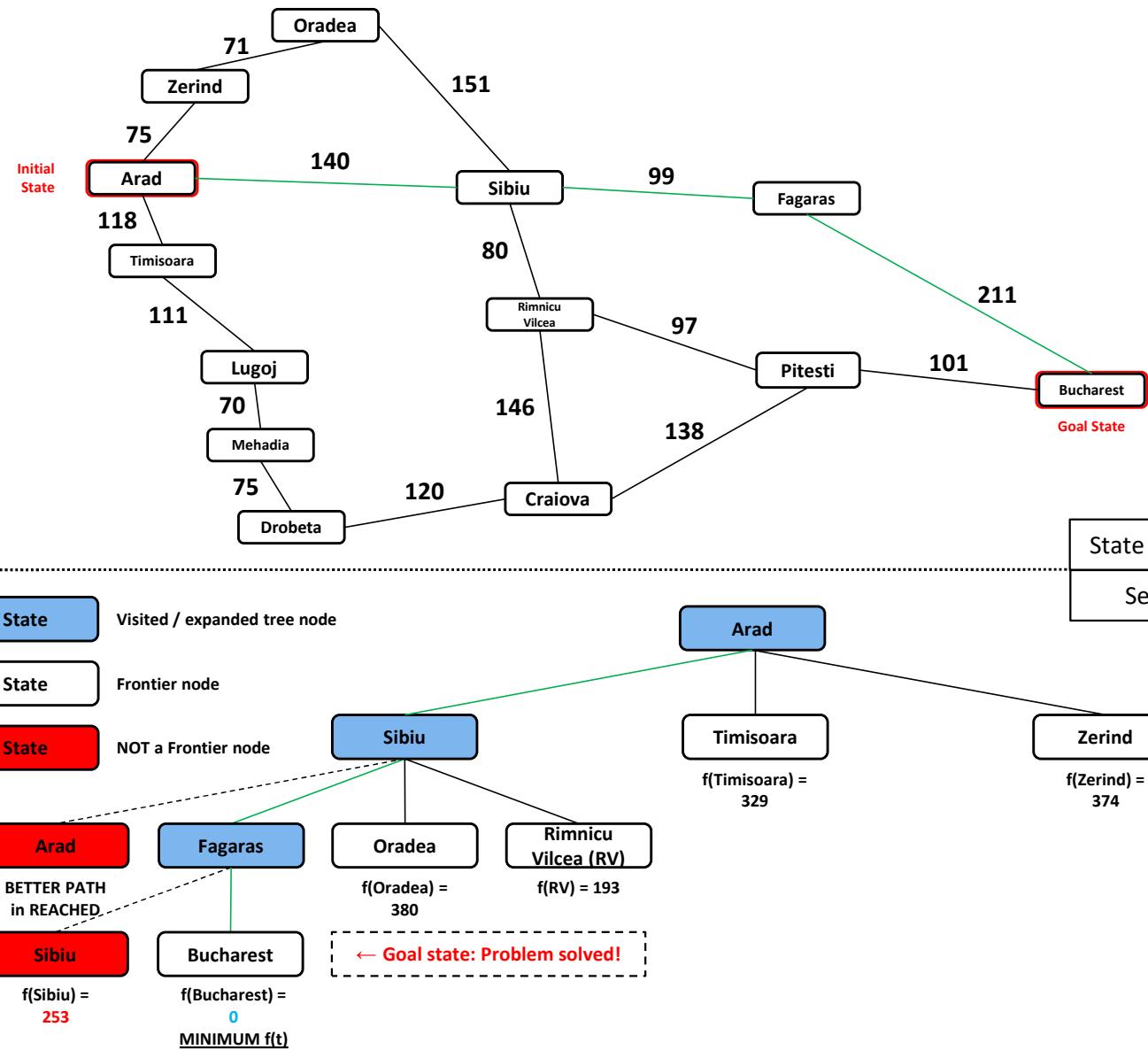
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



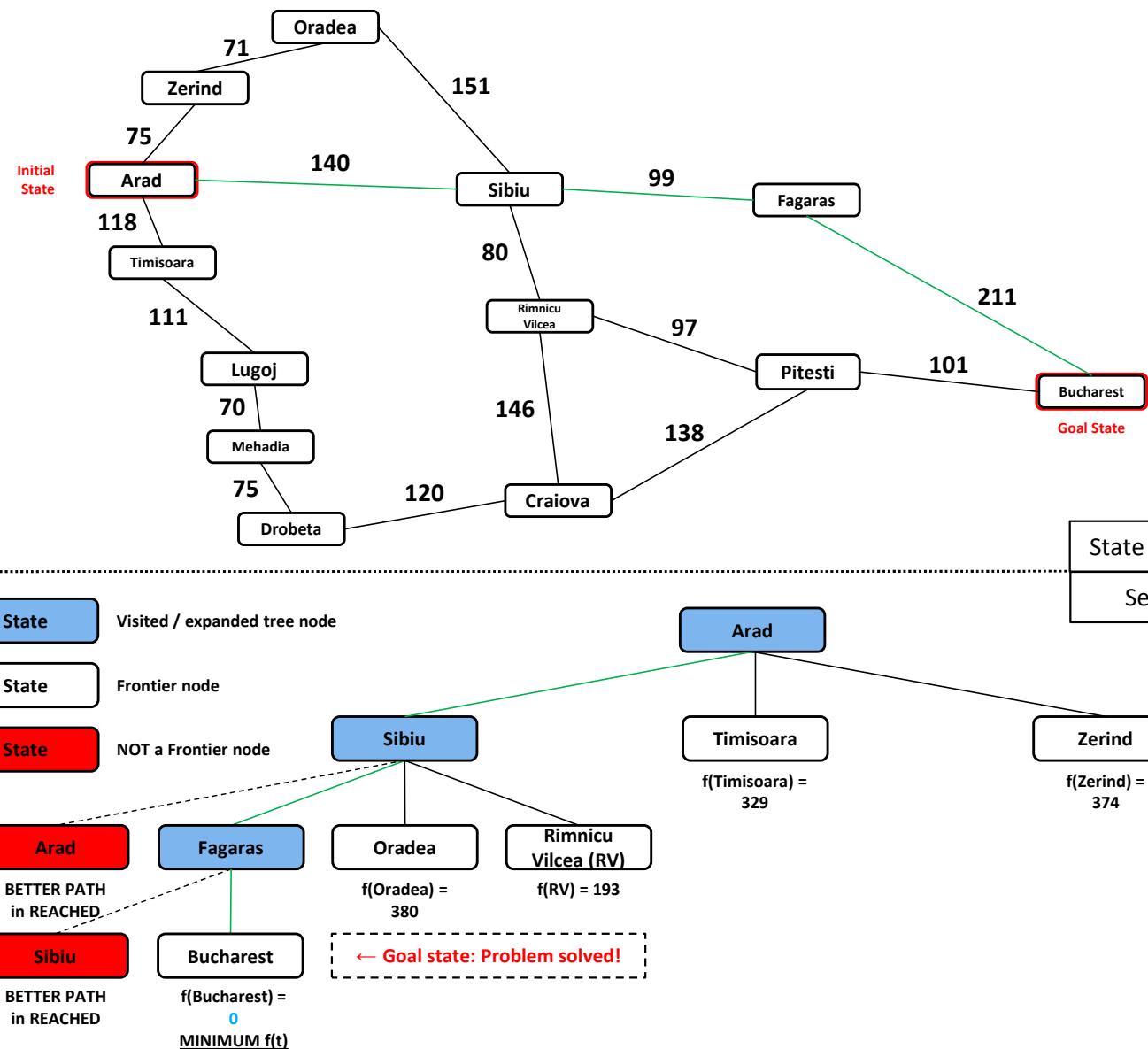
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: Greedy Best First



Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Algorithm: Evaluation Function

Calculate / obtain:

$$f(n) = g(\text{State}_n) + h(\text{State}_n)$$

where:

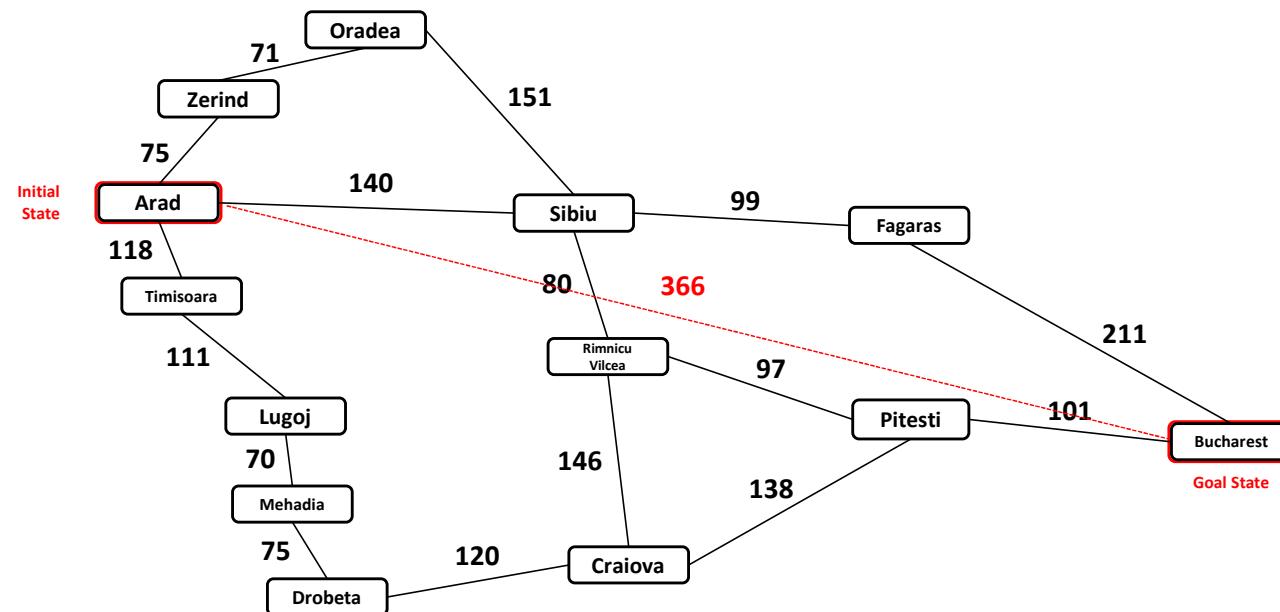
- $g(n)$ - initial node to node n path cost
- $h(n)$ - **estimated cost** of the best path that continues from node n to a goal node

A node n with minimum (maximum) $f(n)$ should be chosen for expansion

Romanian Roadtrip: Heuristics $h(n)$

For this particular problem the heuristic function $h(n)$ is defined by a **straight-line (Euclidean) distance** between two states (cities).

“As the crows flies” in other words.



Best-First Search: A* Pseudocode

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.Is-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

$f(n) = g(\text{State}_n) + h(\text{State}_n)$

```
function EXPAND(problem, node) yields nodes
    s  $\leftarrow$  node.STATE
    for each action in problem.ACTIONS(s) do
        s'  $\leftarrow$  problem.RESULT(s, action)
        cost  $\leftarrow$  node.PATH-COST + problem.ACTION-COST(s, action, s')
        yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Best-First Search: A* Pseudocode

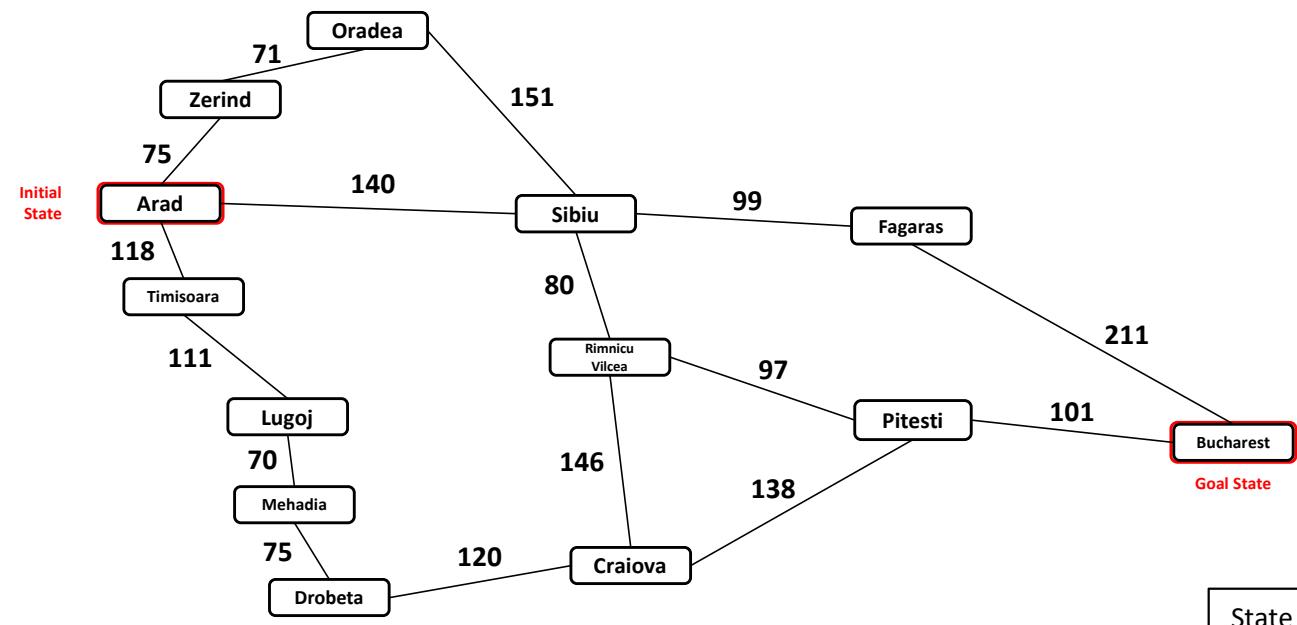
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.Is-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

$f(n) = g(\text{State}_n) + h(\text{State}_n)$

Best-First Search is really a class of search algorithms that:

- Use the evaluation function $f(n)$ to pick next action
- Keep track of visited states
- Keep track of frontier states
- Evaluation function $f(n)$ choice controls their behavior

Romanian Roadtrip: A* Search



State Space Graph
Search Tree

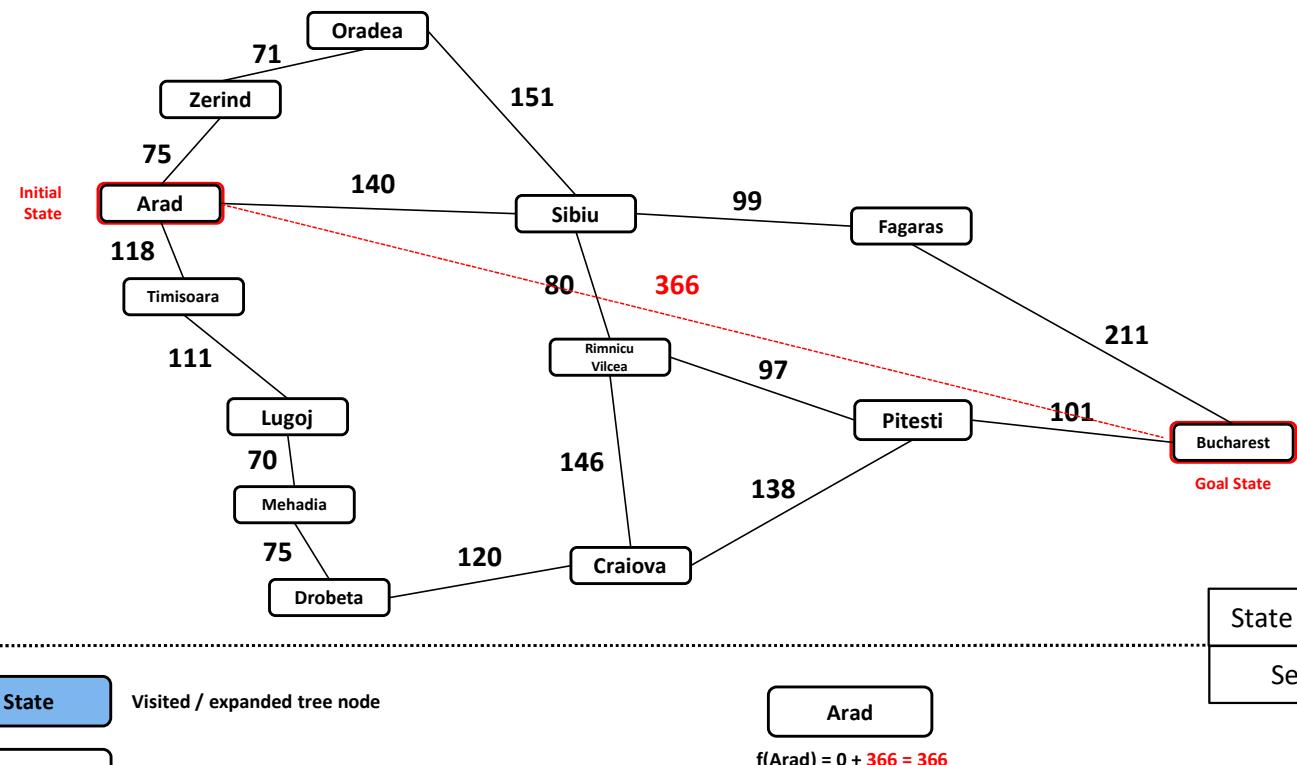
$$f(\text{Arad}) = g(\text{Arad}) + h(\text{Arad})$$

Arad

Straight-line distance to Bucharest ($h(\text{State})$):

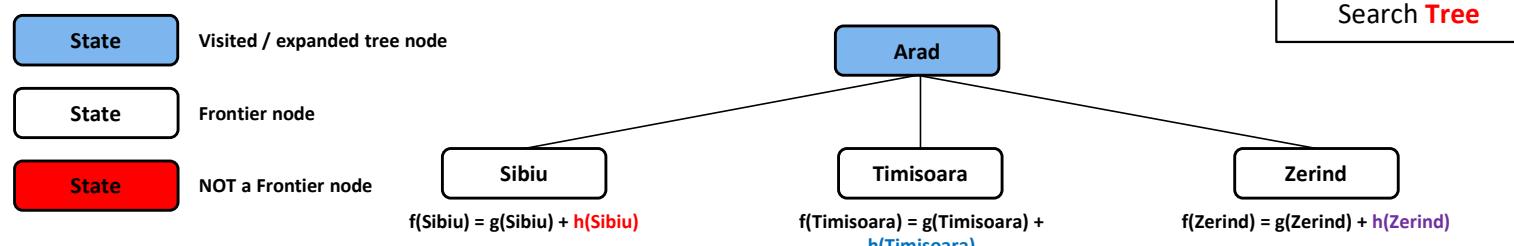
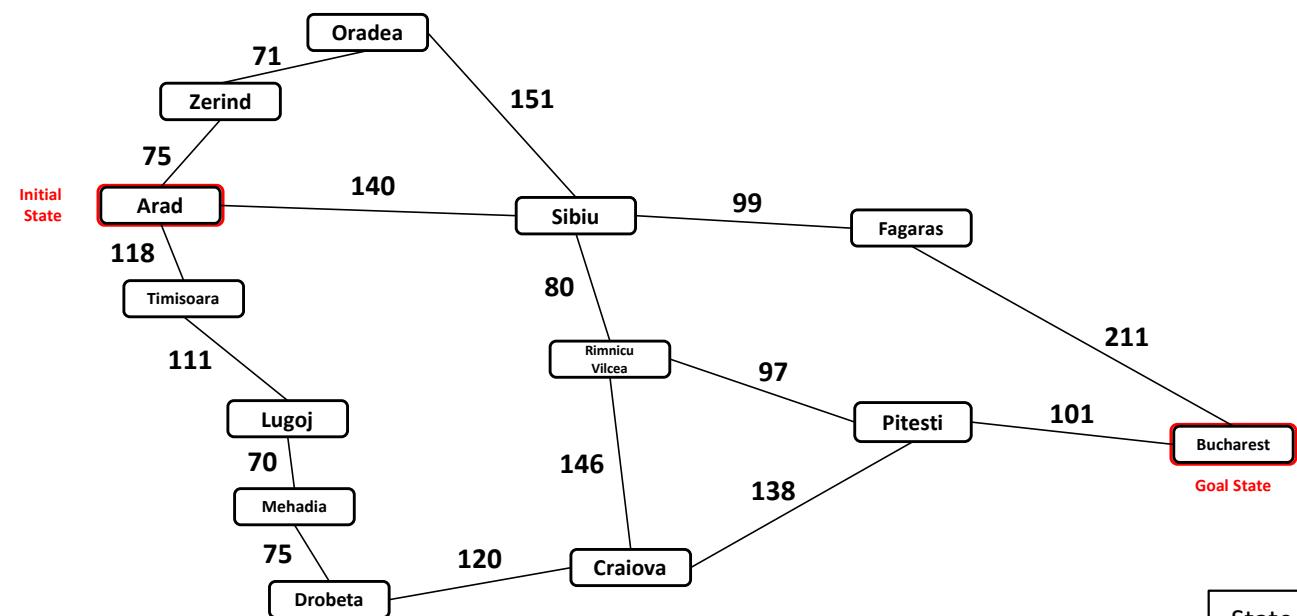
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



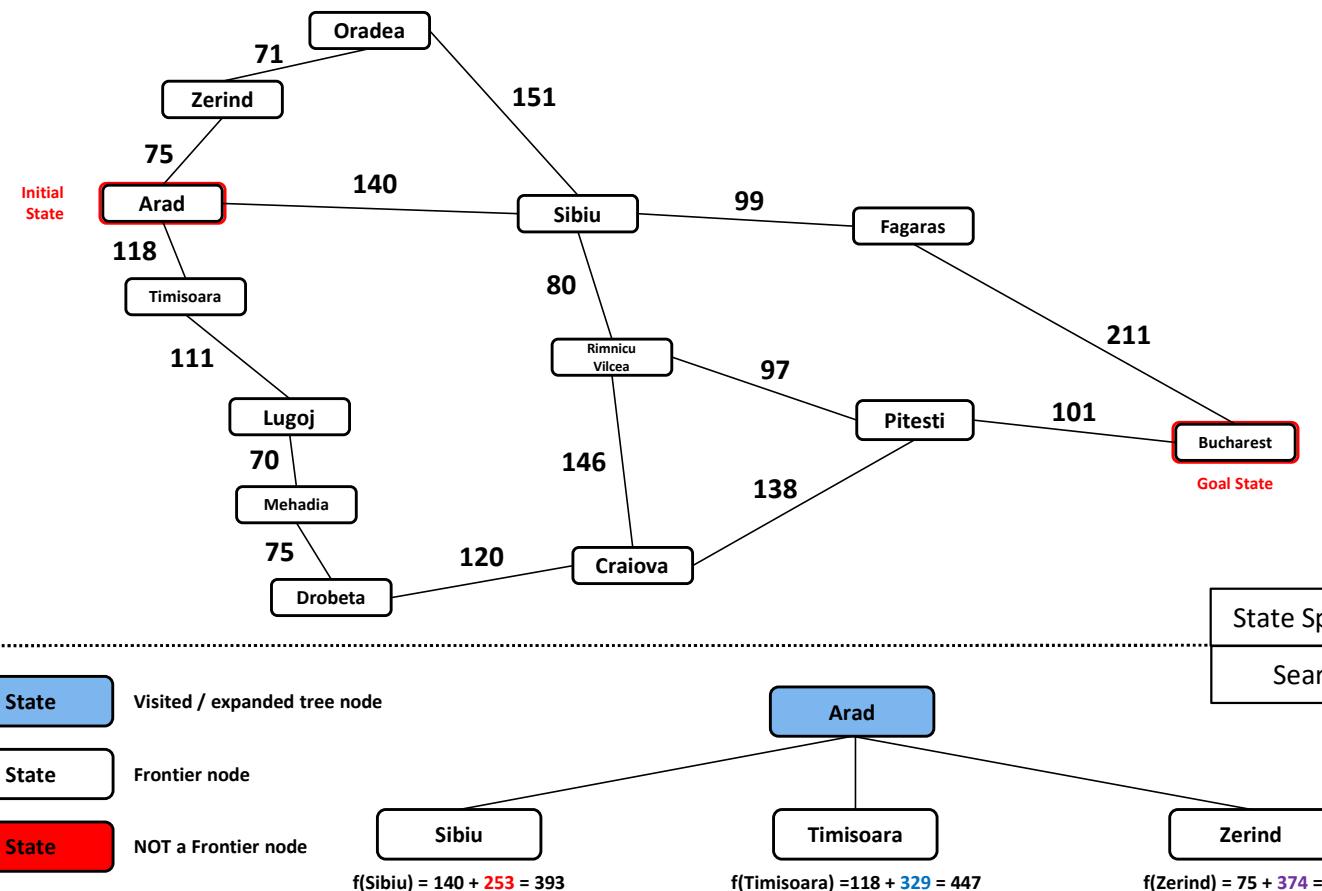
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



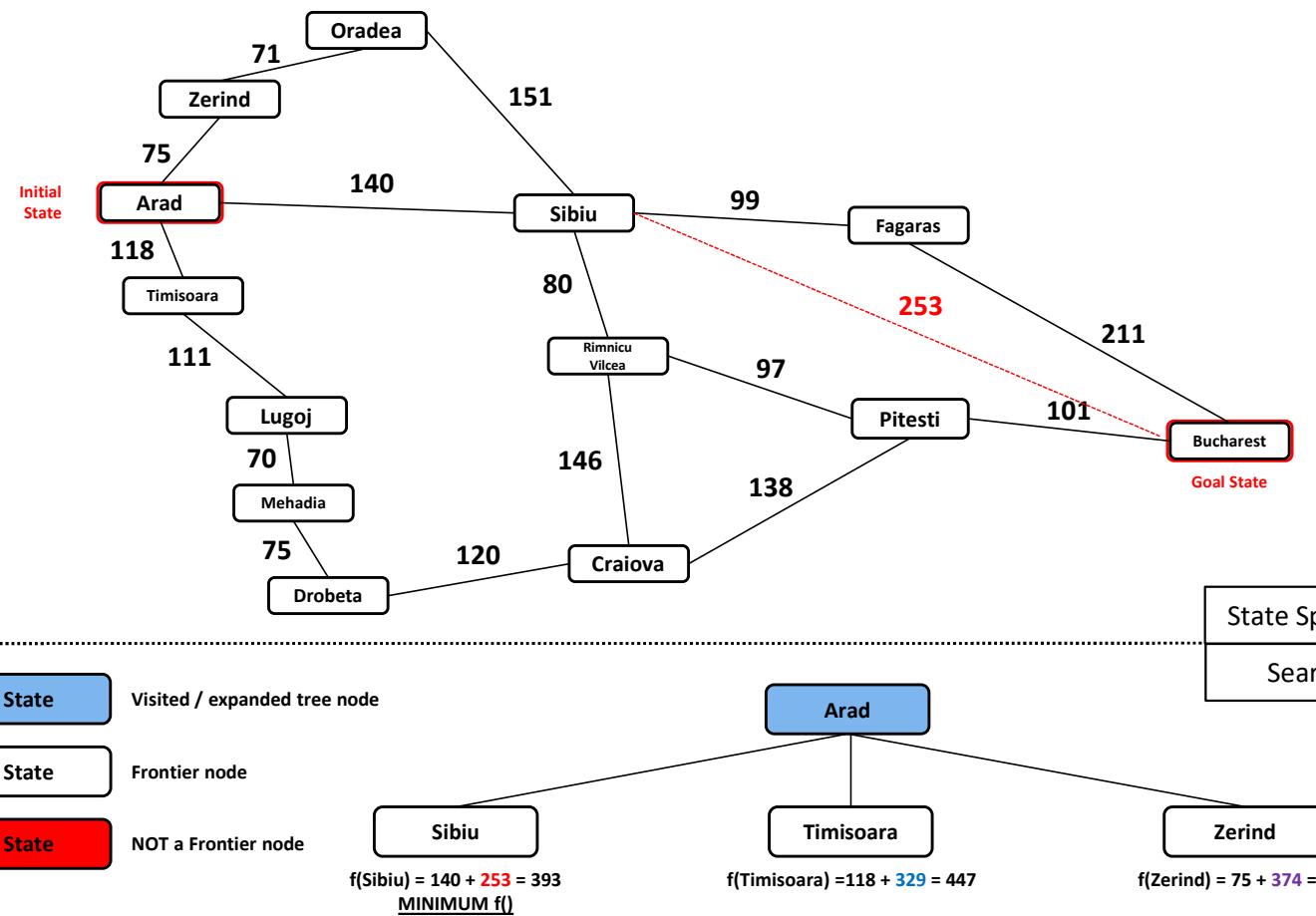
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



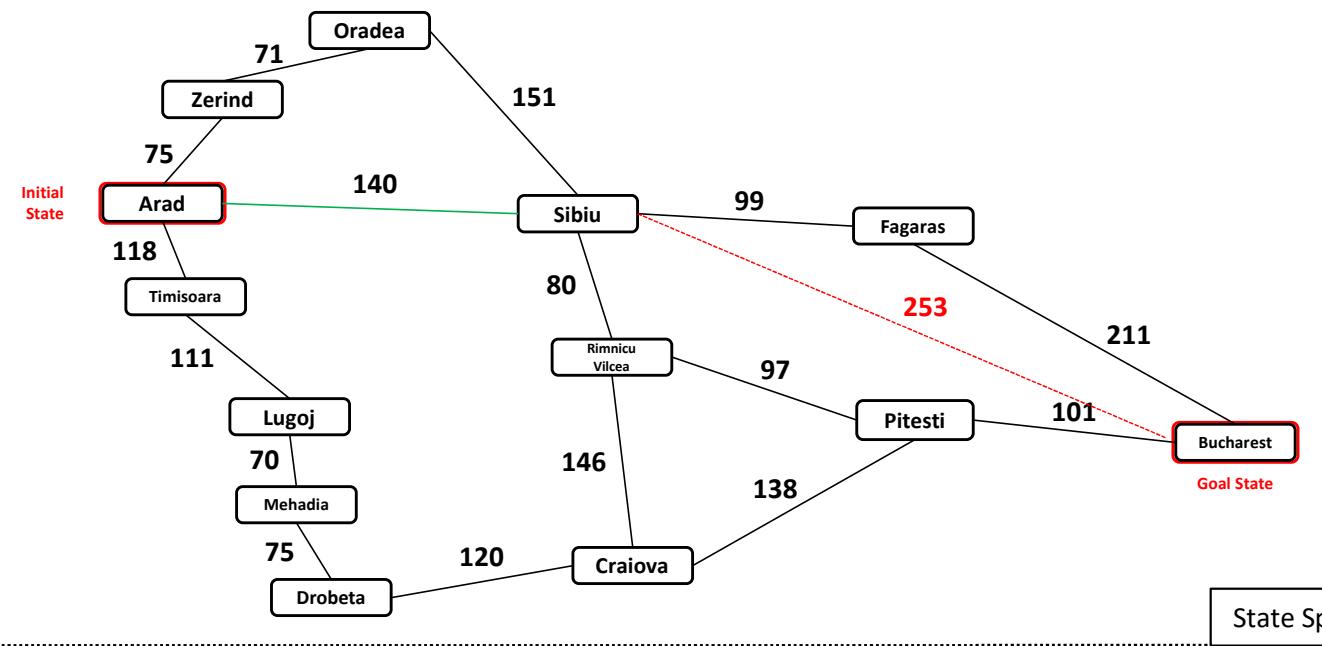
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



- | | |
|-------|------------------------------|
| State | Visited / expanded tree node |
| State | Frontier node |
| State | NOT a Frontier node |

Sibiu
 $f(\text{Sibiu}) = 140 + \underline{\text{253}} = 393$
MINIMUM f()

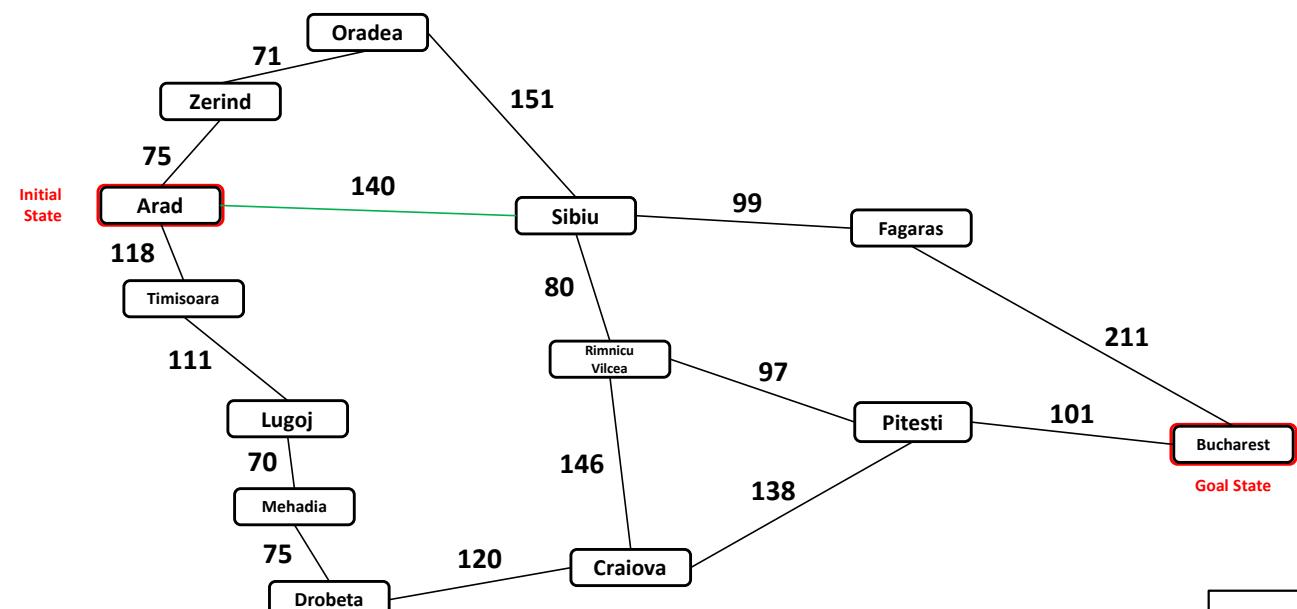
Arad
 $f(\text{Timisoara}) = 118 + \underline{\text{329}} = 447$

Zerind
 $f(\text{Zerind}) = 75 + \underline{\text{374}} = 449$

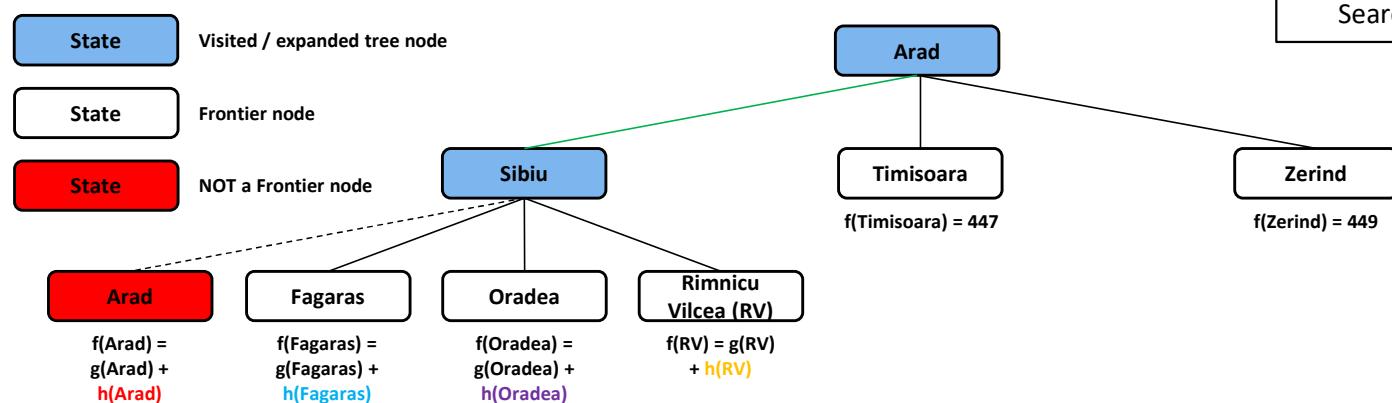
State Space **Graph**
Search Tree

Straight-line distance to Bucharest (h(State)):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

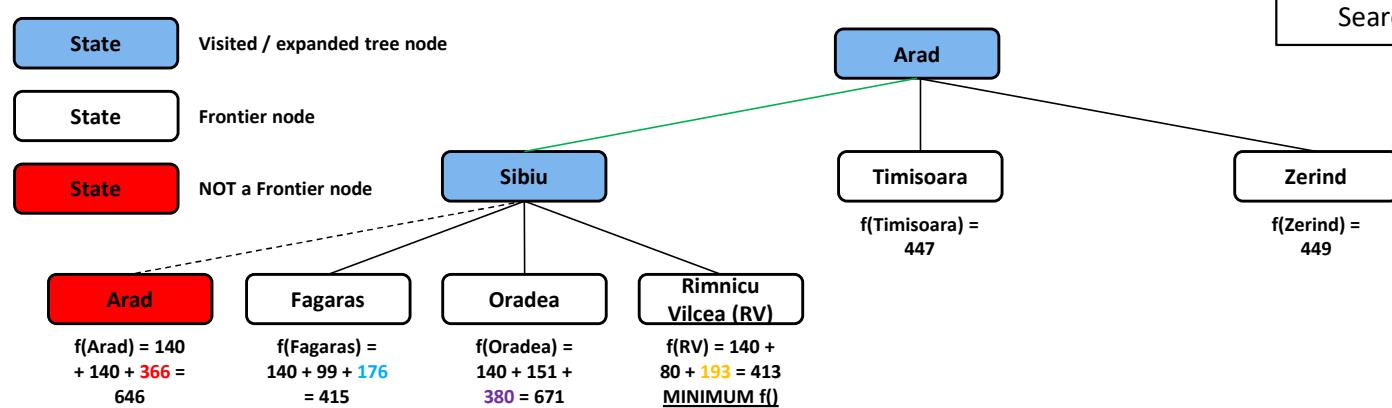
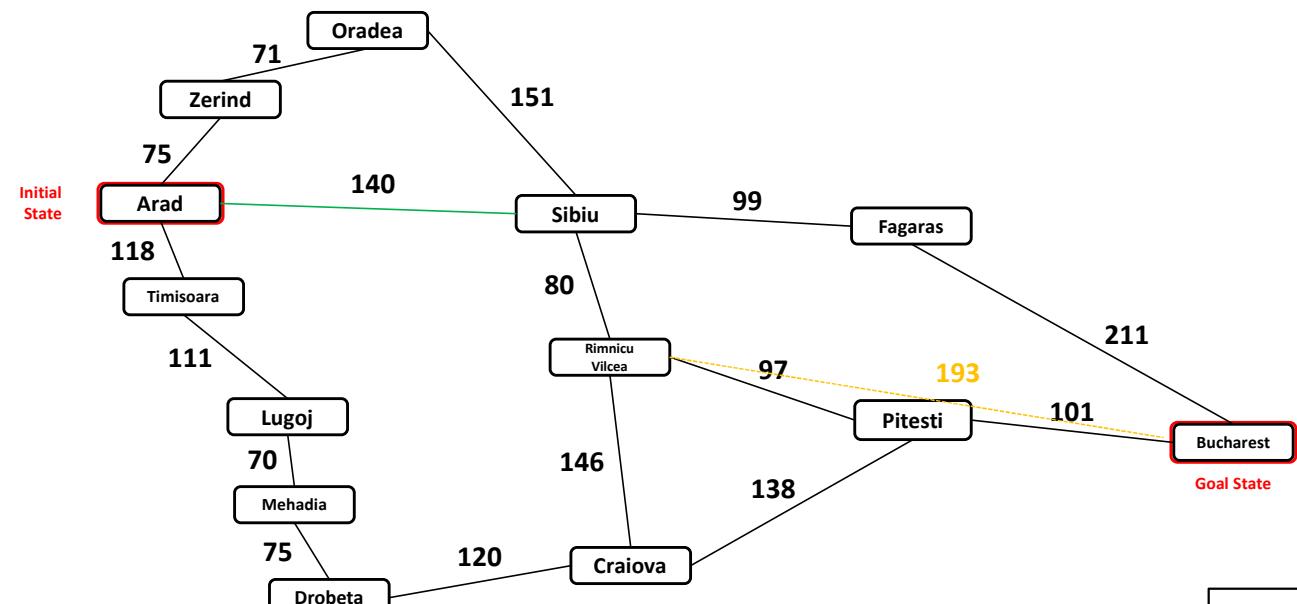
Romanian Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

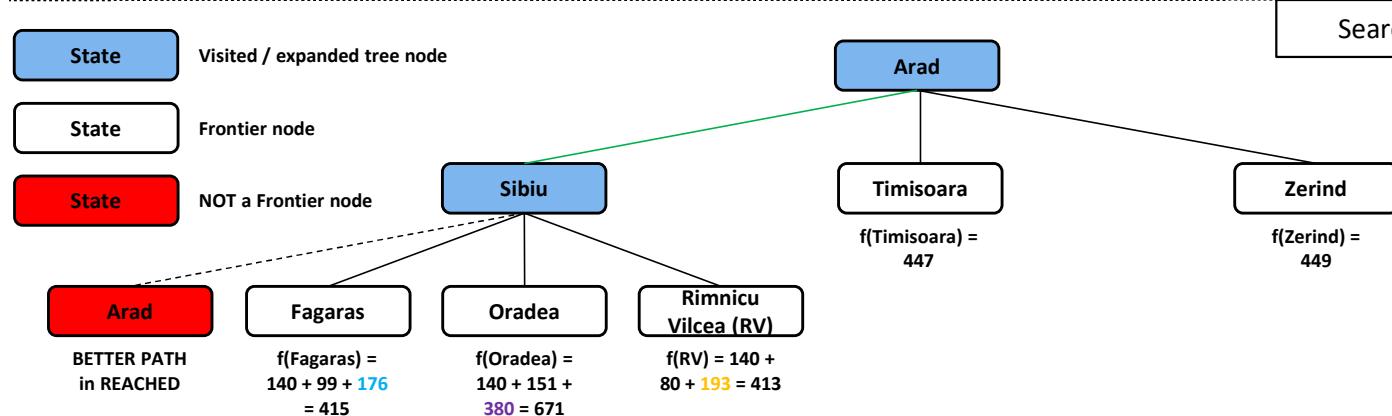
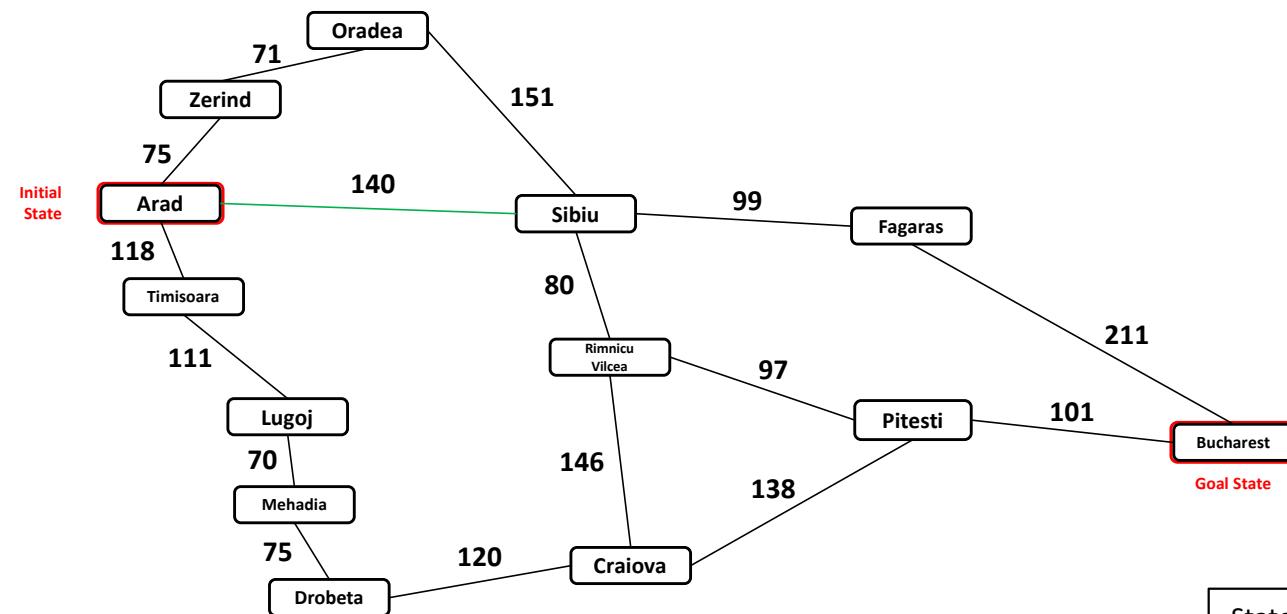


Romanian Roadtrip: A* Search



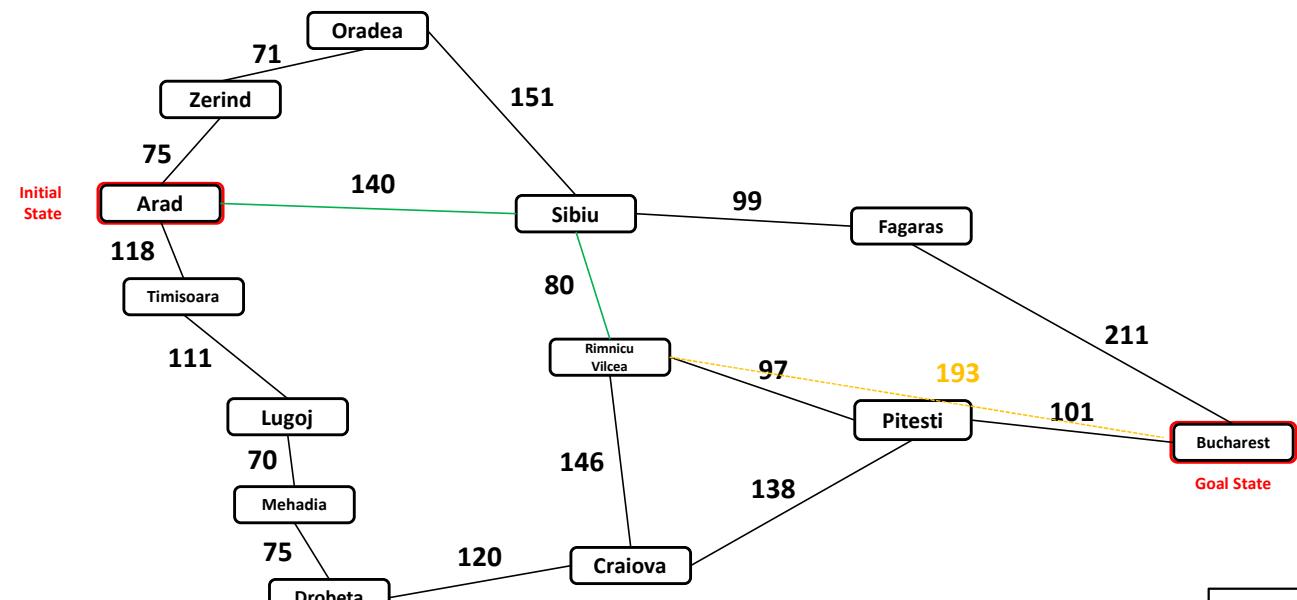
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search

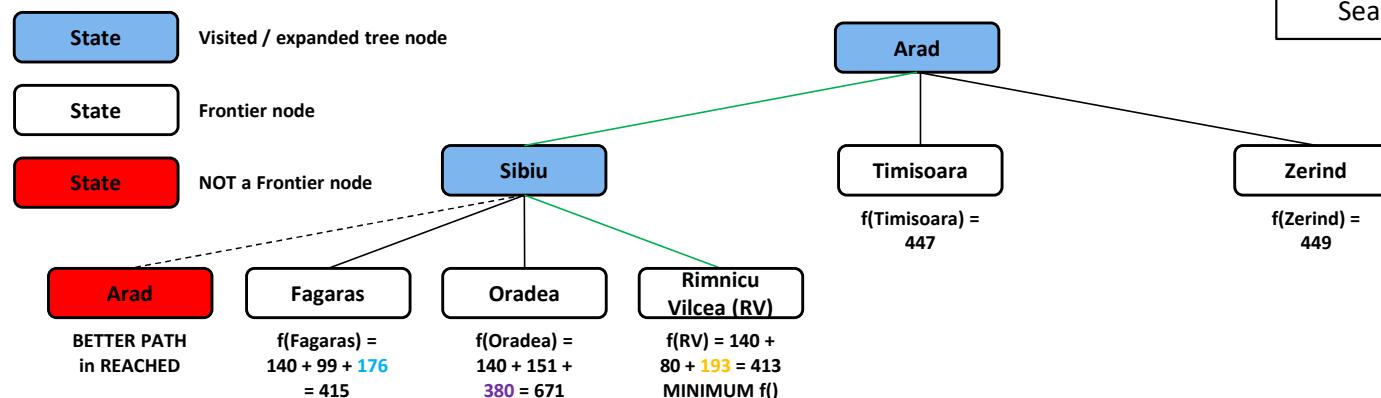


Straight-line distance to Bucharest (h(State)):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search

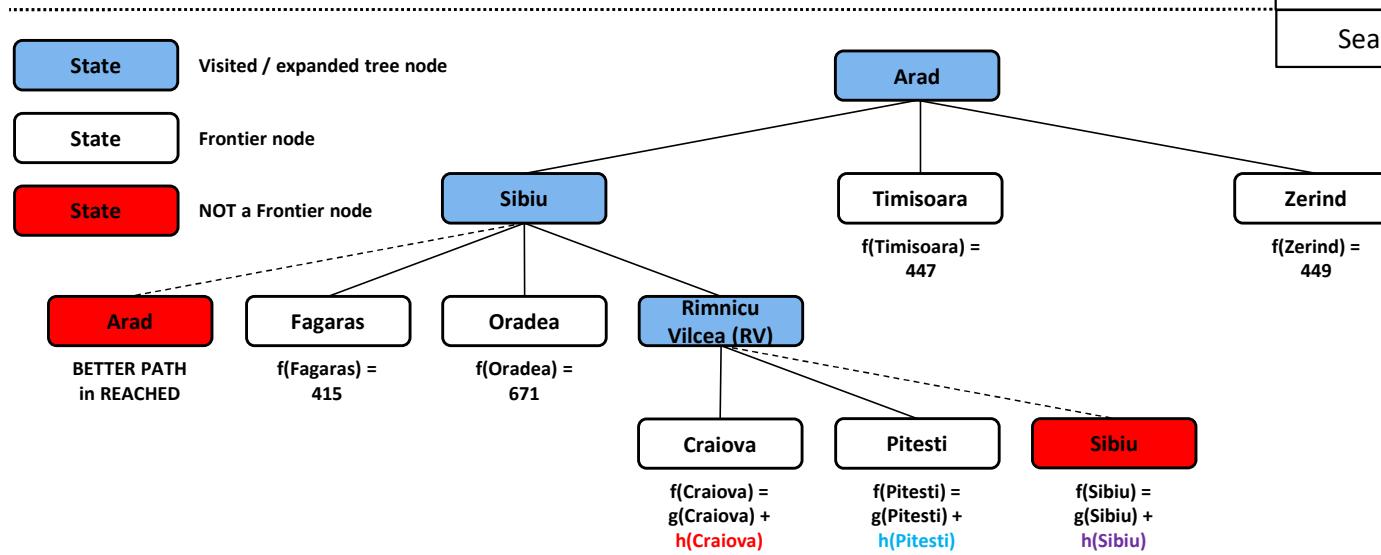
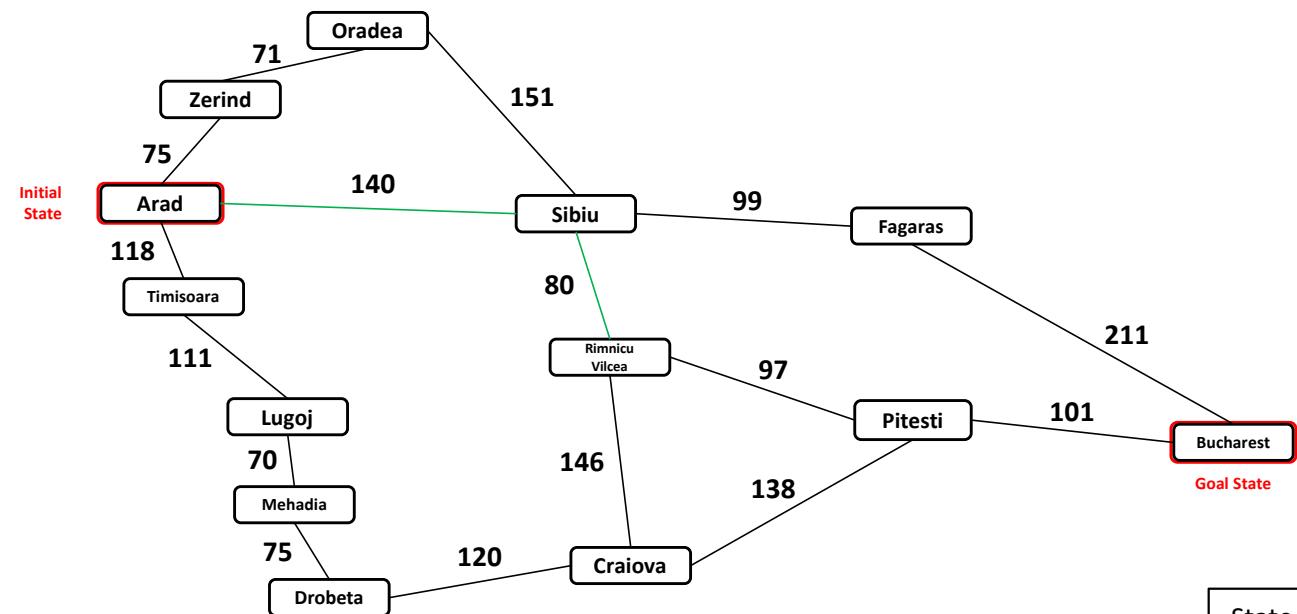


State Space Graph
Search Tree



Straight-line distance to Bucharest (h(State)):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

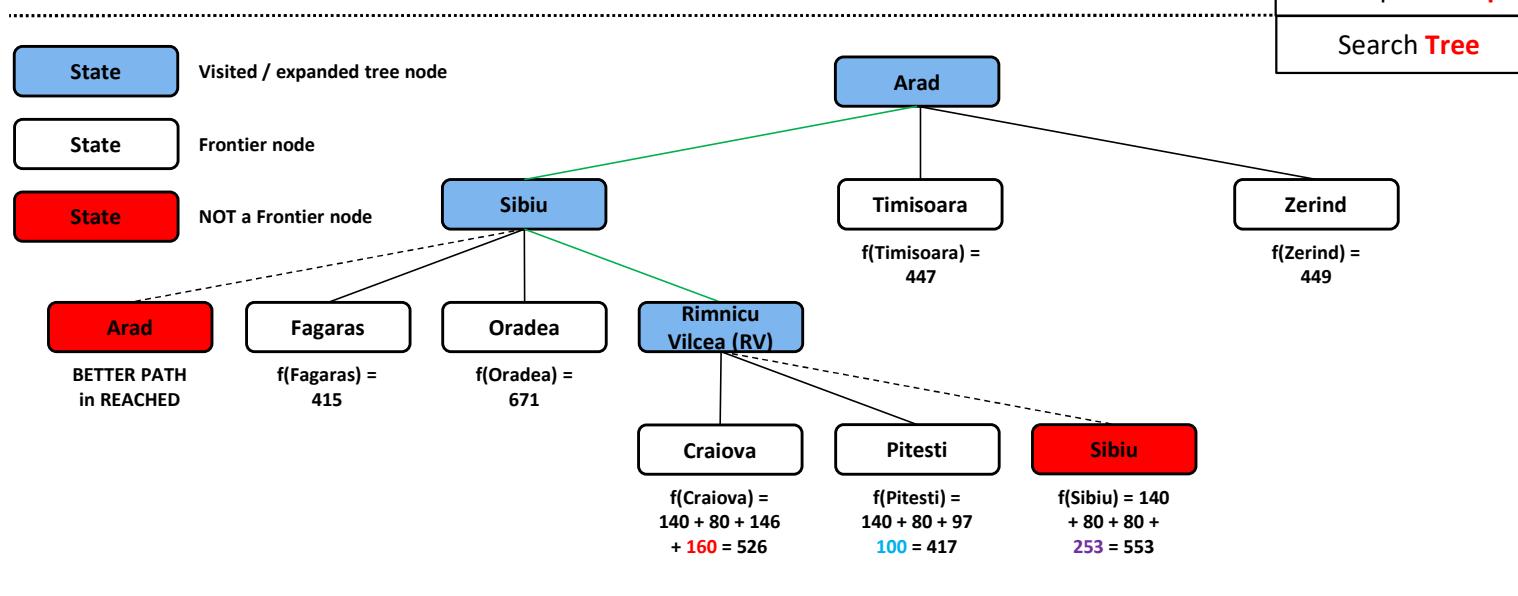
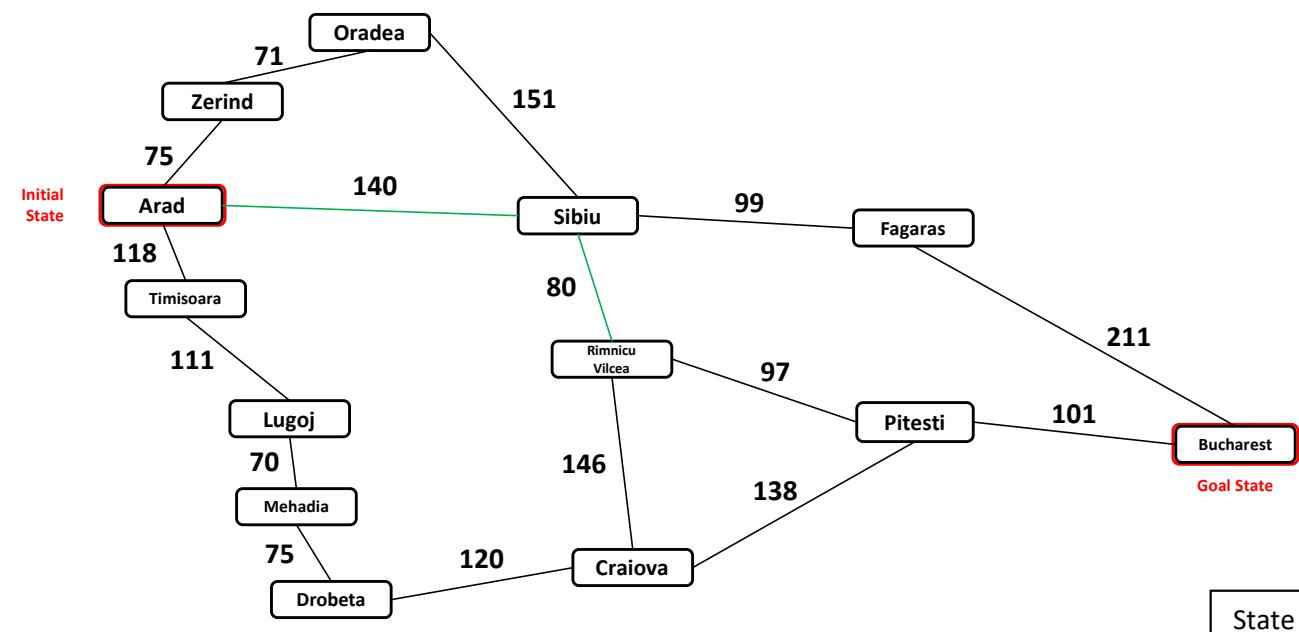
Romanian Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

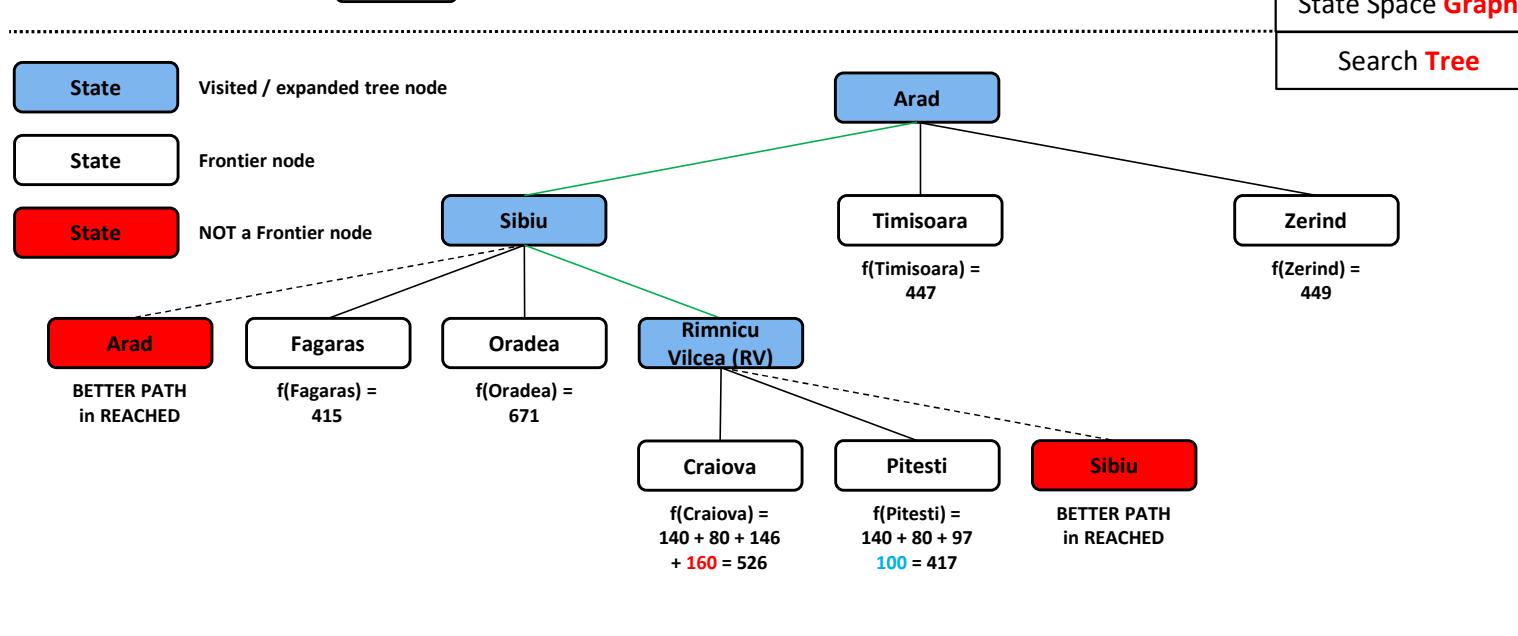
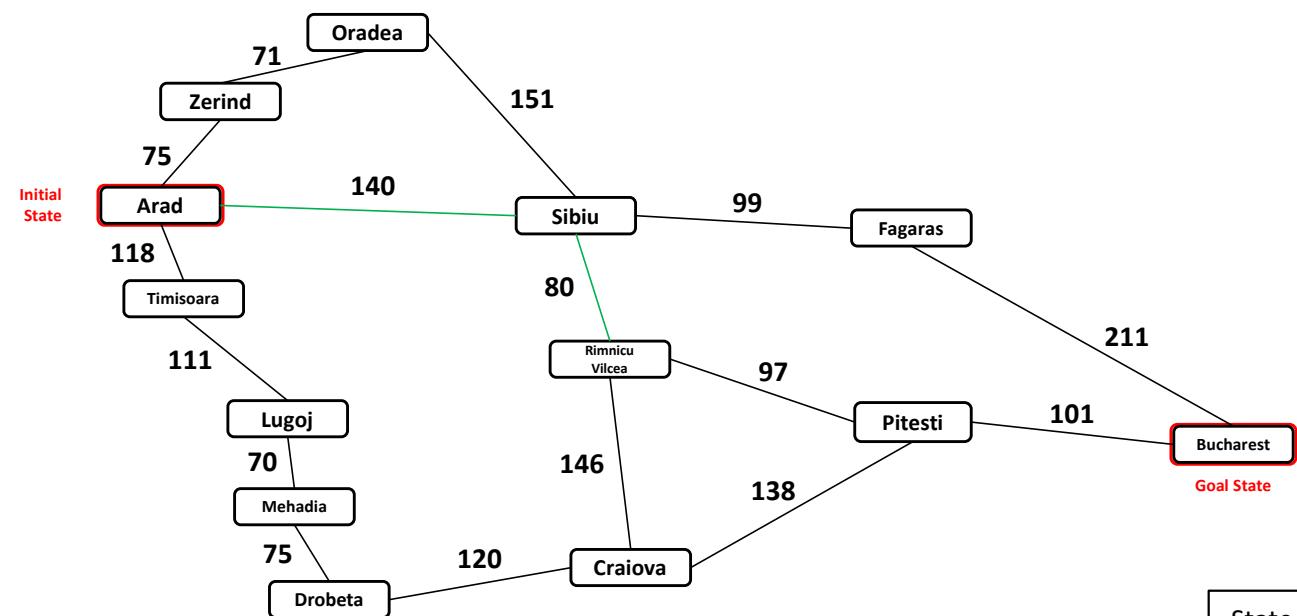
Romanian Roadtrip: A* Search



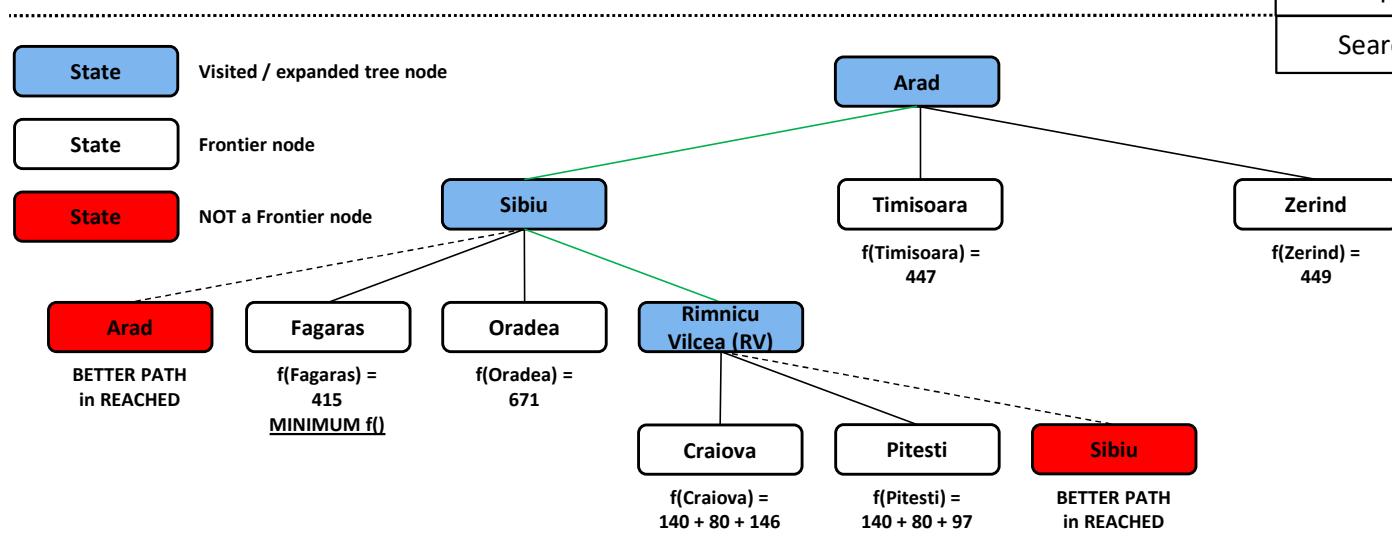
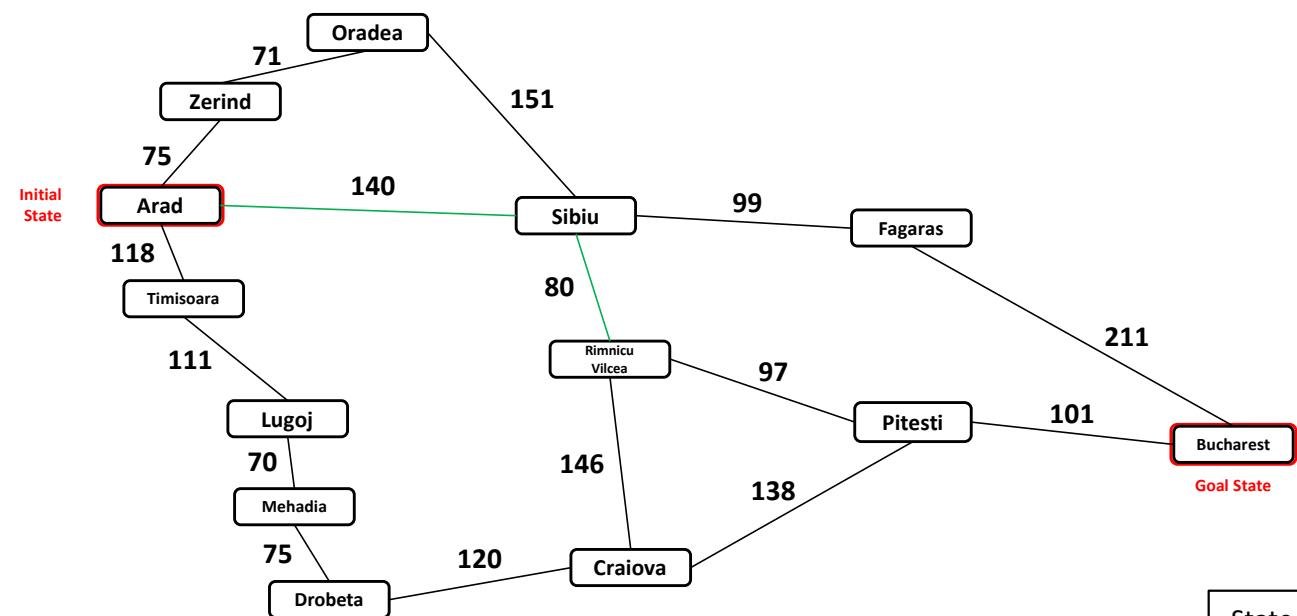
Straight-line distance to Bucharest (h(State)):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

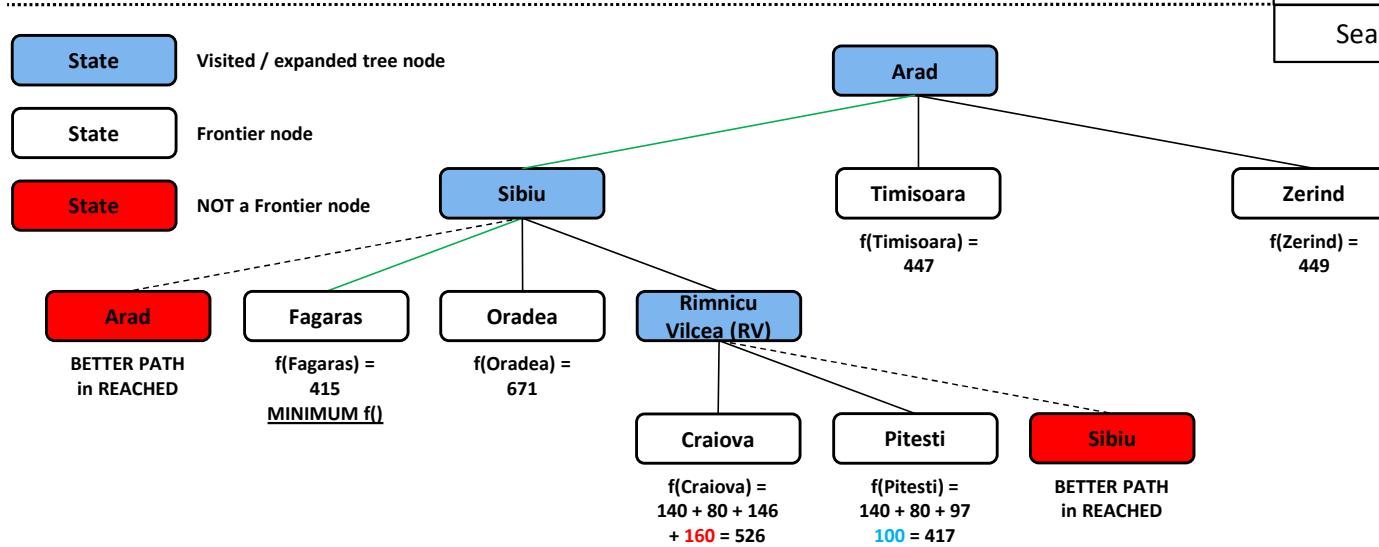
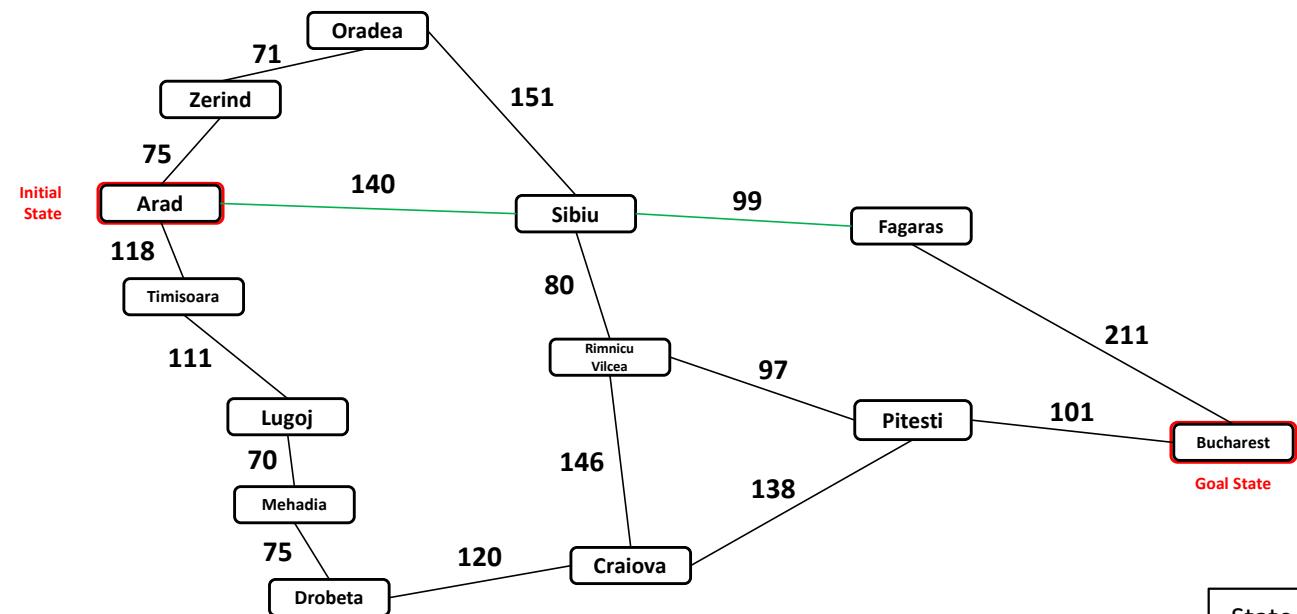
Romanian Roadtrip: A* Search



Romanian Roadtrip: A* Search



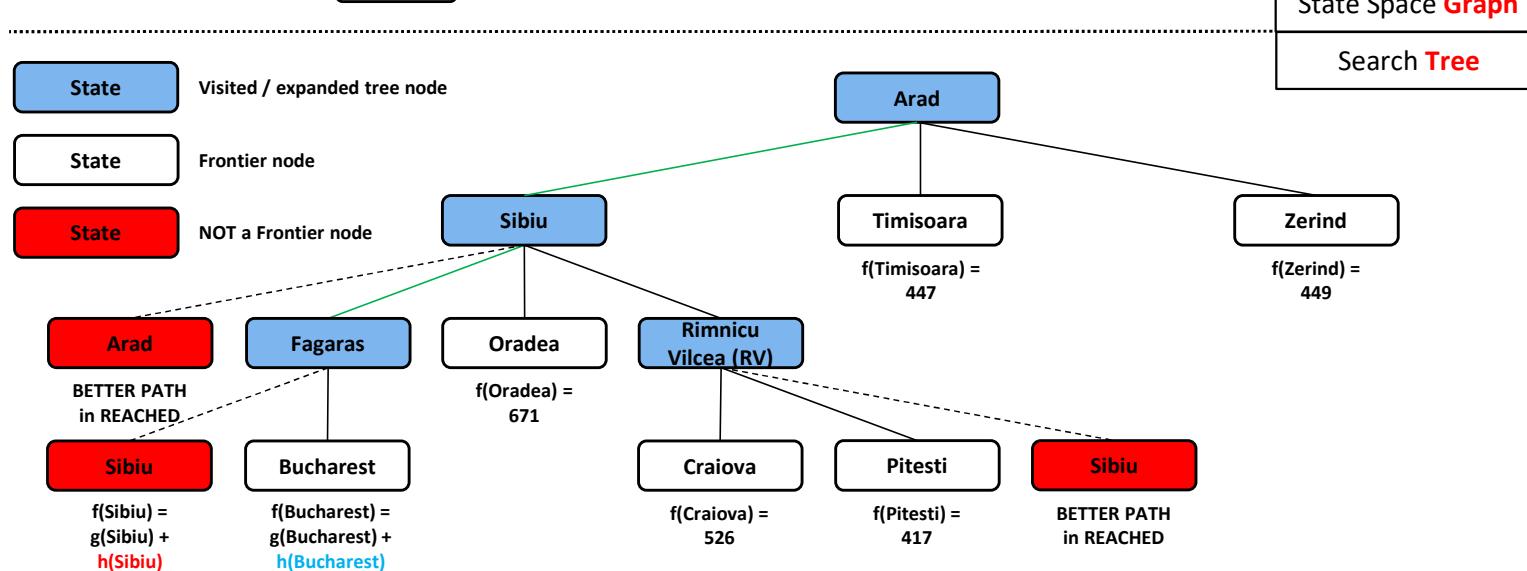
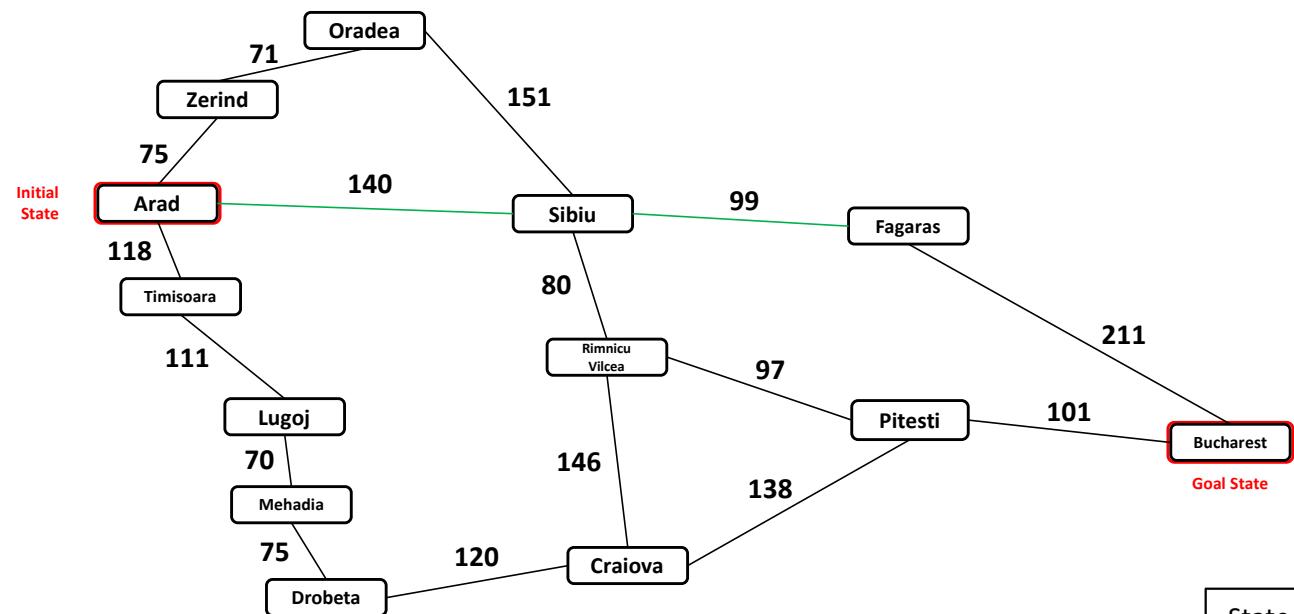
Romanian Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

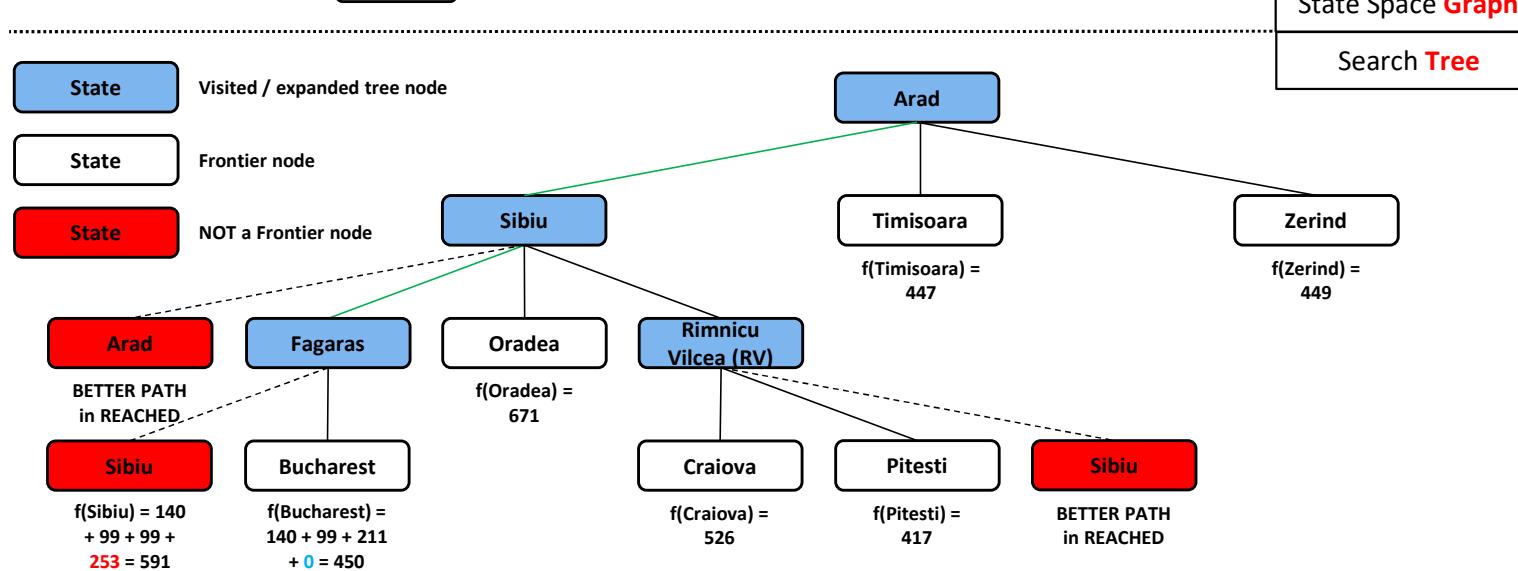
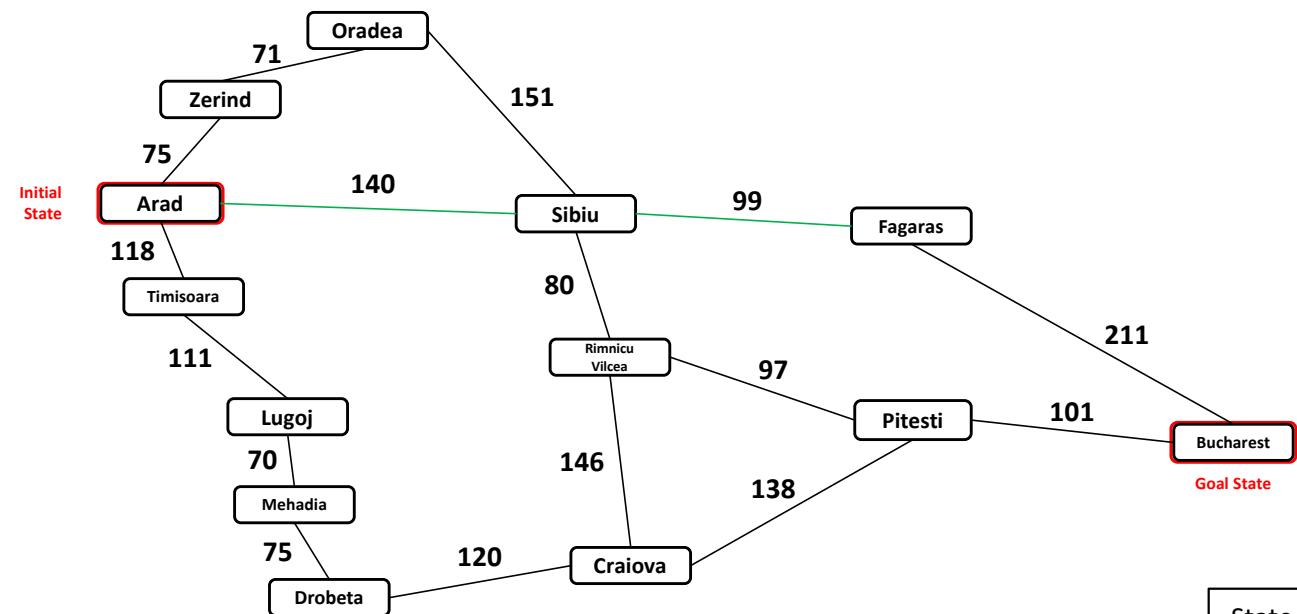
Romanian Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

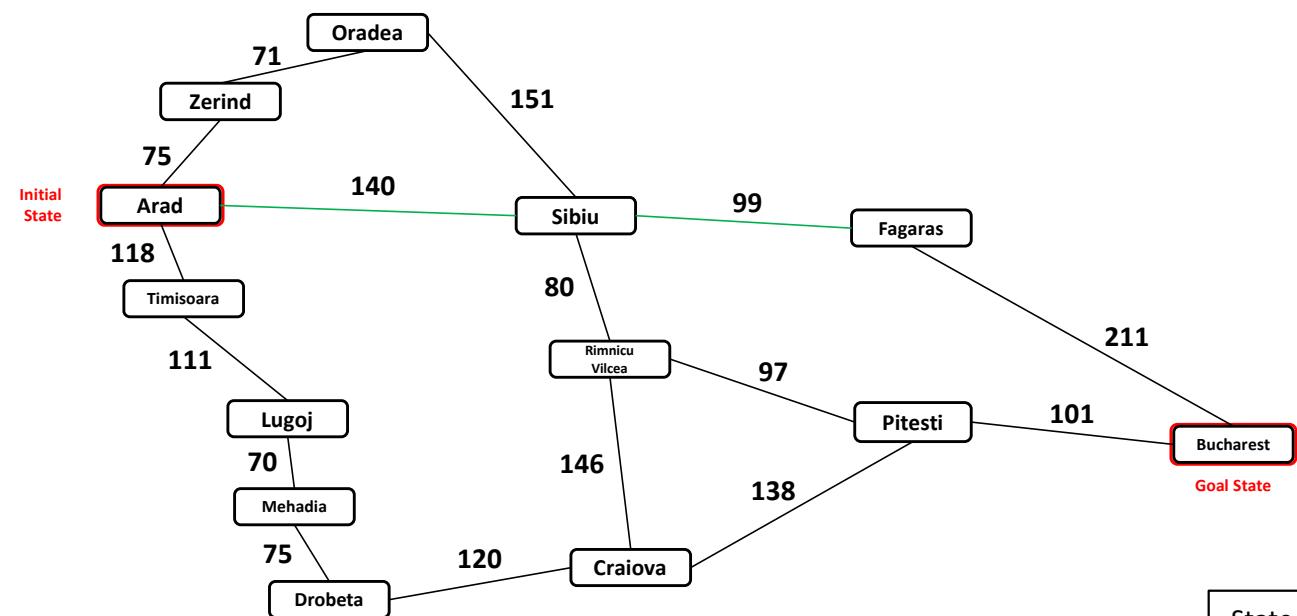
Romanian Roadtrip: A* Search



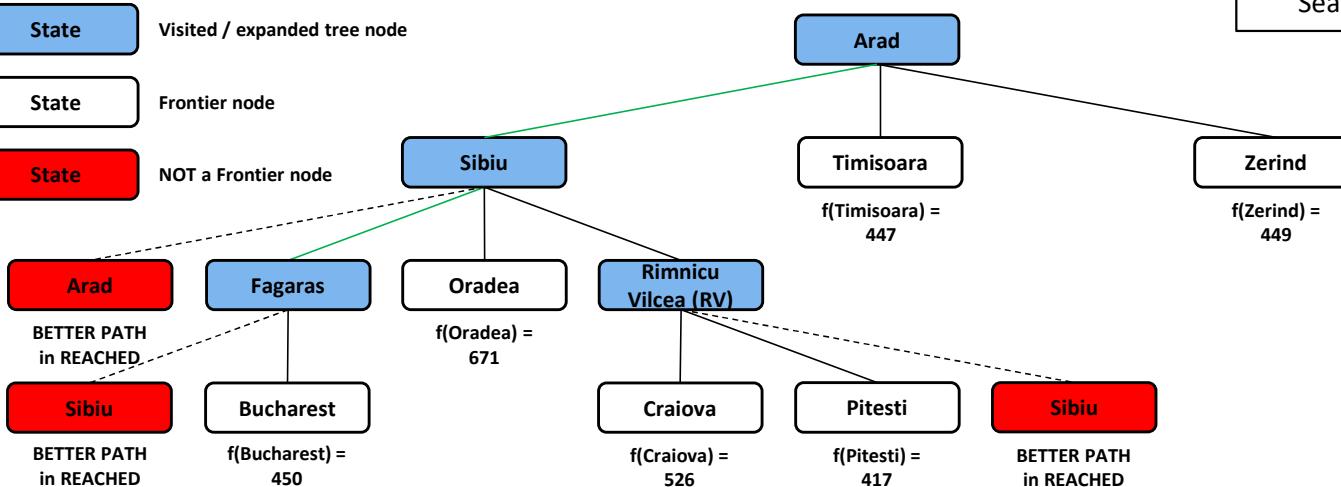
Straight-line distance to Bucharest ($h(\text{State})$):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



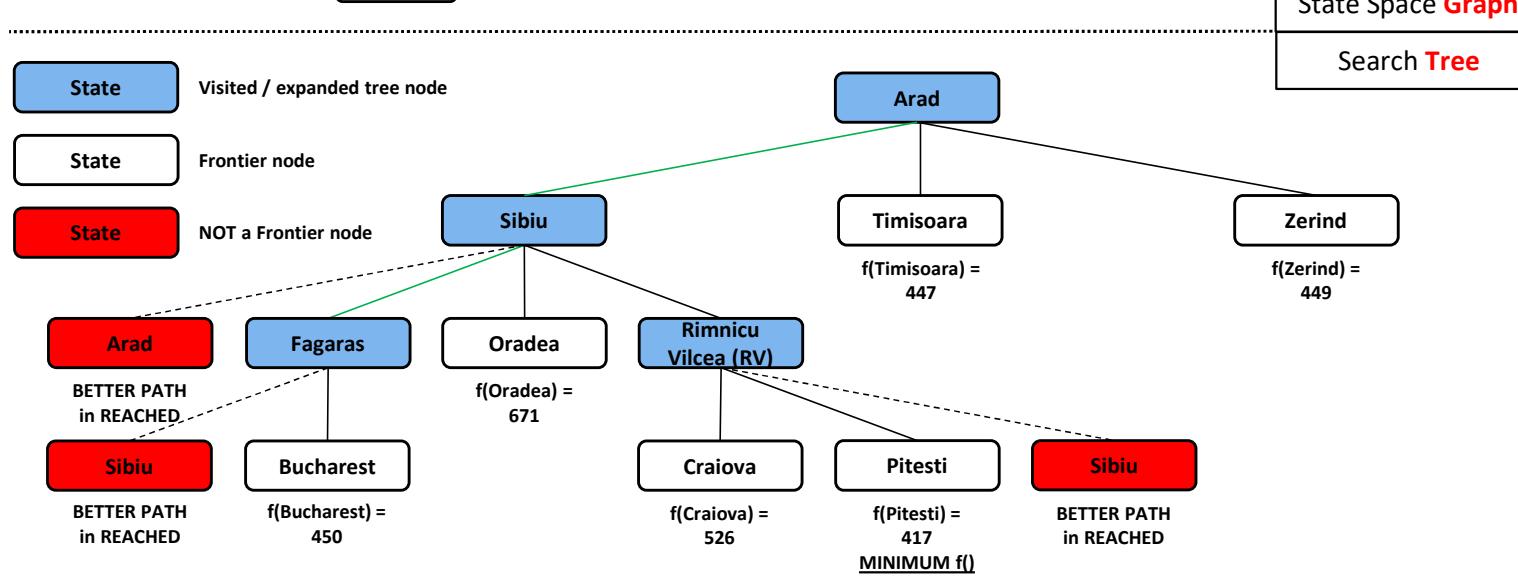
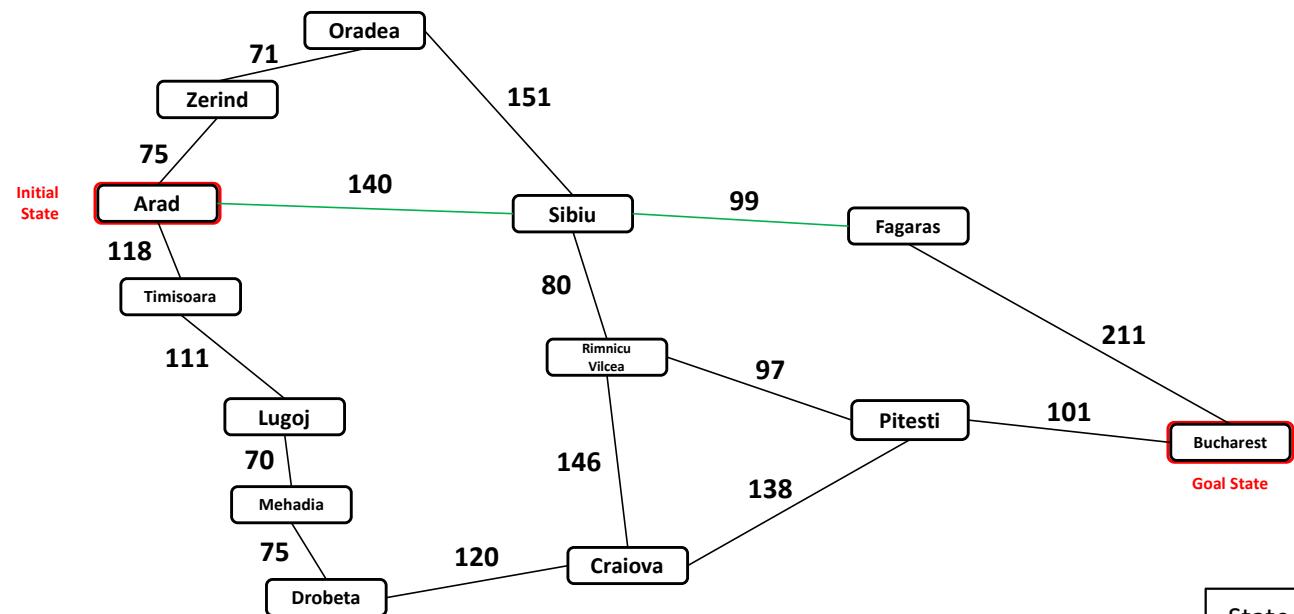
State Space Graph
Search Tree



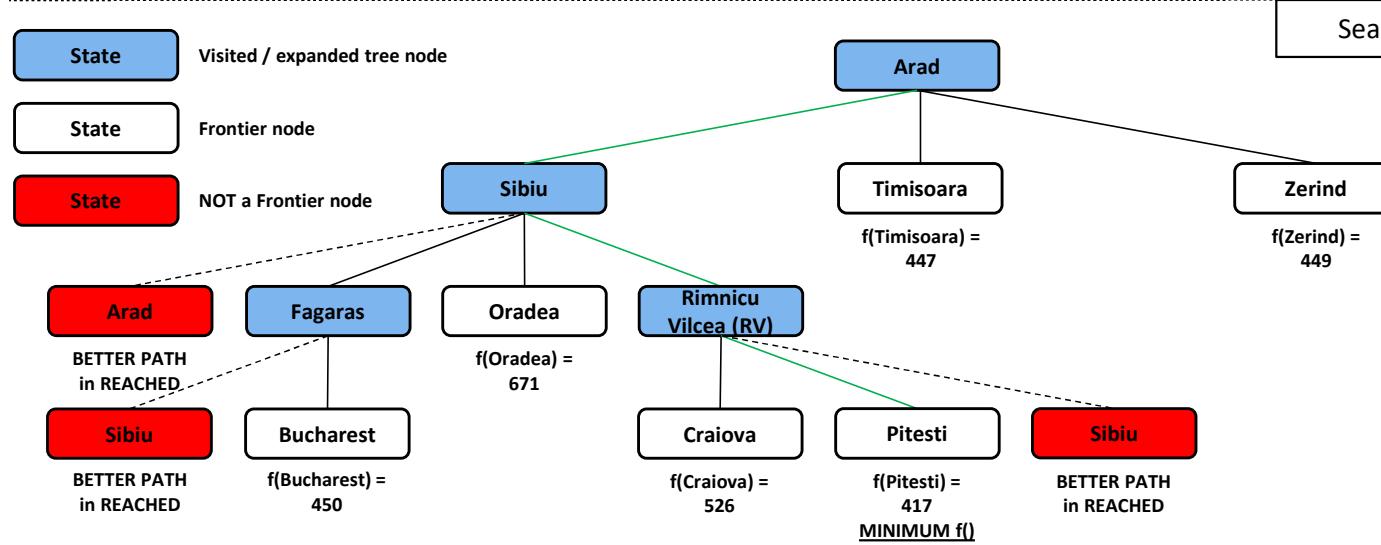
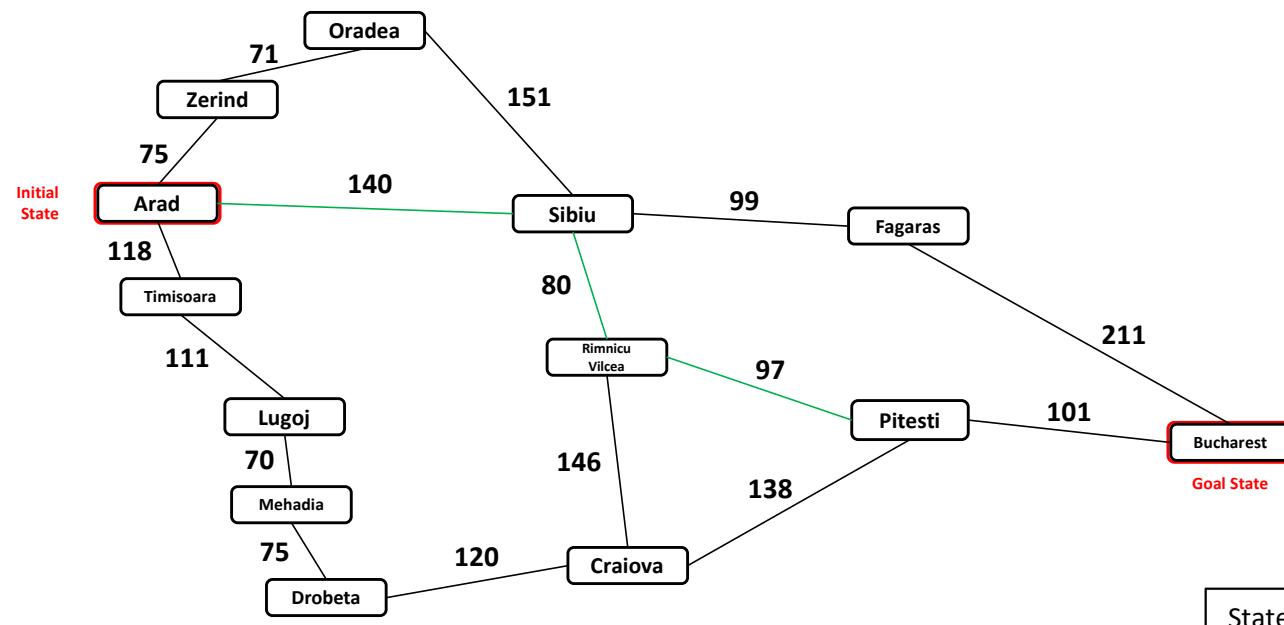
Straight-line distance to Bucharest ($h(\text{State})$):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

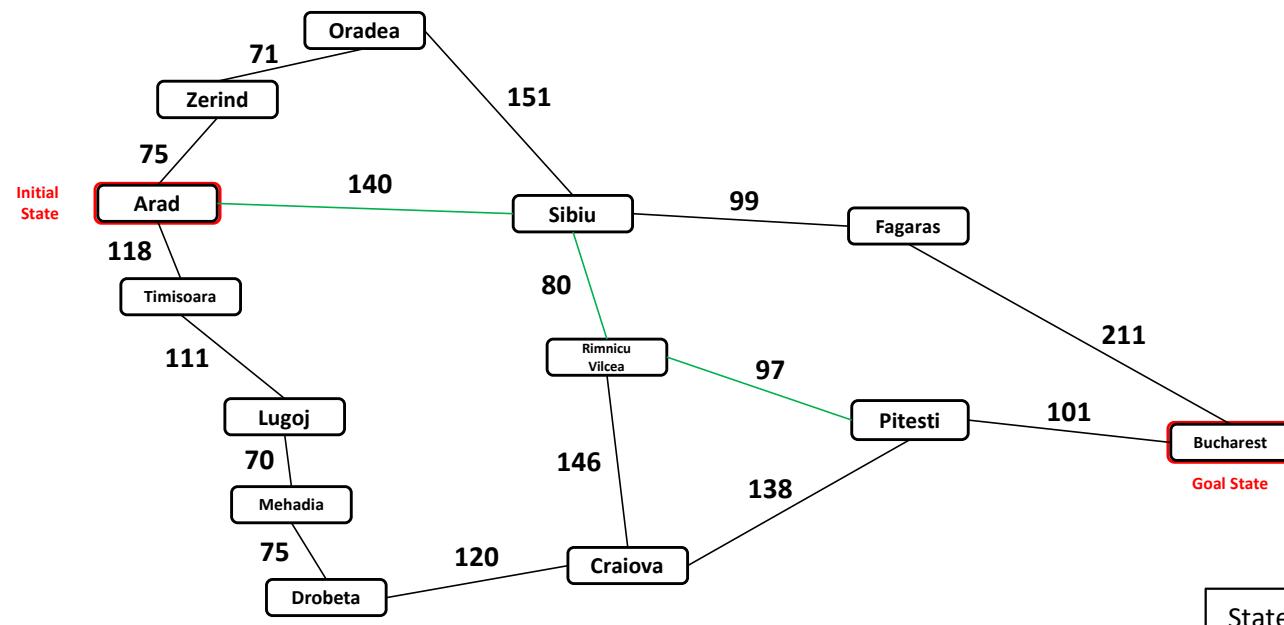
Romanian Roadtrip: A* Search



Romanian Roadtrip: A* Search

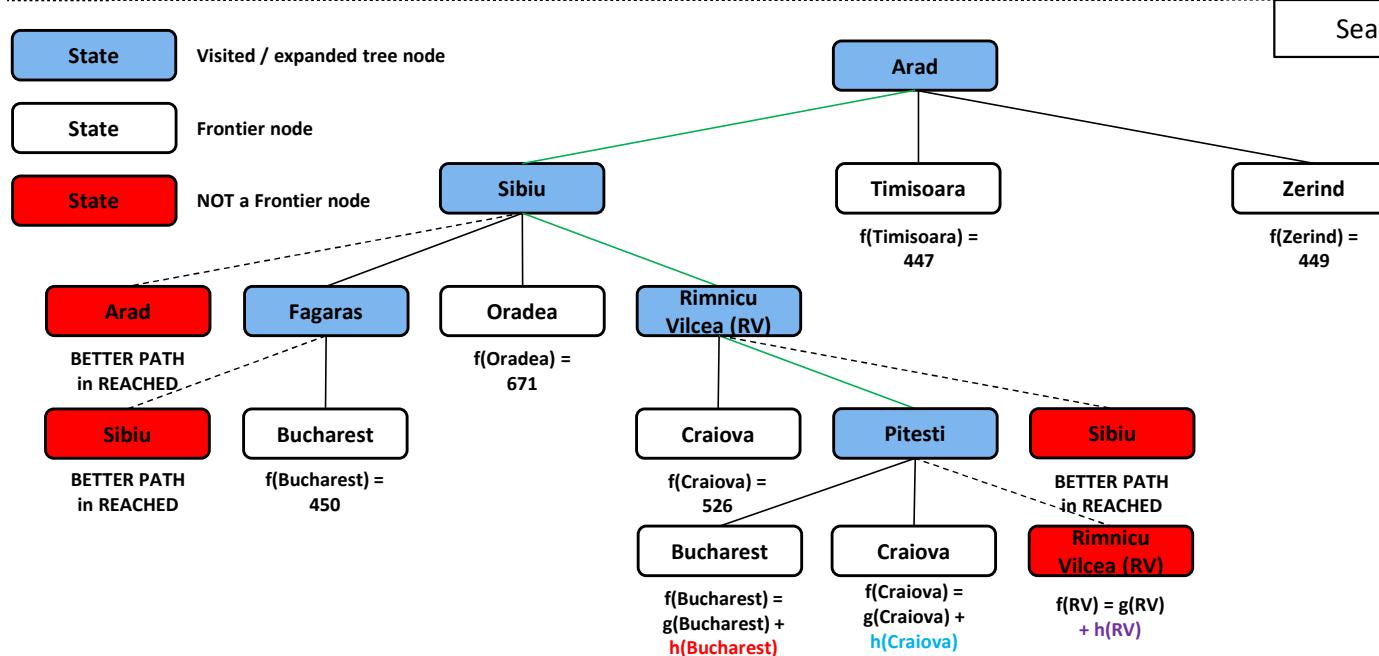


Romanian Roadtrip: A* Search

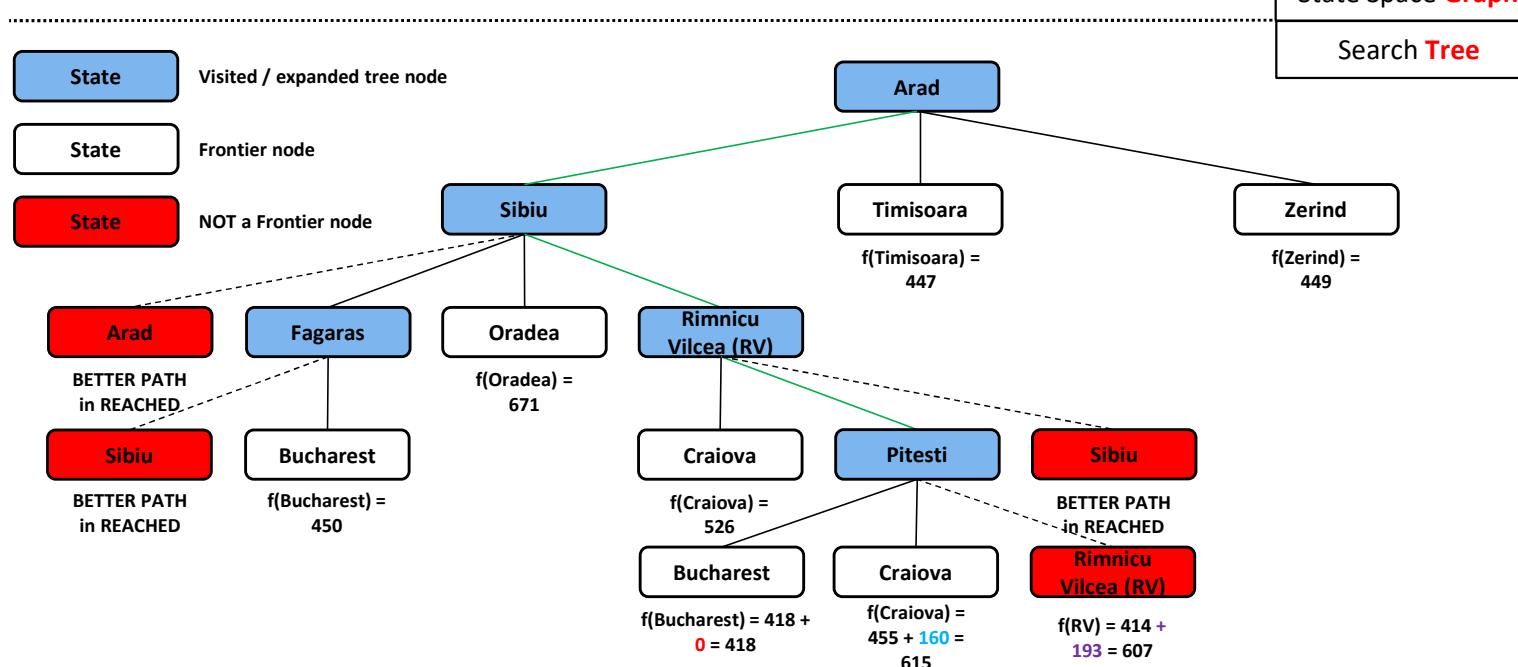
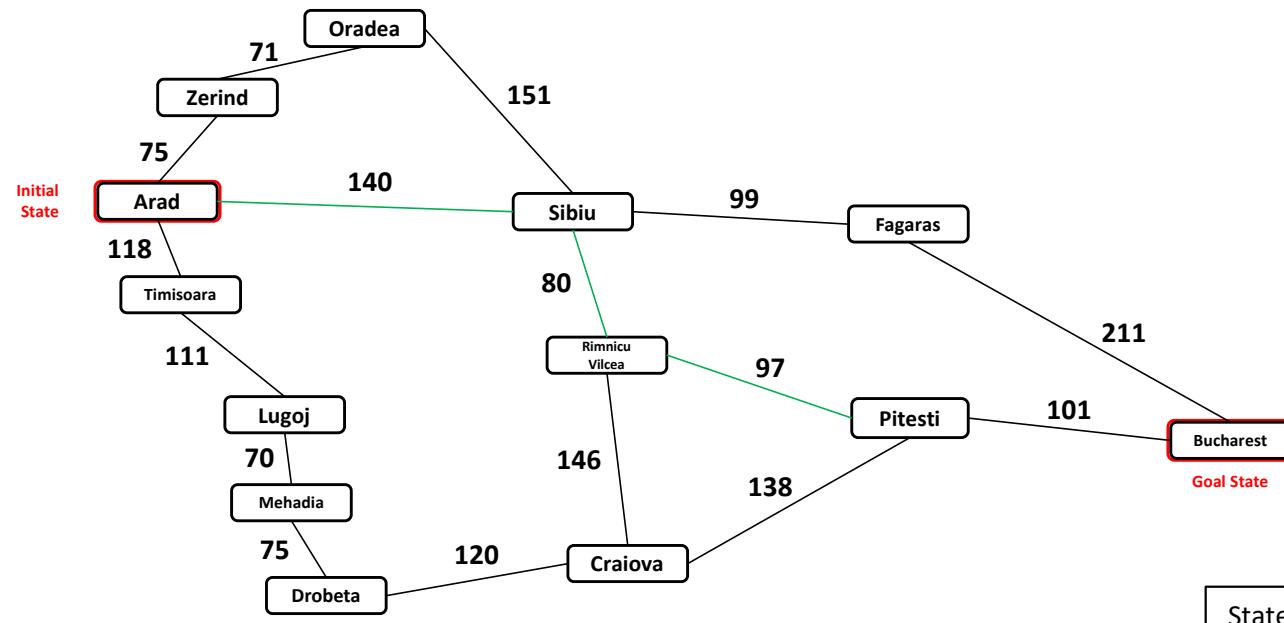


Straight-line distance to Bucharest ($h(\text{State})$):

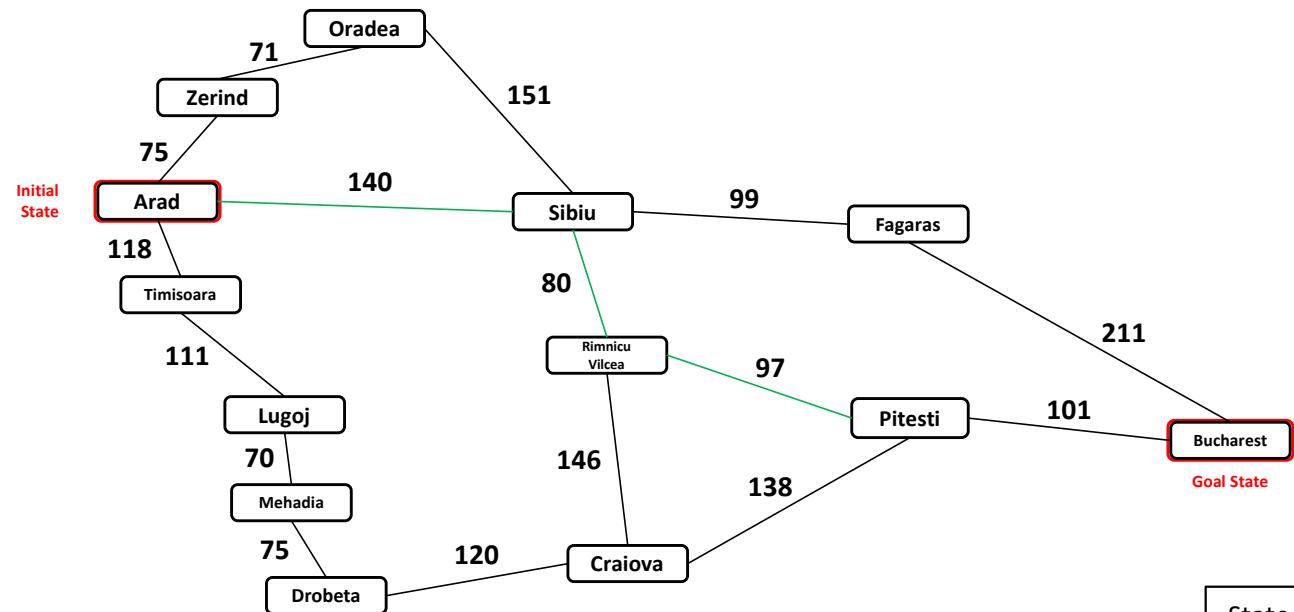
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



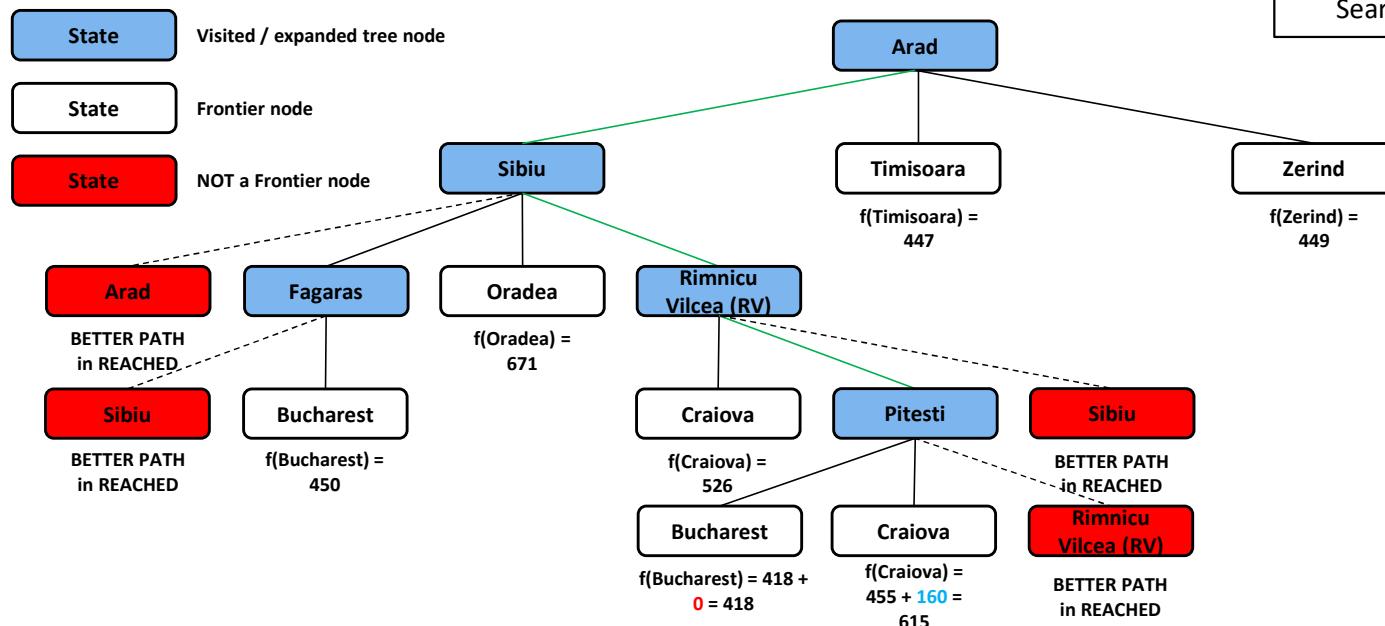
Romanian Roadtrip: A* Search



Romanian Roadtrip: A* Search



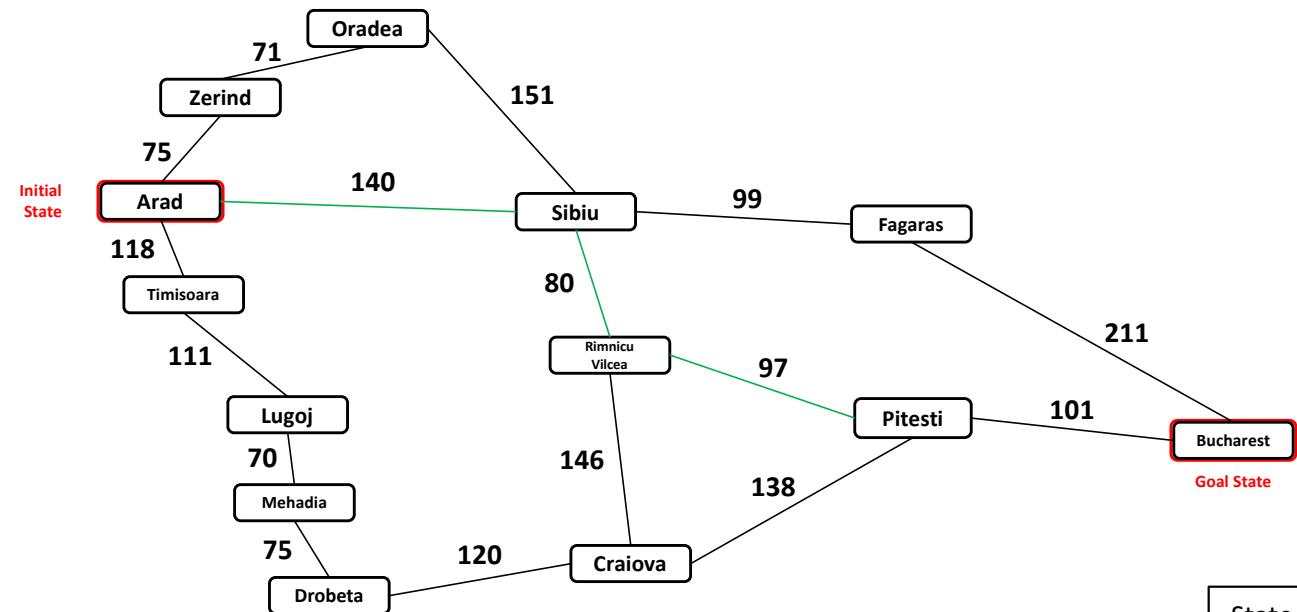
State Space Graph
Search Tree



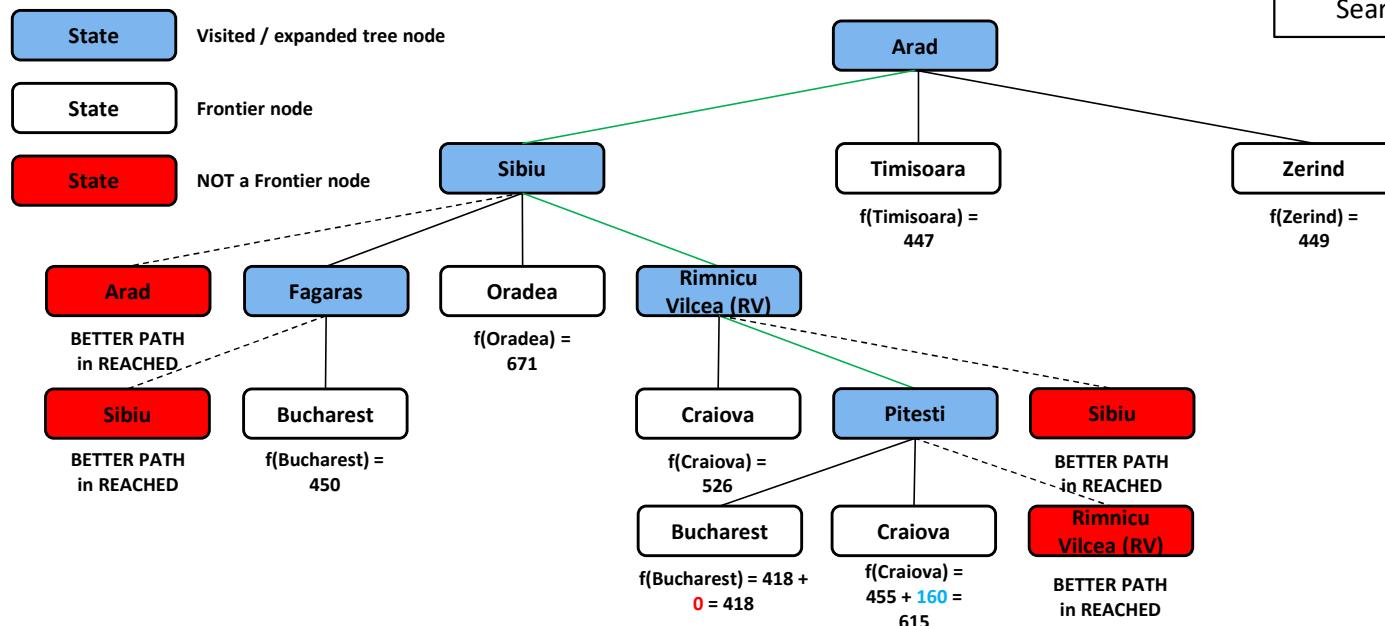
Straight-line distance to Bucharest (h(State)):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



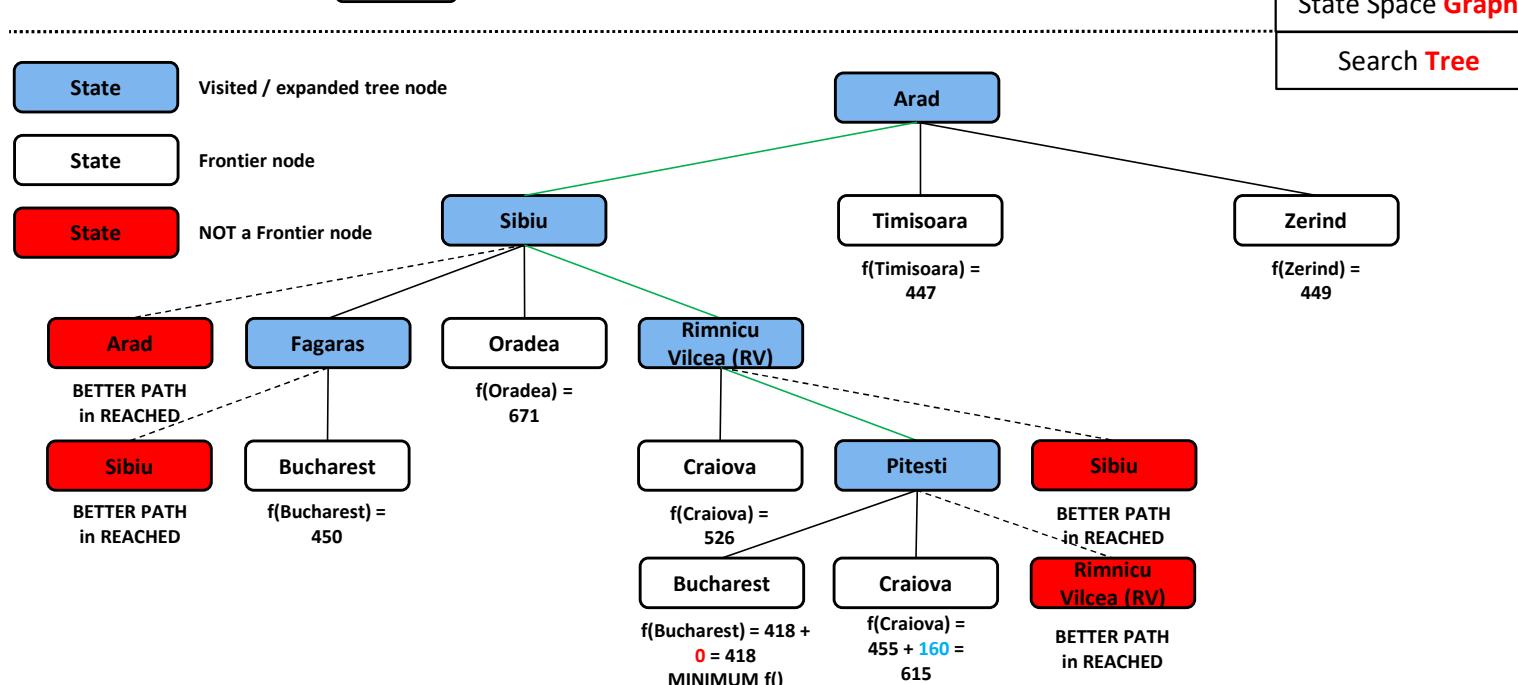
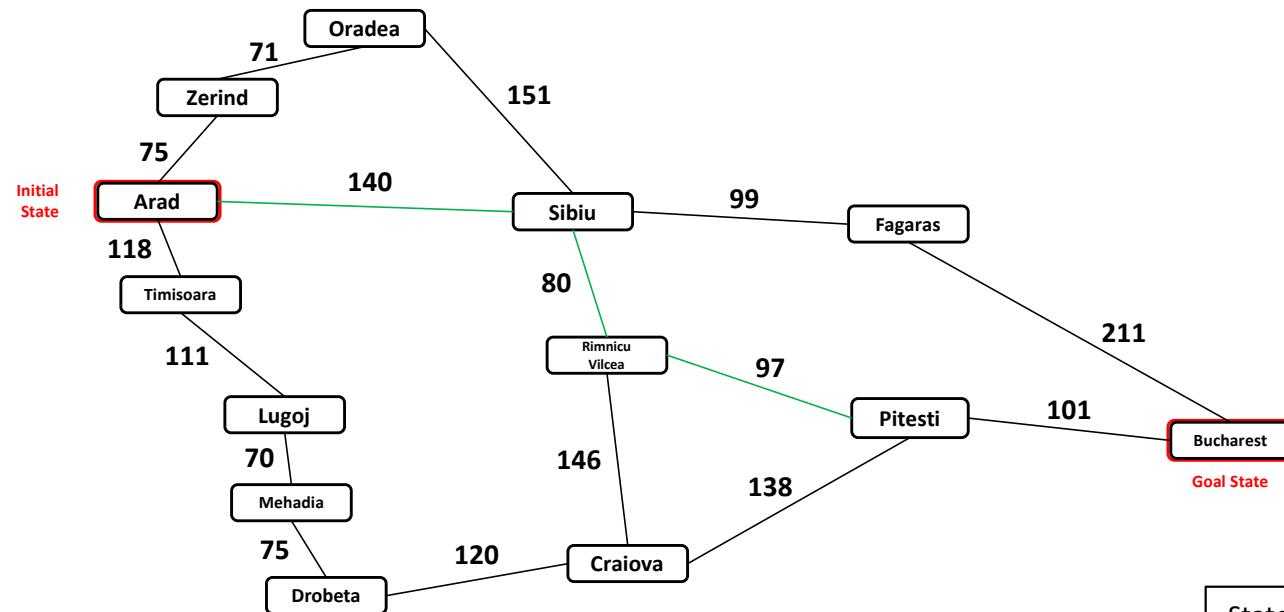
State Space Graph
Search Tree



Straight-line distance to Bucharest (h(State)):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

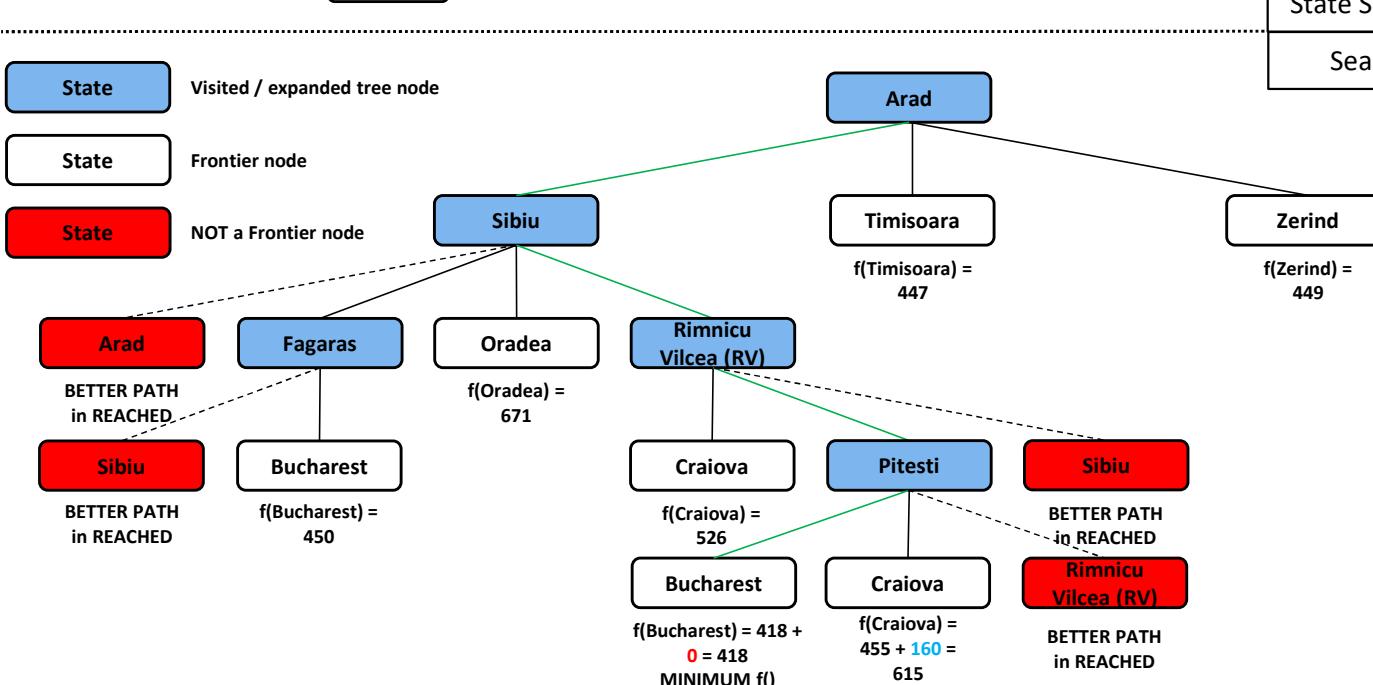
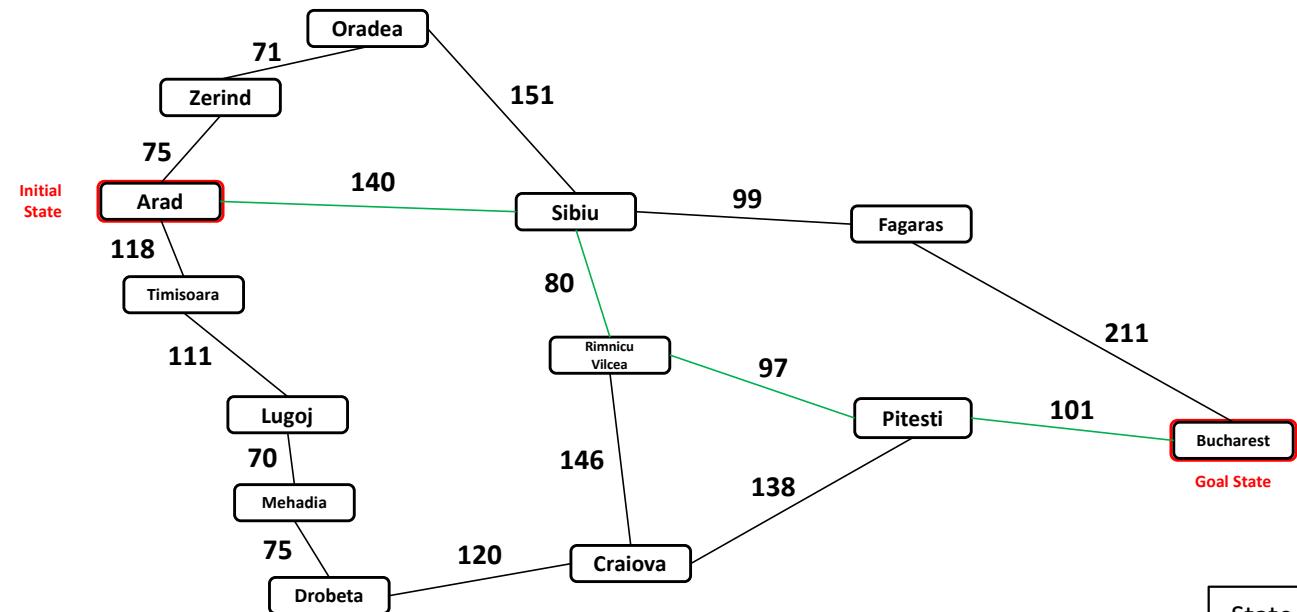
Romanian Roadtrip: A* Search



Straight-line distance to Bucharest (h(State)):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

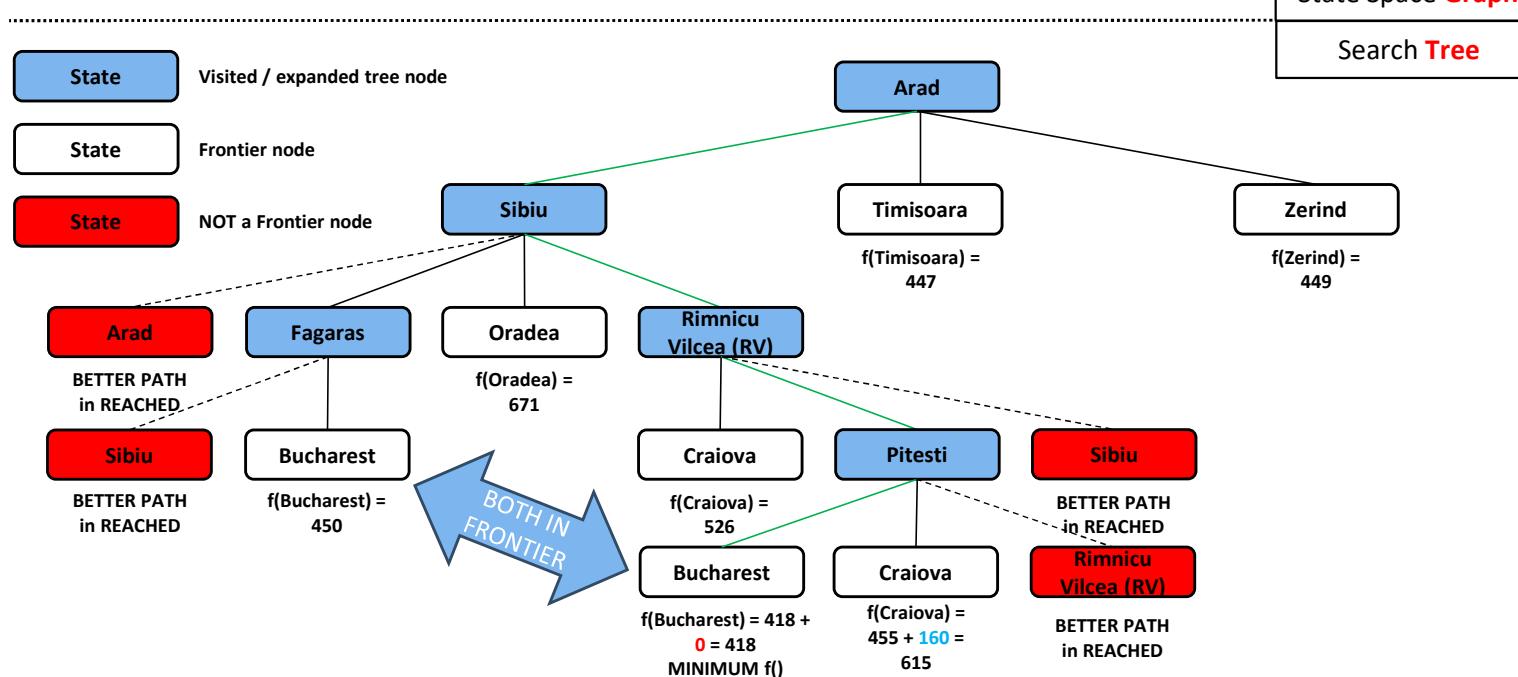
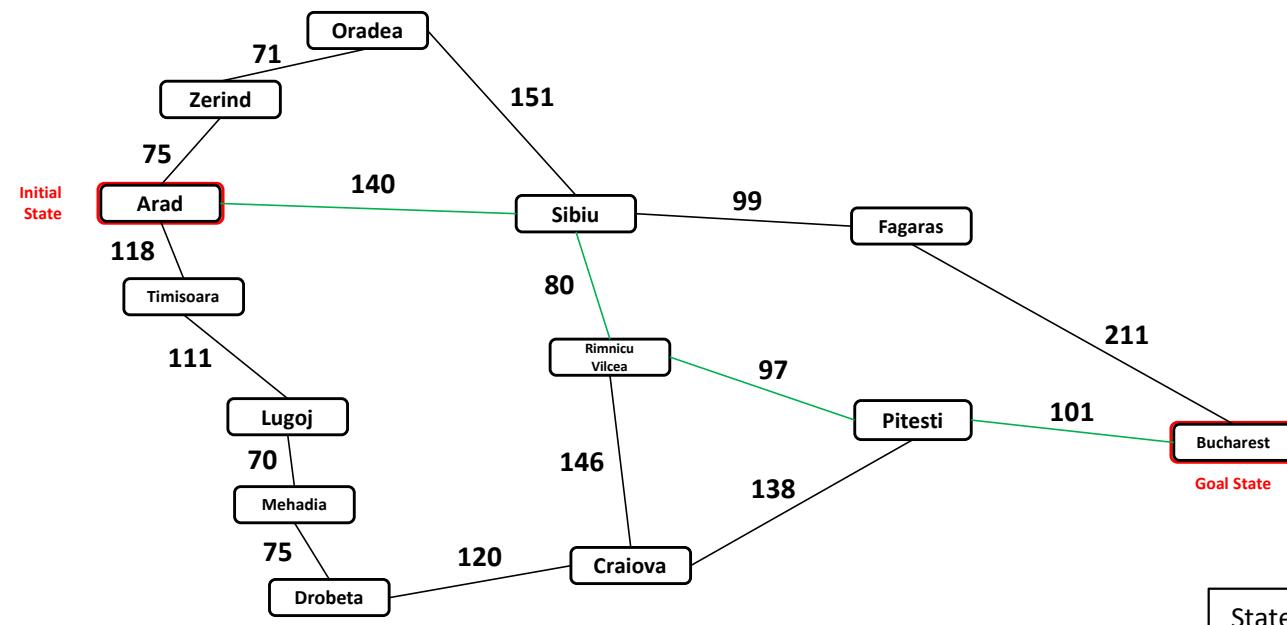
Romanian Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

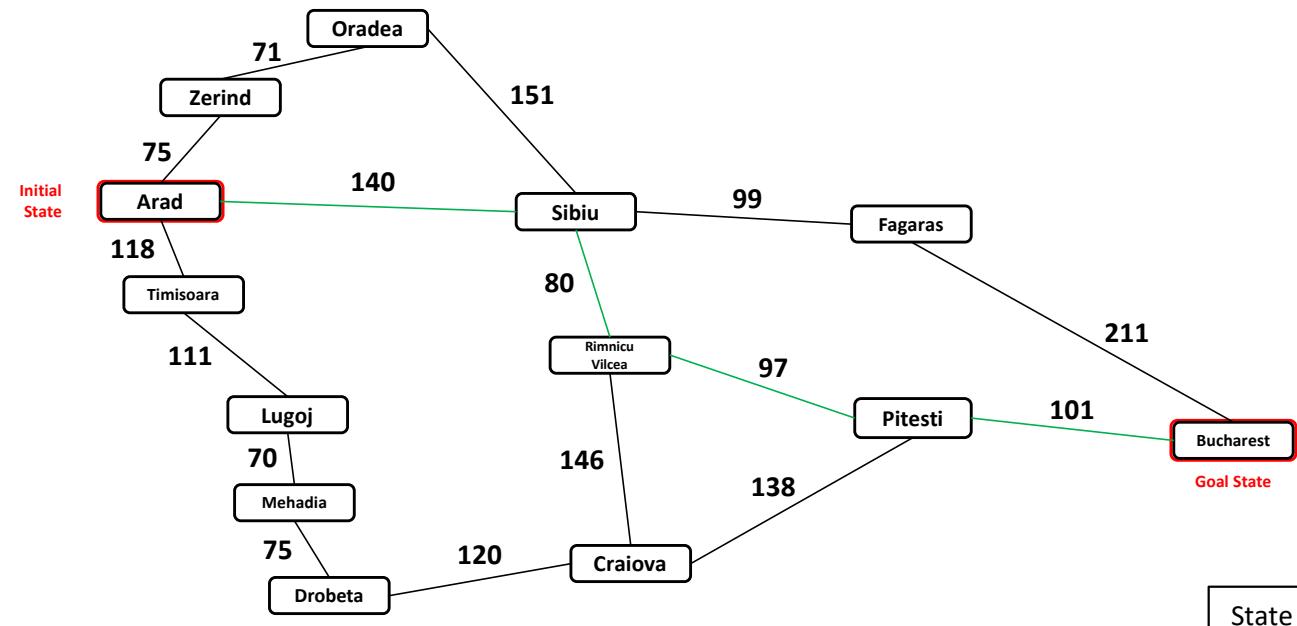
Romanian Roadtrip: A* Search



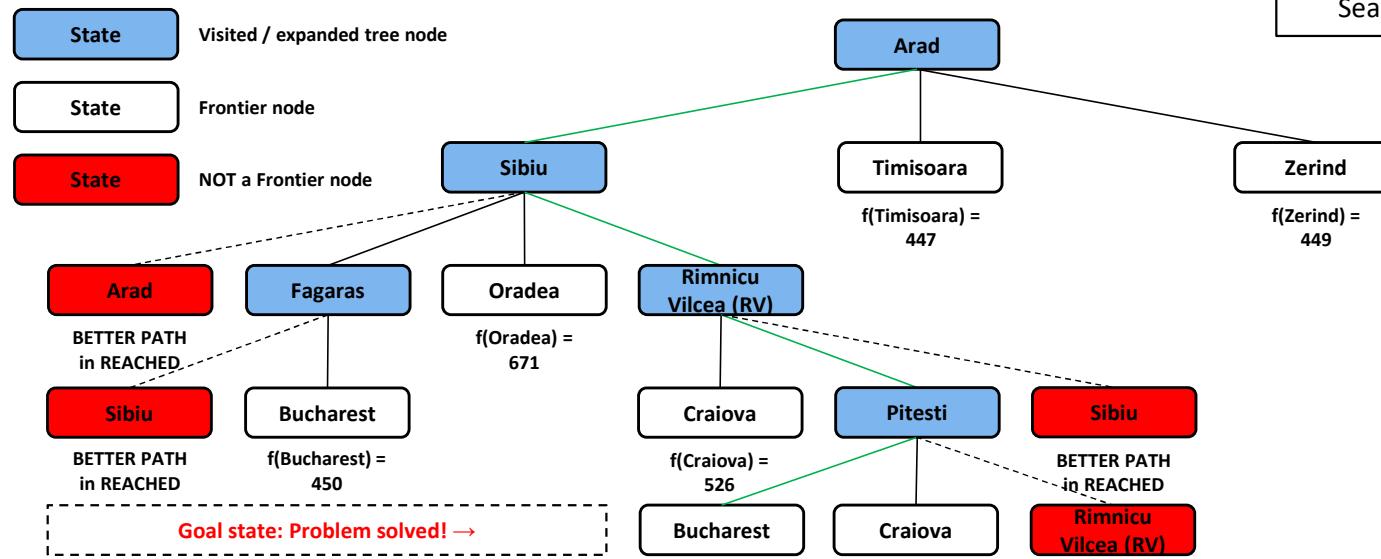
Straight-line distance to Bucharest ($h(\text{State})$):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romanian Roadtrip: A* Search



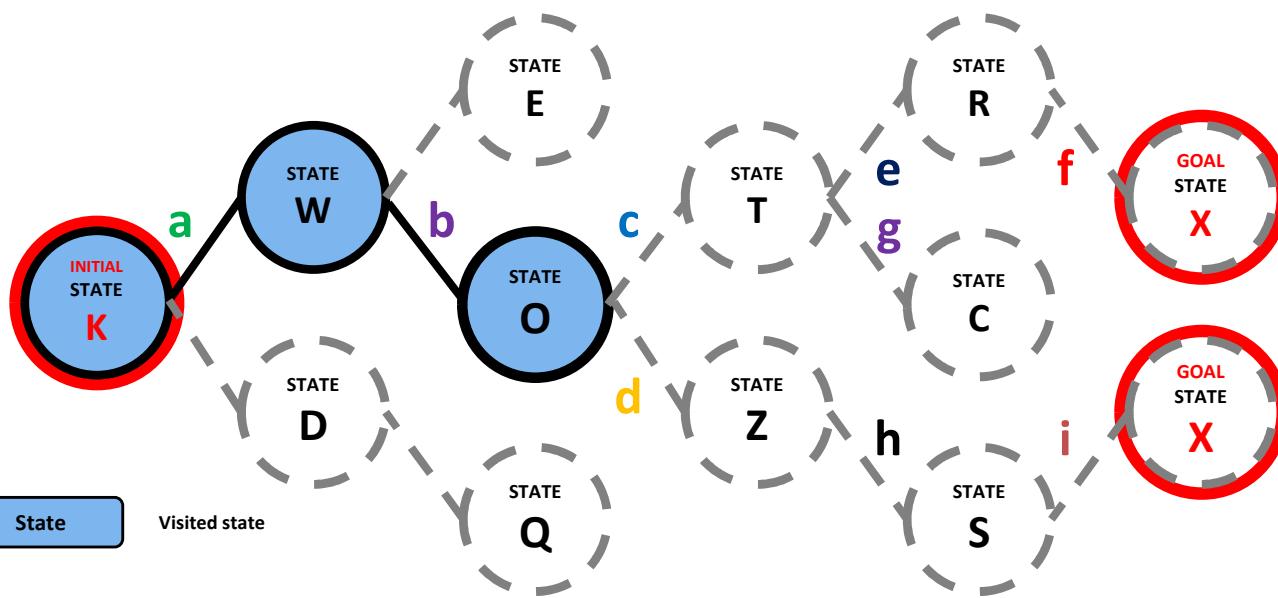
State Space Graph
Search Tree



Straight-line distance to Bucharest (h(State)):

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

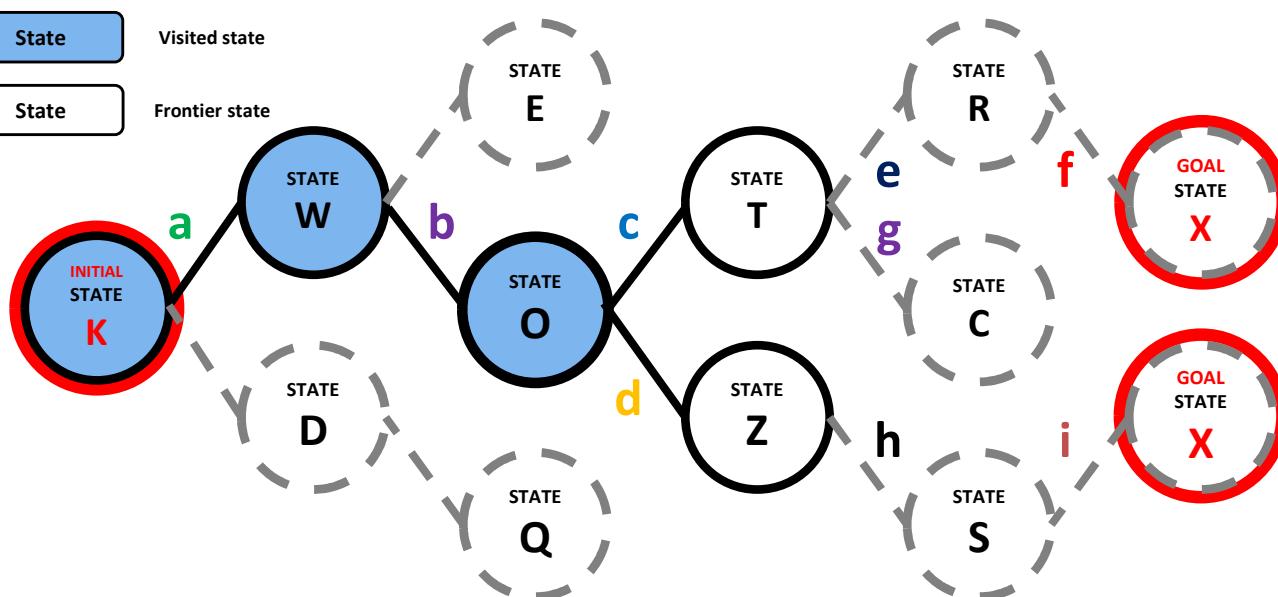
Hill Climbing Search vs. A* Search



Best First Search:
Go to T or Z?
 $f(T) = c$

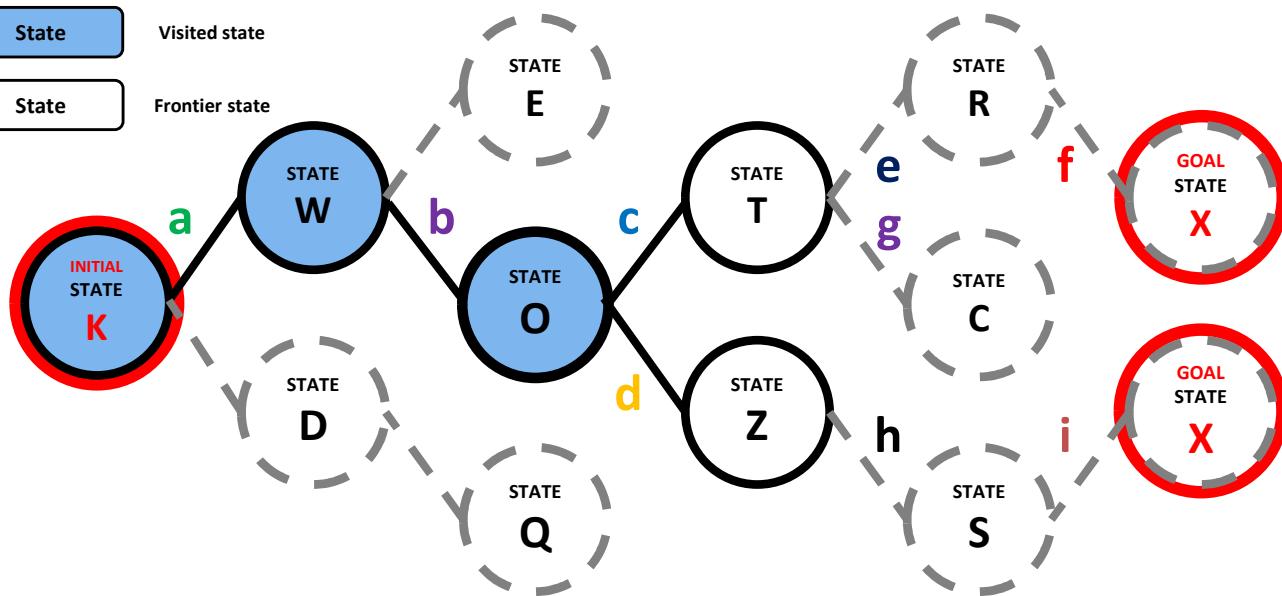
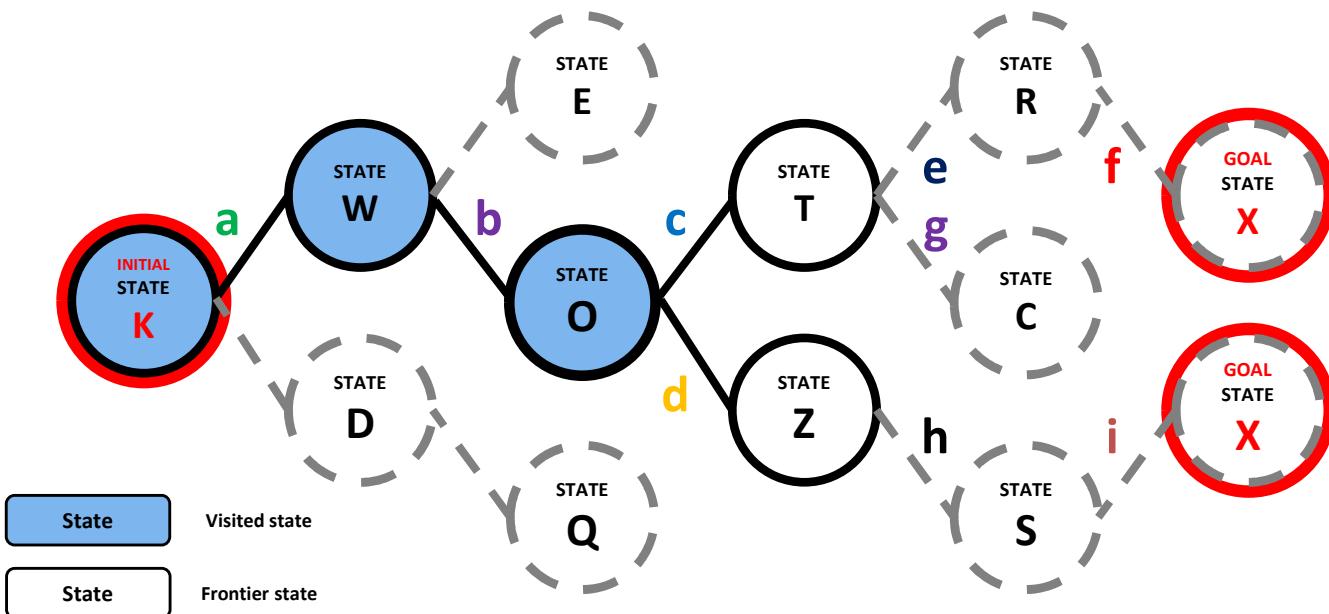
$$f(Z) = d$$

Pick state with min $f()$



A* Search:
Expand T or Z?
 $f(T) = g(T) + h(T)$
 $f(T) = a + b + c + h(T)$
 $f(Z) = g(Z) + h(Z)$
 $f(T) = a + b + d + h(Z)$
 Pick state with min $f()$

Greedy Best First Search vs. A* Search



Greedy Best First:
Expand T or Z?
 $f(T) = h(T)$

$$f(Z) = h(Z)$$

Pick state with min $f()$

A* Search:
Expand T or Z?
 $f(T) = g(T) + h(T)$
 $f(T) = a + b + c + h(T)$

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

$$f(T) = a + b + d + h(Z)$$

Pick state with min $f()$