

CS 581

Advanced Artificial Intelligence

January 17, 2024

Announcements / Reminders

- Please follow the Week 02 To Do List instructions (if you haven't already):
- **UPDATED Exam Dates:**
 - **Midterm:**
 - WAS: ~~02/28/2024 during Wednesday lecture time~~
 - IS: ~~02/21/2024 during Wednesday lecture time~~
 - **Final:**
 - WAS: ~~04/24/2024 during Wednesday lecture time~~
 - IS: ~~04/22/2024 during Monday lecture time~~

Plan for Today

- Intelligent Agents
- Solving problems by Searching
- A* Algorithm

Designing the Agent for the Task

Analyze the
Problem / Task
(PEAS)

Select Agent
Architecture

Select Internal
Representations

Apply
Corresponding
Algorithms

Agent Structure / Architecture

Agent = Architecture + Program

Typical Agent Architectures

- **Simple reflex agent:** uses condition-action rules
- **Model-based reflex agent:** keeps track of the unobserved parts of the environment by maintaining internal state:
 - “how the world works”: state transition model
 - how percepts and environment is related: sensor model
- **Goal-based reflex agent:** maintains the model of the world and goals to select decisions (that lead to goal)
- **Utility-based reflex agent:** maintains the model of the world and utility function to select PREFERRED decisions (that lead to the best expected utility: avg (EU * p))

Designing the Agent for the Task

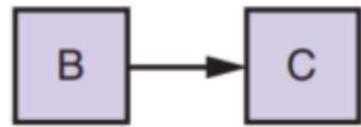
Analyze the
Problem / Task
(PEAS)

Select Agent
Architecture

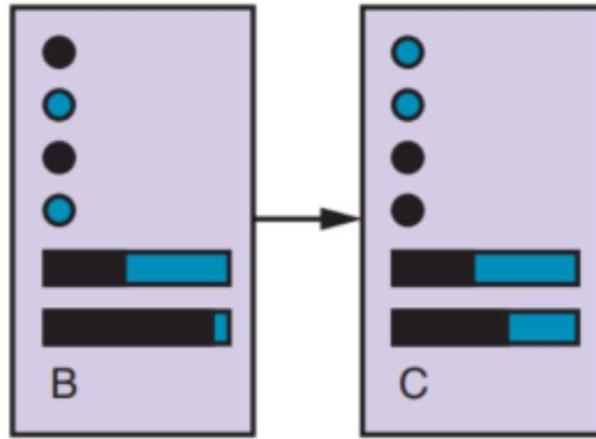
Select Internal
Representations

Apply
Corresponding
Algorithms

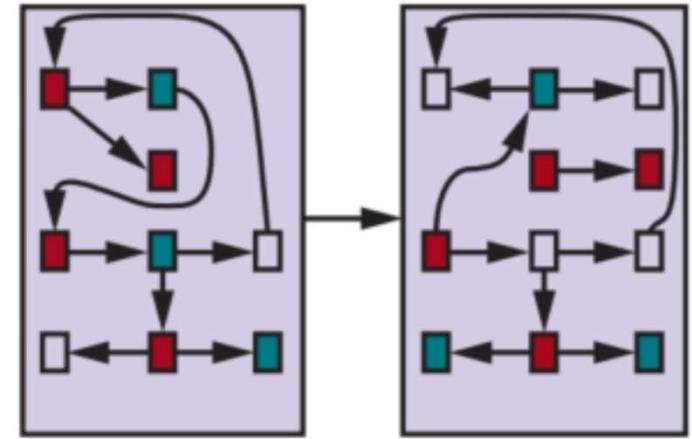
State and Transition Representations



(a) Atomic



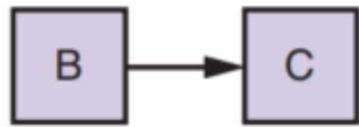
(b) Factored



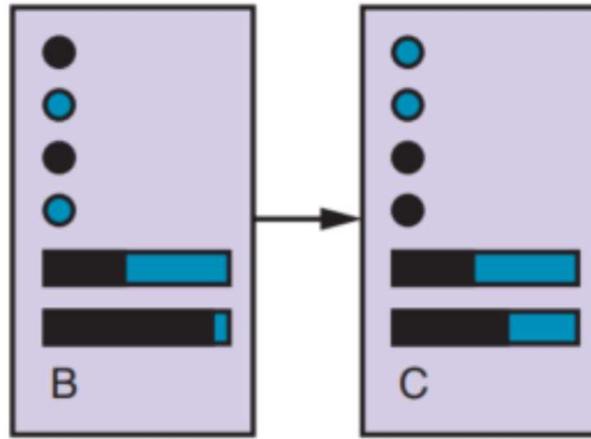
(c) Structured

- **Atomic:** state representation has **NO internal structure**
- **Factored:** state representation includes **fixed attributes (which can have values)**
- **Structured:** state representation includes **objects and their relationships**

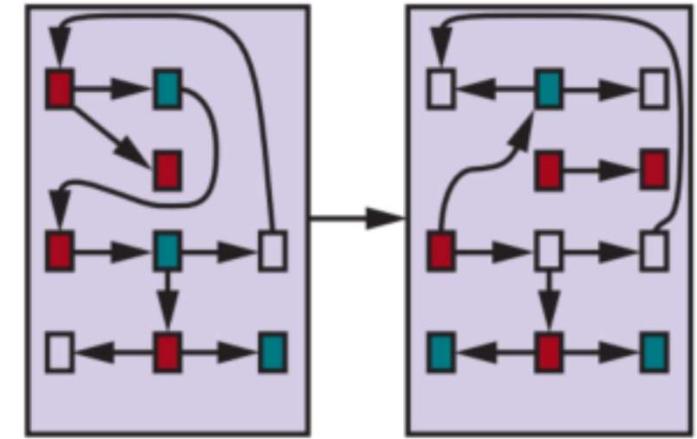
State and Transition Representations



(a) Atomic



(b) Factored



(c) Structured

Complexity, level of detail, expressiveness, more difficult to process

Designing the Agent for the Task

Analyze the
Problem / Task
(PEAS)

Select Agent
Architecture

Select Internal
Representations

Apply
Corresponding
Algorithms

State Space

Finite State Machine: A Turnstile

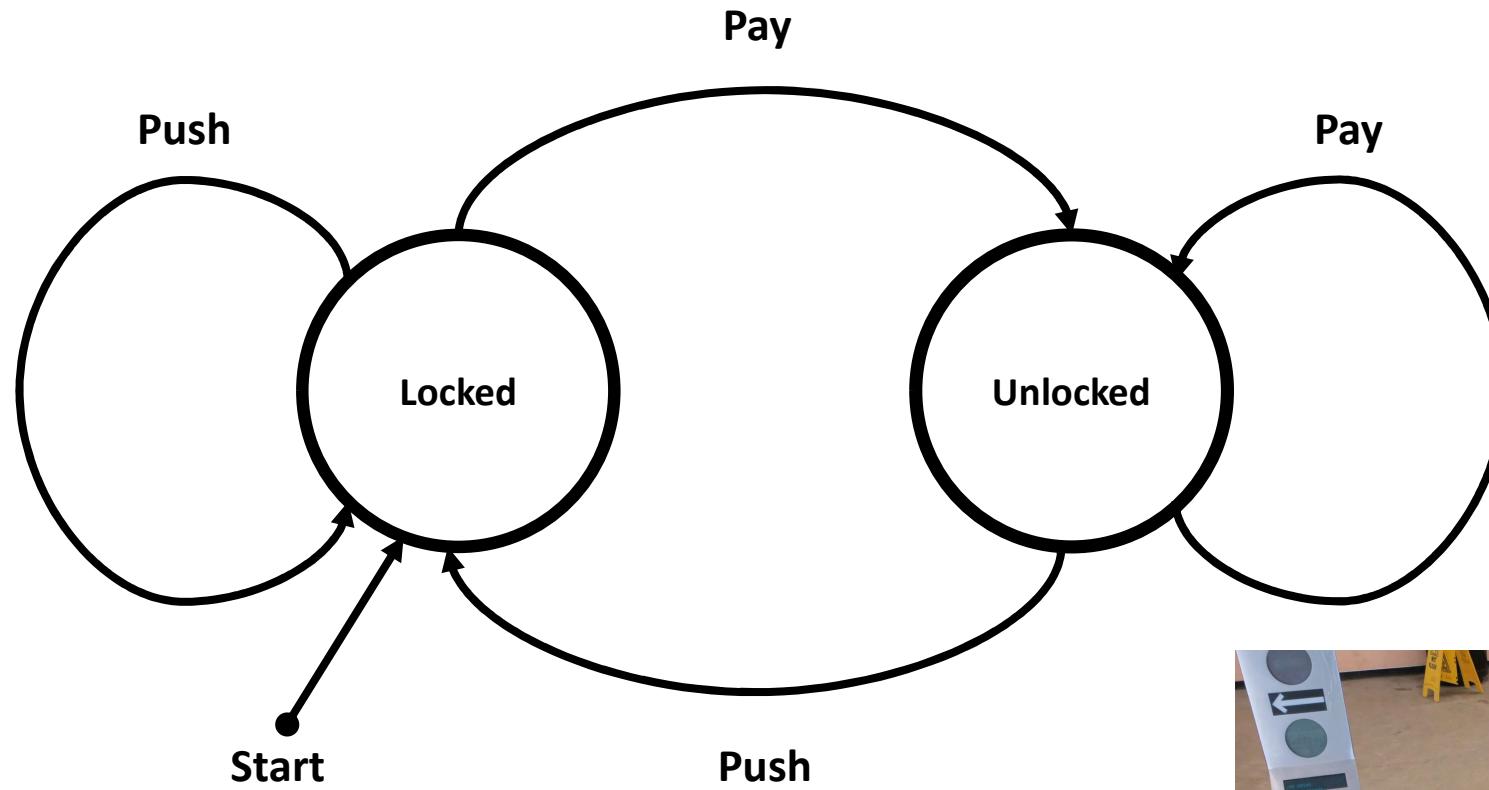
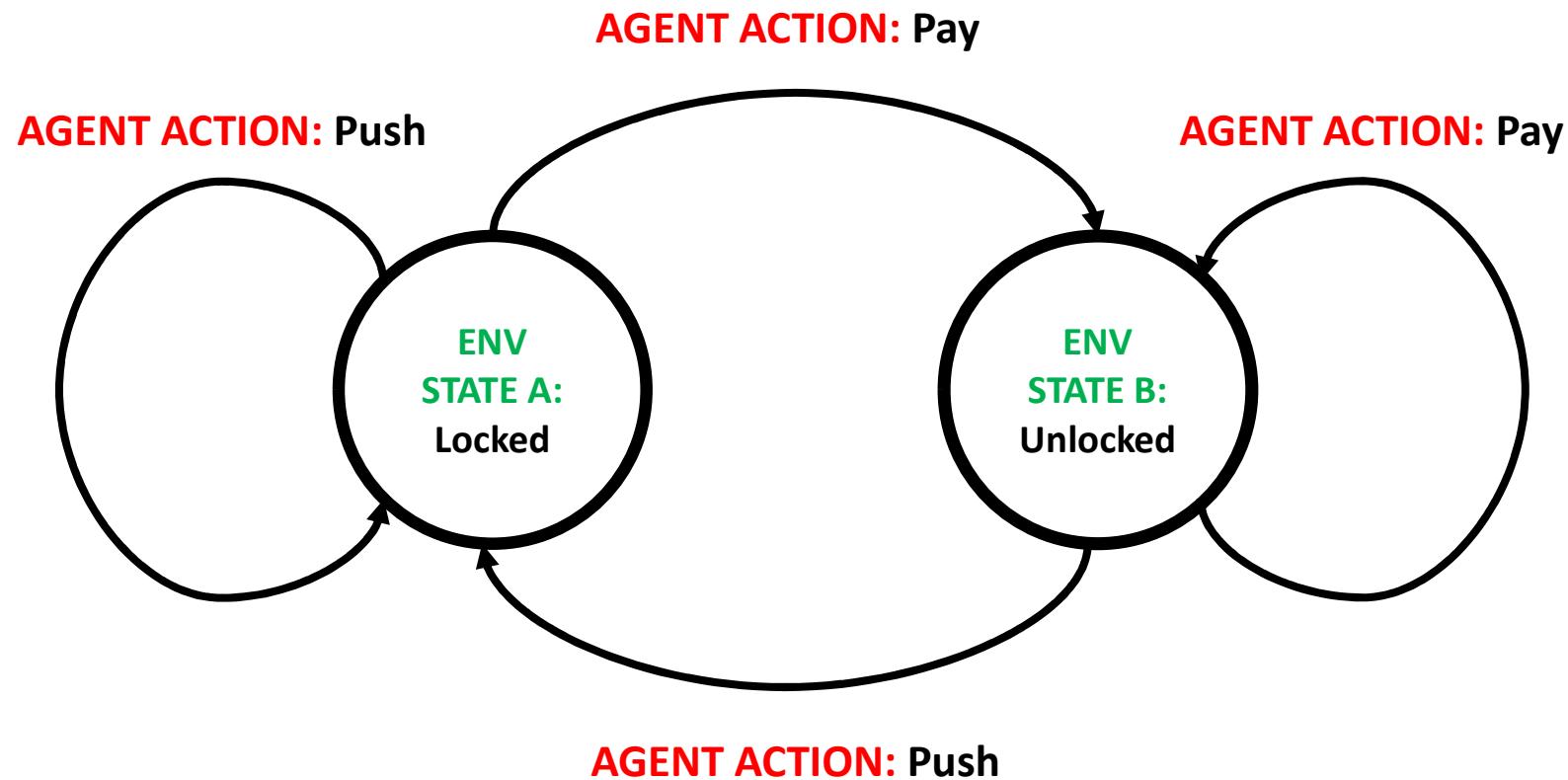
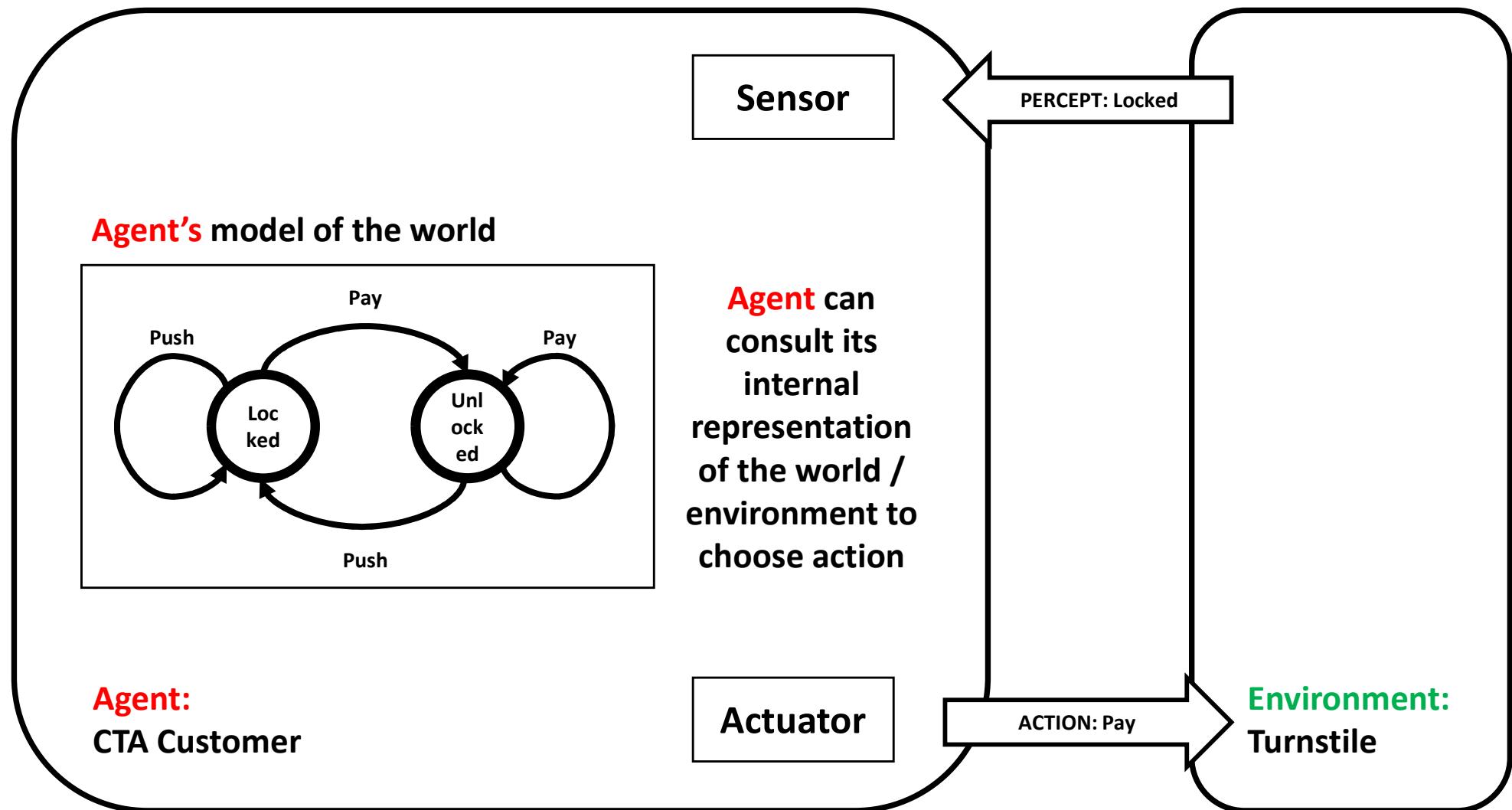


Image source: Wikipedia

Finite State Machine: A Turnstile

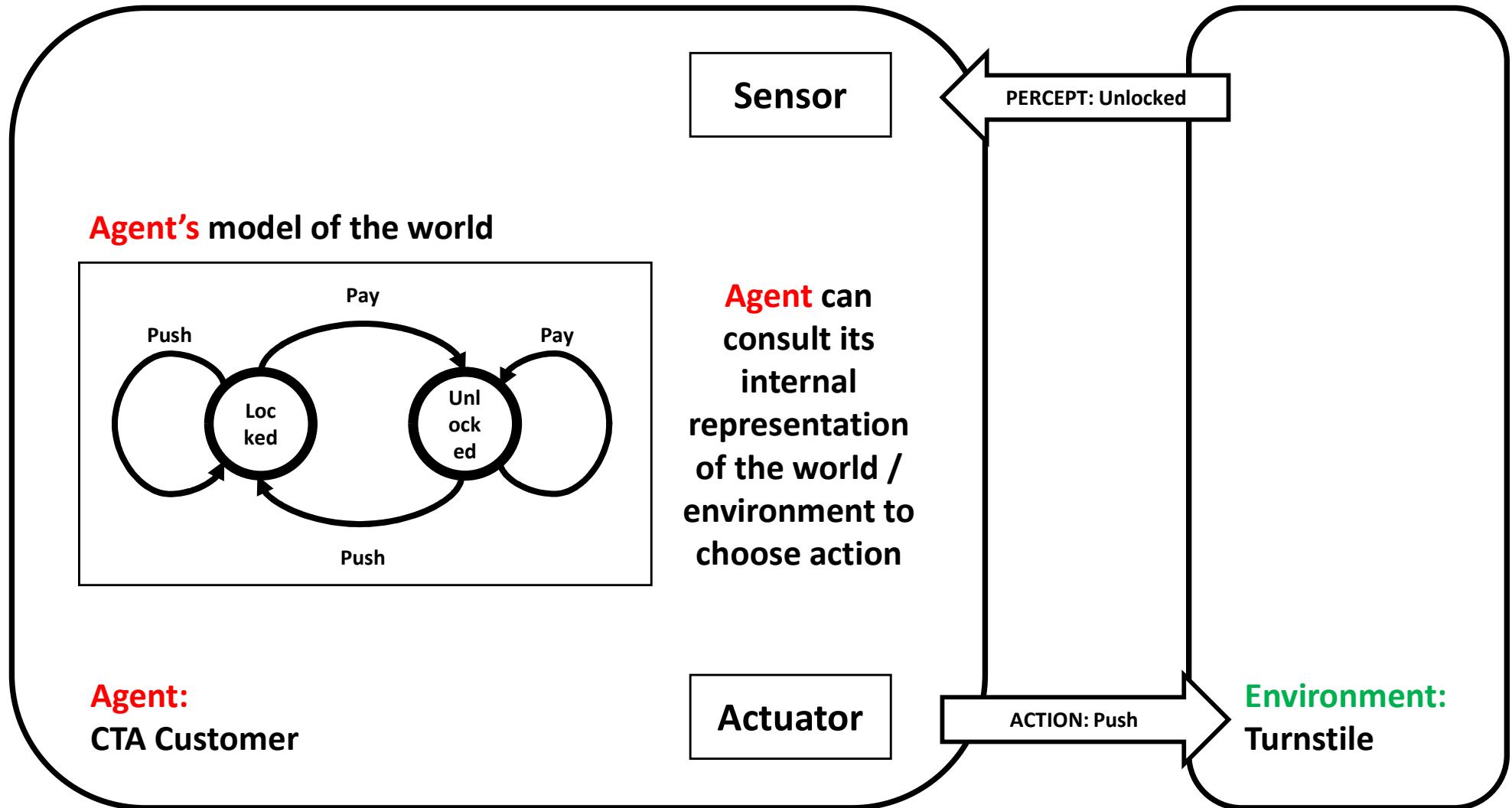


Model-based Reflex Agent Example



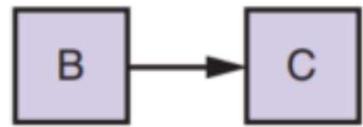
Note: This problem could be easily solved with a simple (without internal model) reflex agent.

Model-based Reflex Agent Example

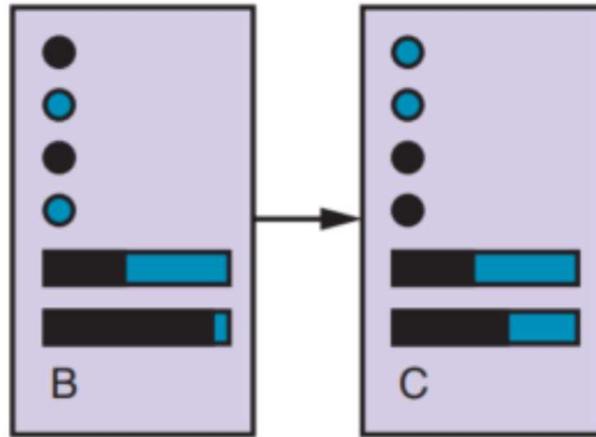


Note: This problem could be easily solved with a simple (without internal model) reflex agent.

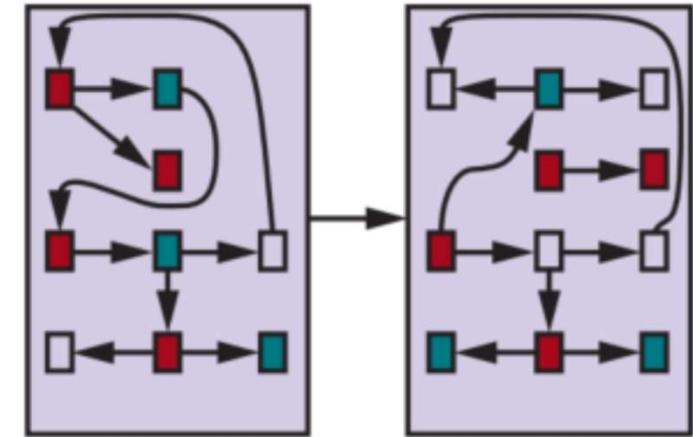
Representations: Examples



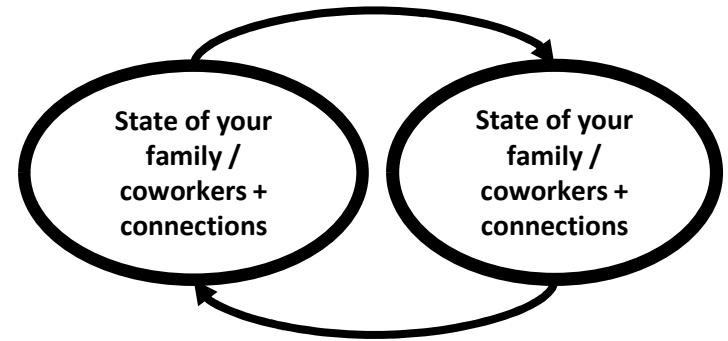
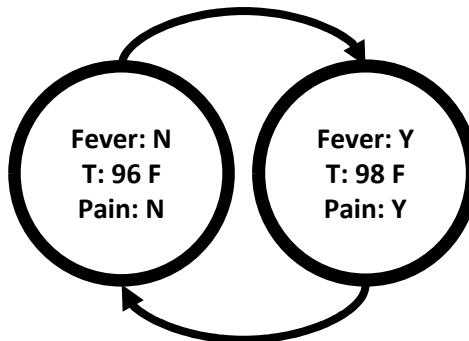
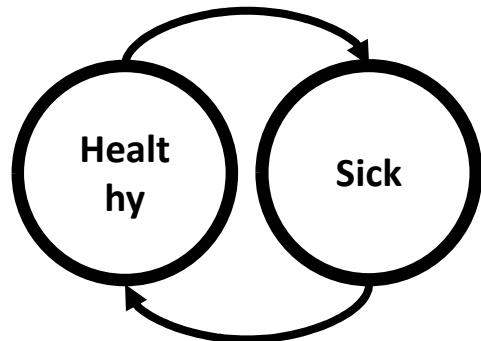
(a) Atomic



(b) Factored



(c) Structured



Designing the Agent for the Task

Analyze the
Problem / Task
(PEAS)

Select Agent
Architecture

Select Internal
Representations

Apply
Corresponding
Algorithms

Problem-Solving / Planning Agent

- Context / Problem:
 - correct action is NOT immediately obvious
 - a plan (a sequence of actions leading to a goal) may be necessary
- Solution / Agent:
 - come up with a computational process that will search for that plan
- Planning Agent:
 - uses factored or structured representations of states
 - uses searching algorithms

Planning: Environment Assumptions

Works with a “Simple Environment”:

- Fully observable
- Single agent (for now -> it can be multiagent)
- Deterministic
- Static
- Episodic
- Discrete
- Known to the agent

Problem-Solving Process

- Goal formulation:
 - adopt a goal (think: desirable state)
 - a concrete goal should help you reduce the amount of searching
- Problem formulation:
 - an **abstract** representation of states and actions
- Search:
 - search for solutions within the **abstract** world model
- Execute actions in the solution

Planning: Environment Assumptions

Works with a “Simple Environment”:

- Fully observable
- Single agent (for now -> it can be multiagent)
- Deterministic
- Static
- Episodic
- Discrete
- Known to the agent

Important and helpful:
Such assumptions **GUARANTEE** a
FIXED sequence of actions as a
solution
What does it mean?
You can execute the “plan”
without worrying about incoming
percepts (open-loop control)

Designing the Searching Problem

Analyze and
define the
Problem / Task

Model and build
the State Space

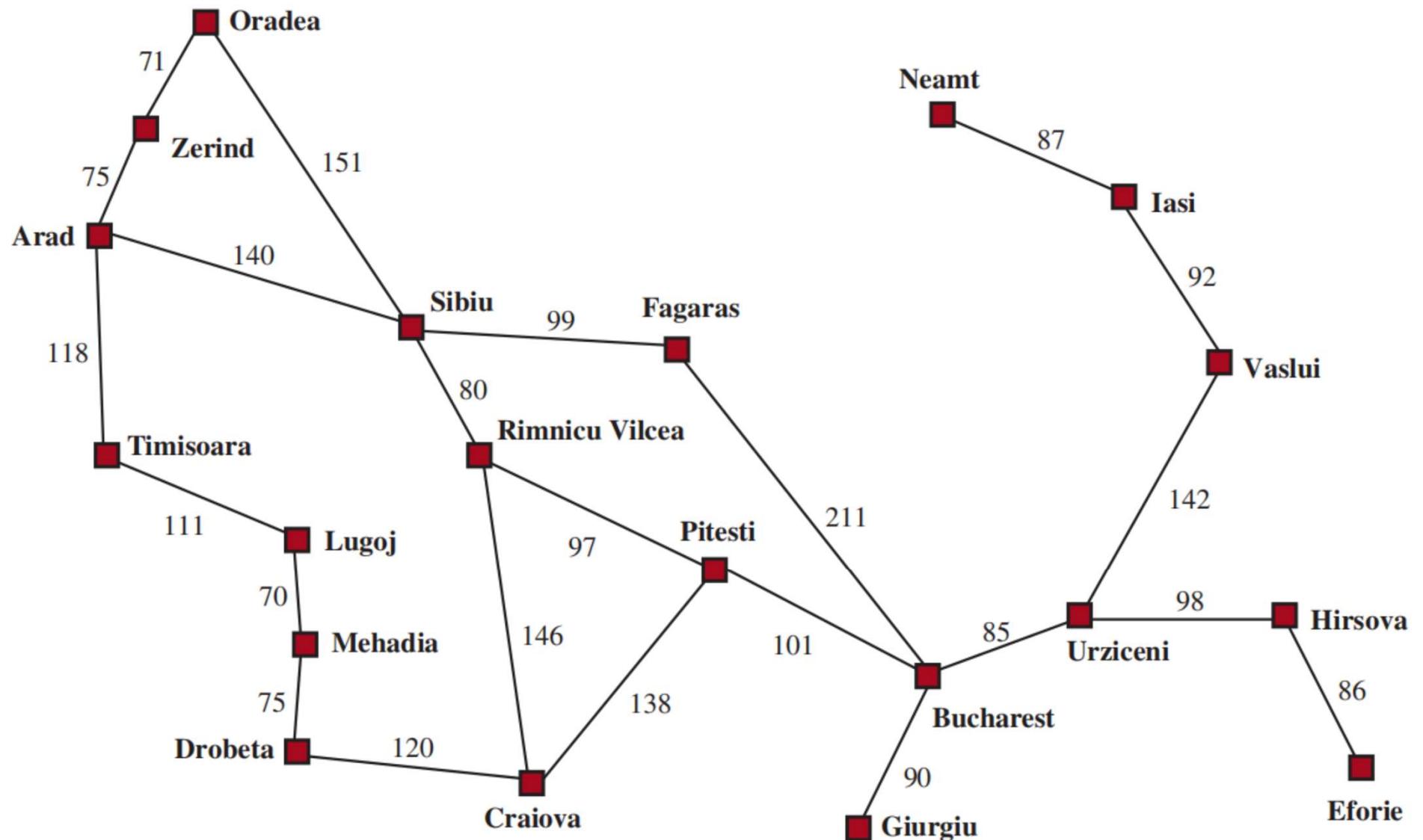
Select searching
algorithm

Search

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

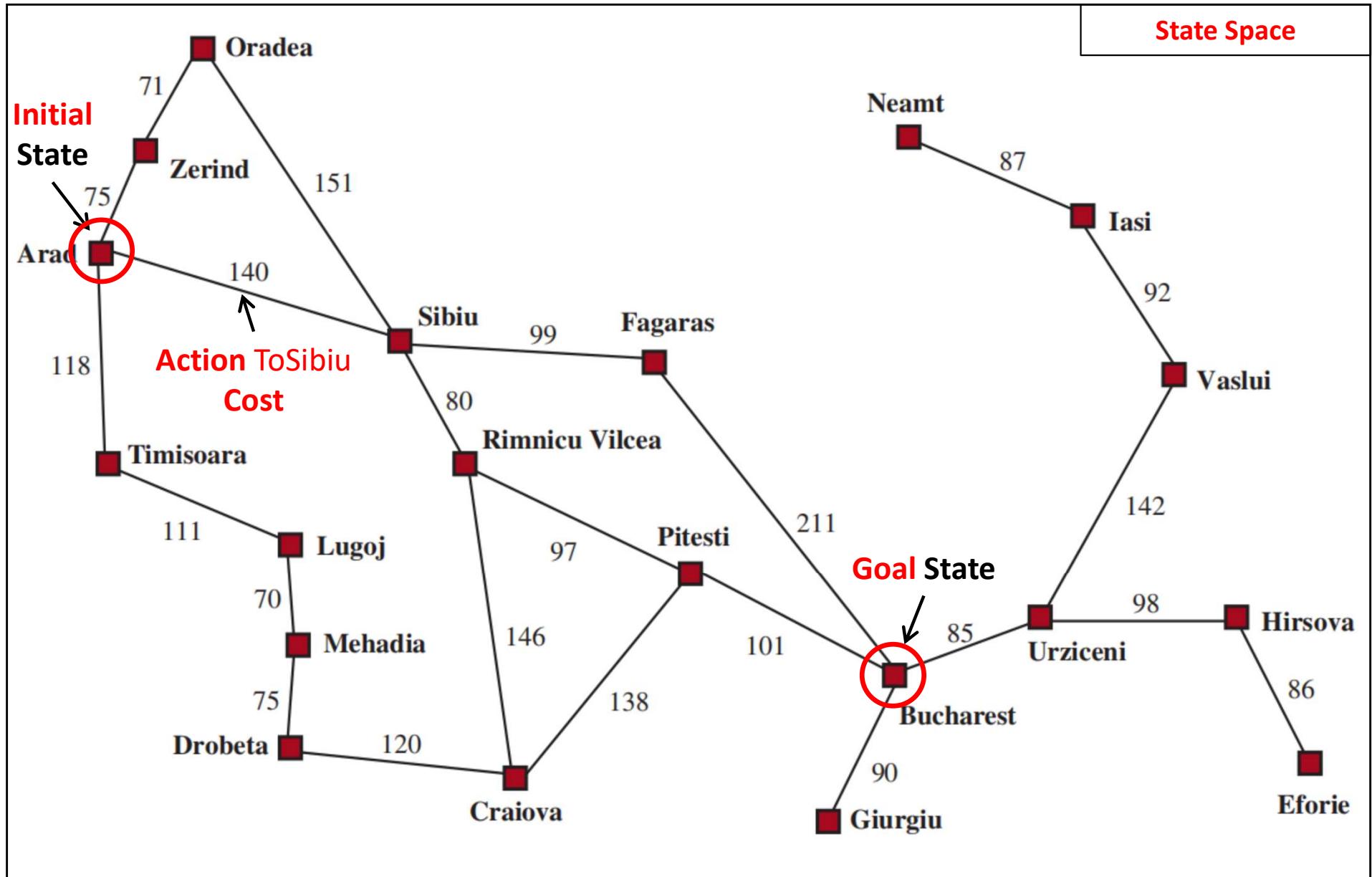


Problem: Get from Arad to Bucharest efficiently (for example: quickly or cheaply).

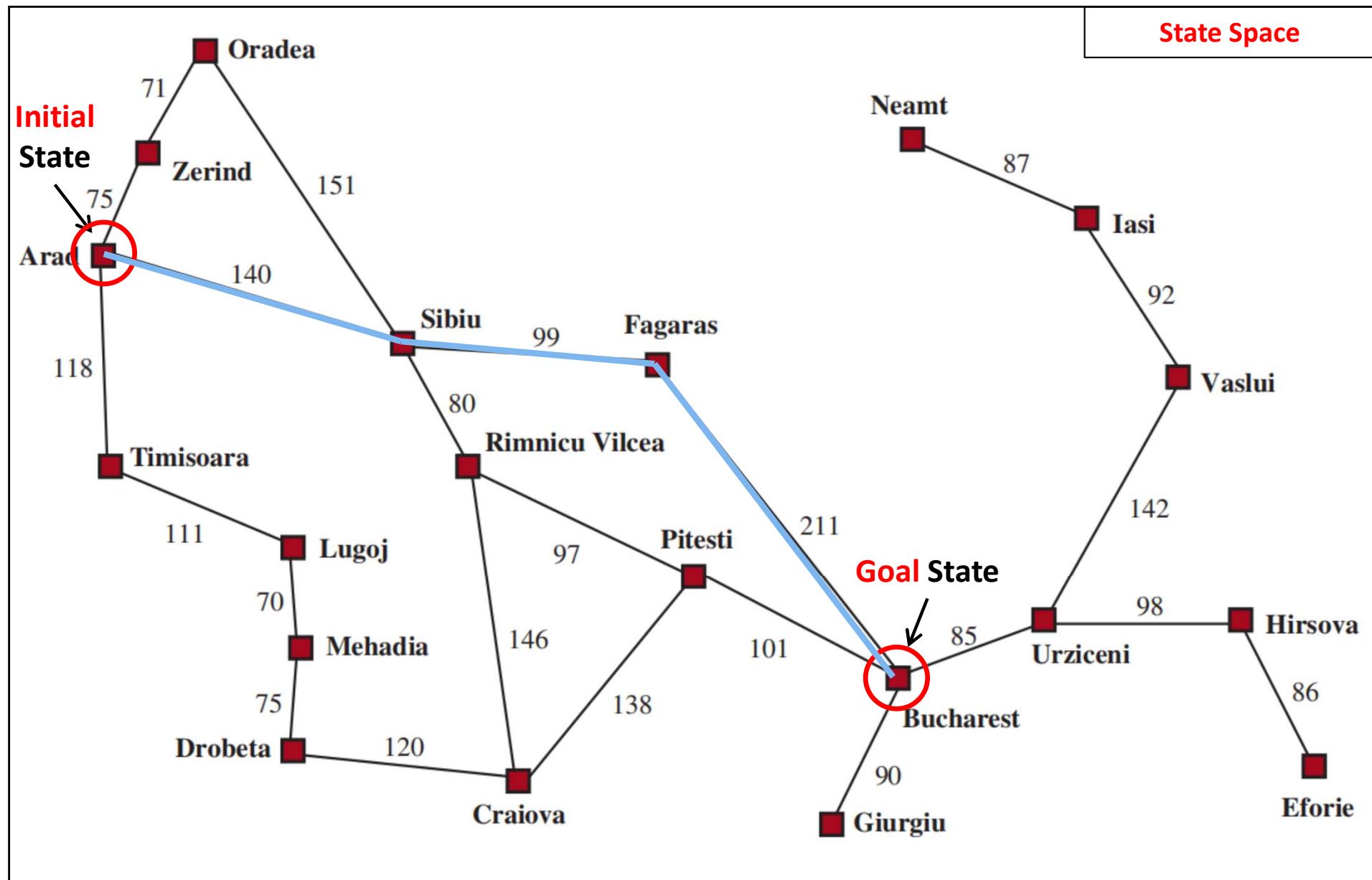
Search Problem: Romanian Roadtrip

- 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: $\text{ActionCost}(\text{Arad}, \text{ToSibiu}, \text{Sibiu}) = 140$

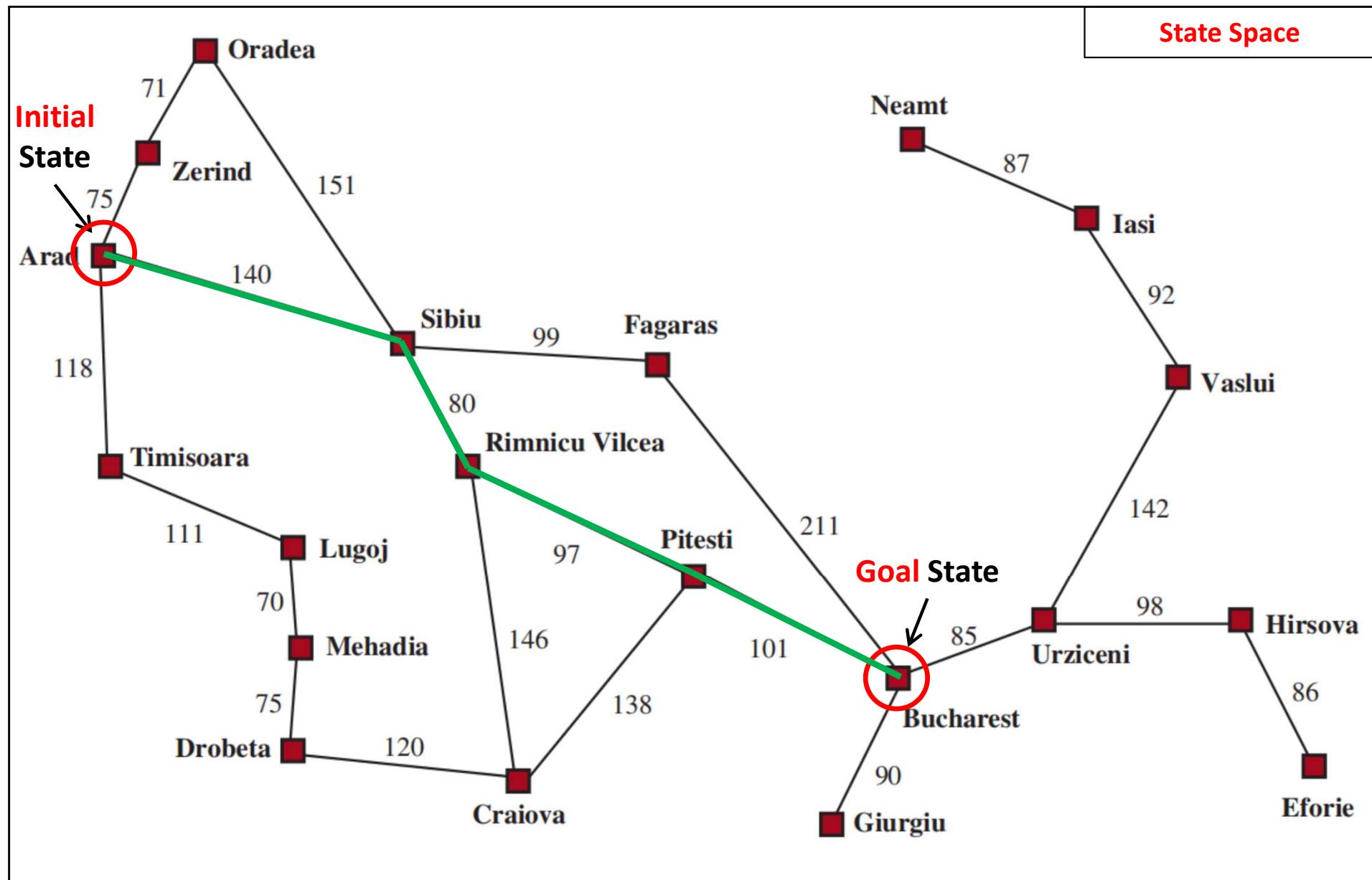
Sample Problem: Romanian Roadtrip



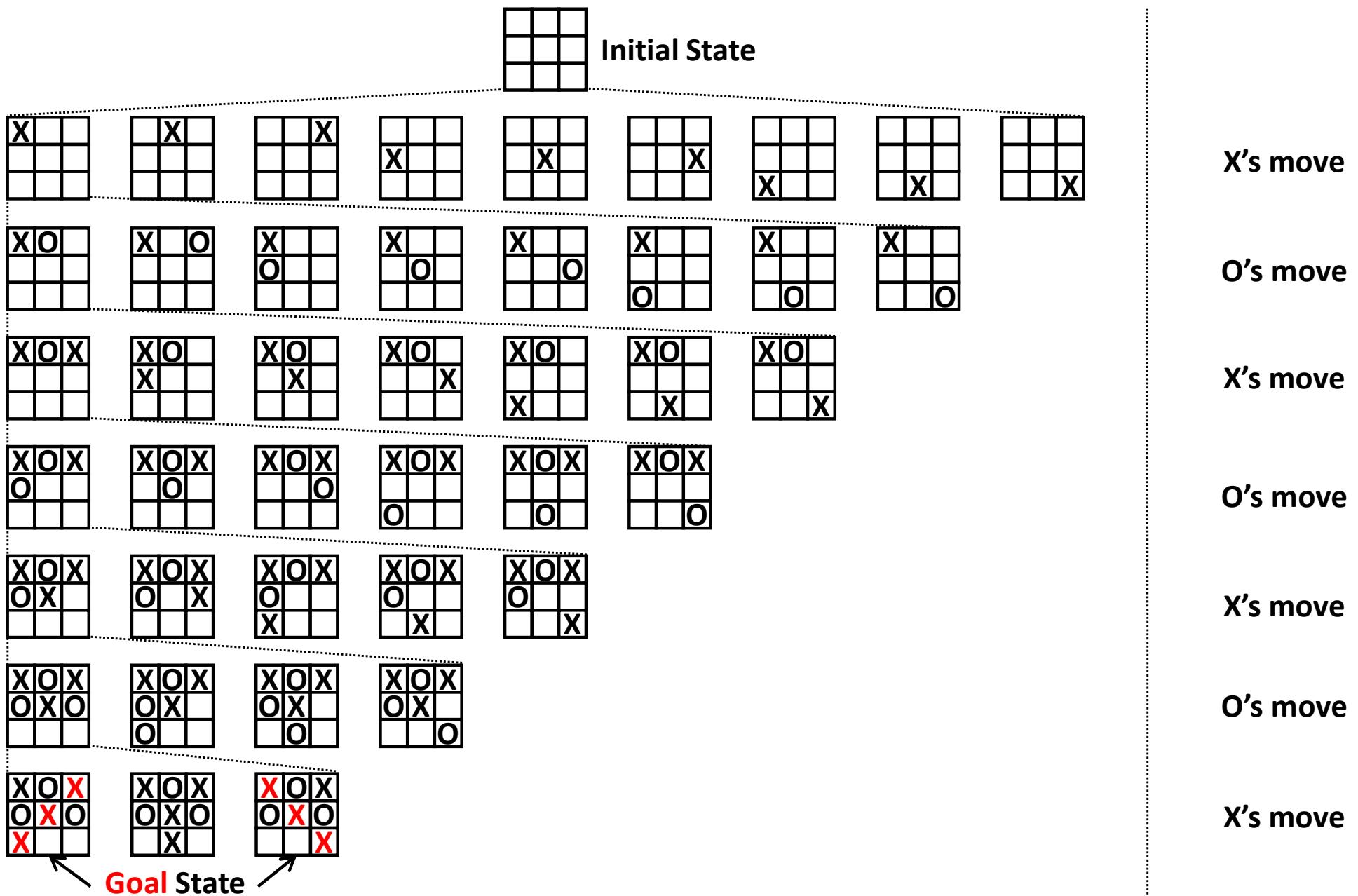
Romanian Roadtrip: Potential Solution



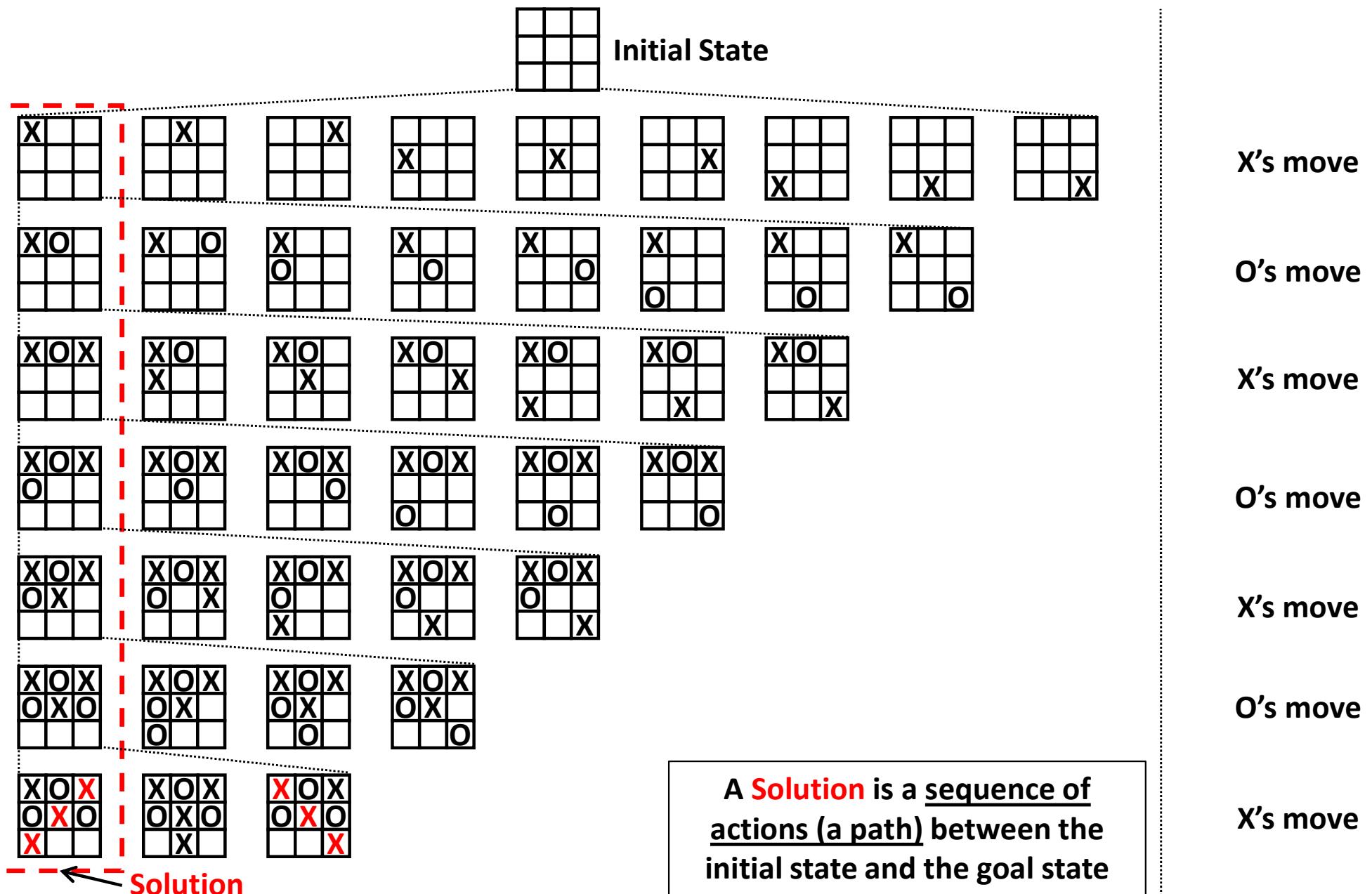
Romanian Roadtrip: Potential Solution



Tic Tac Toe: (Partial) State Space



Tic Tac Toe: Solution



Chess: (First Move) State Space

Initial
State



20 Possible **legal** first moves:

16 pawn moves

4 knight moves



Designing the Searching Problem

Analyze and
define the
Problem / Task

Model and build
the State Space

Select searching
algorithm

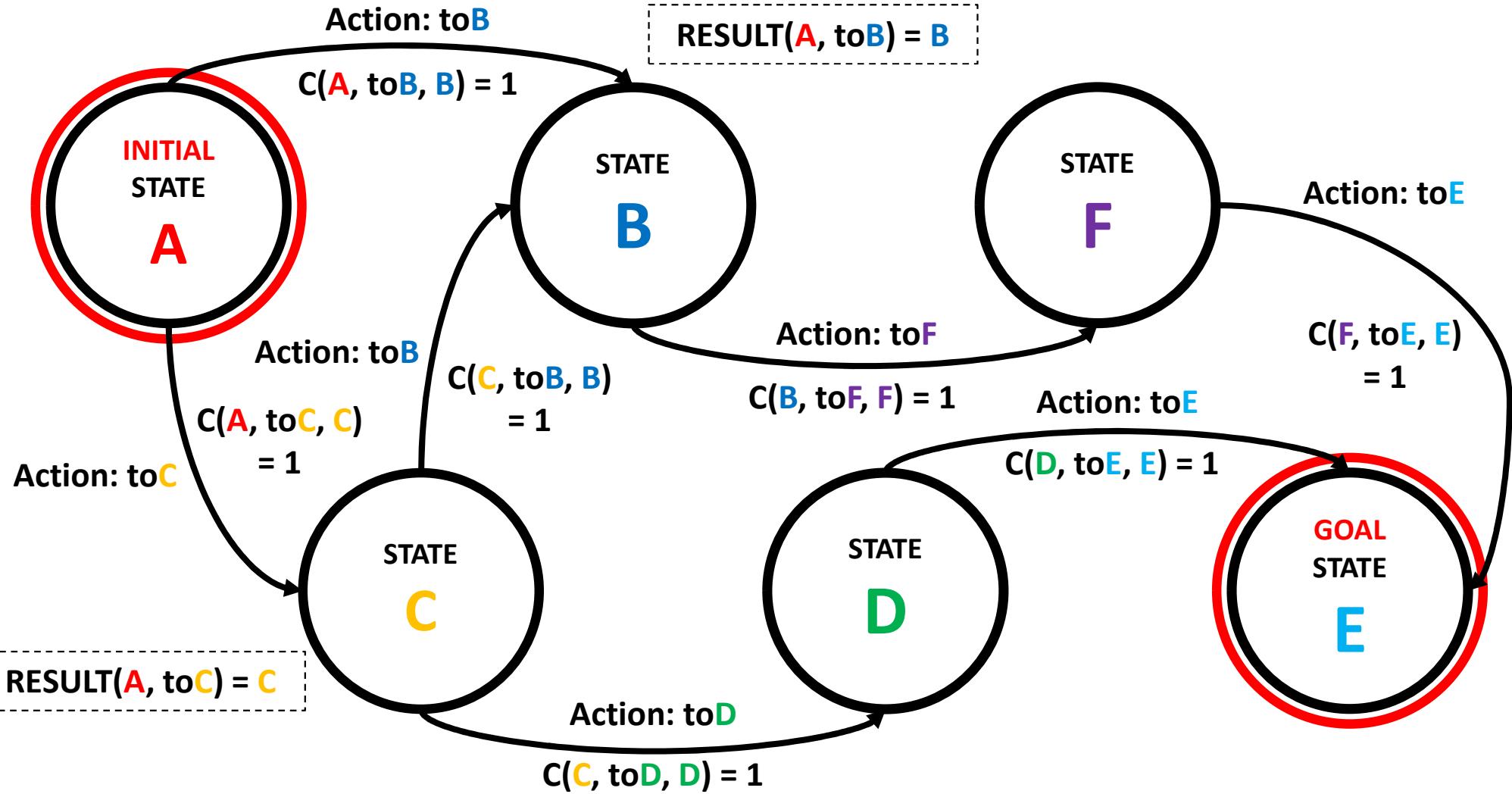
Search

State Space Model: A Graph

$\text{ACTIONS}(A) = \{\text{toB}, \text{toC}\}$

$\text{ACTIONS}(B) = \{\text{toF}\}$

$\text{ACTIONS}(F) = \{\text{toE}\}$

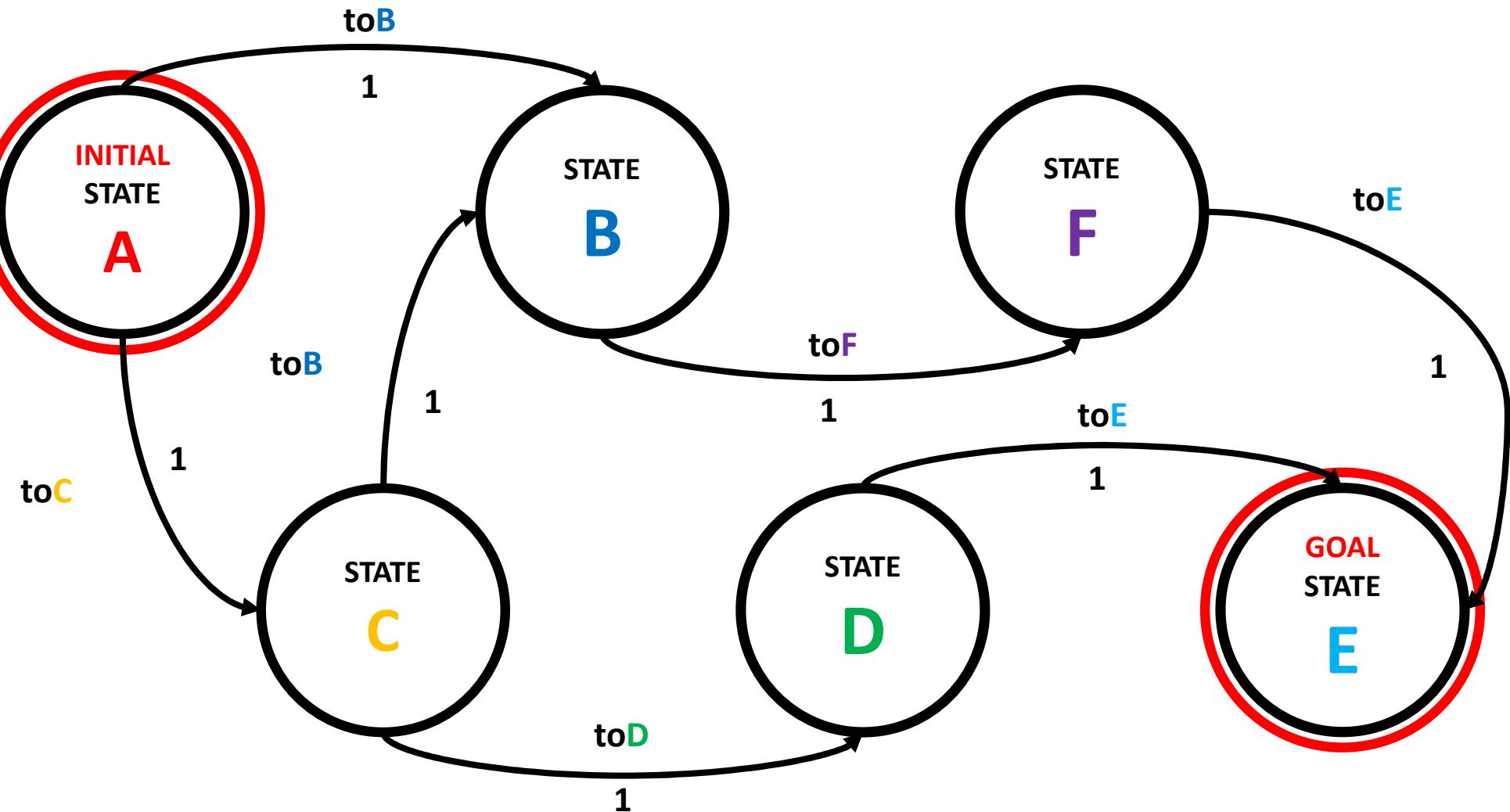


$\text{ACTIONS}(C) = \{\text{toB}, \text{toD}\}$

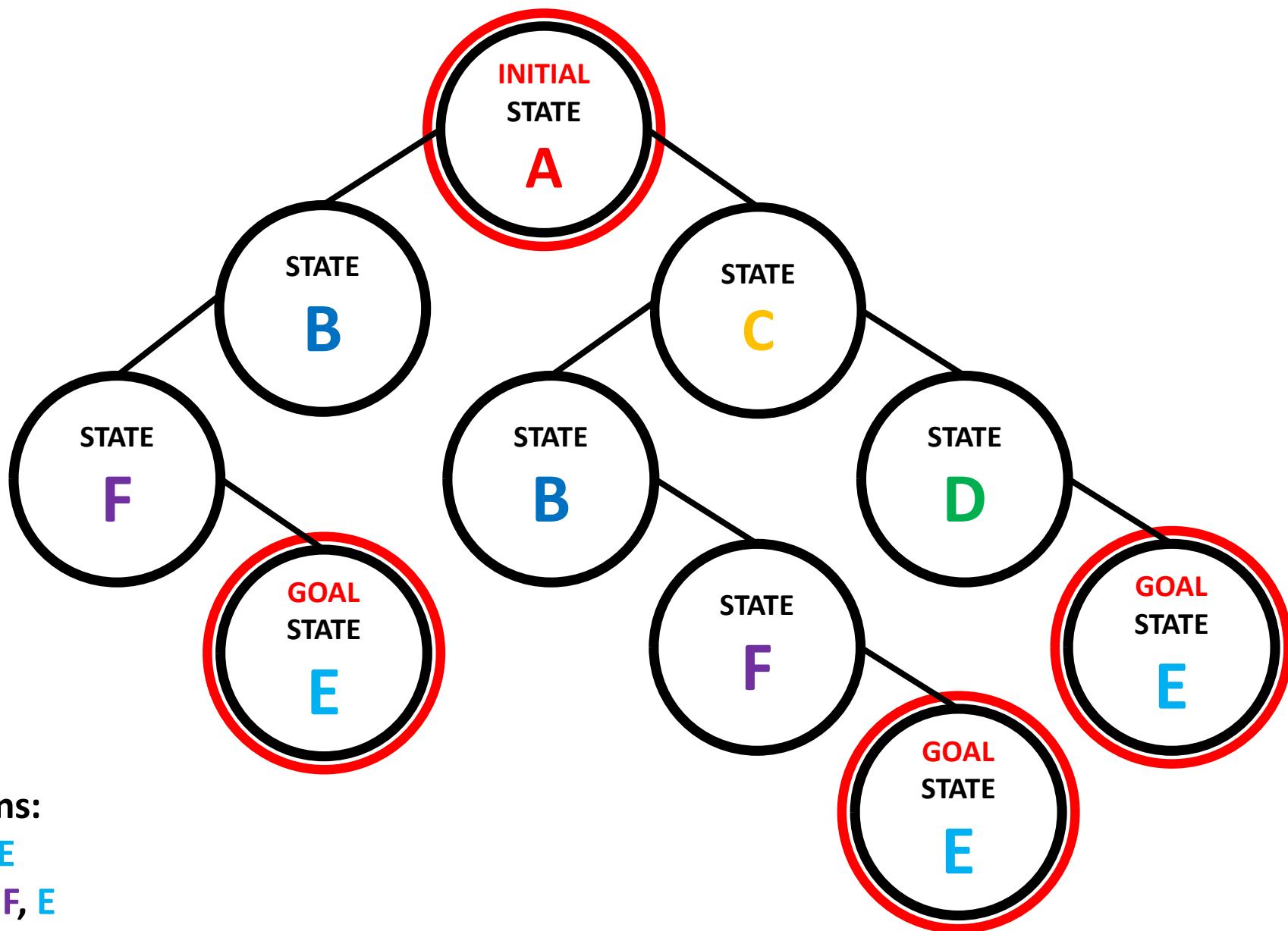
$\text{ACTIONS}(D) = \{\text{toE}\}$

$\text{ACTIONS}(E) = \emptyset$

State Space Model: A Graph



Searching State Space: Search Tree



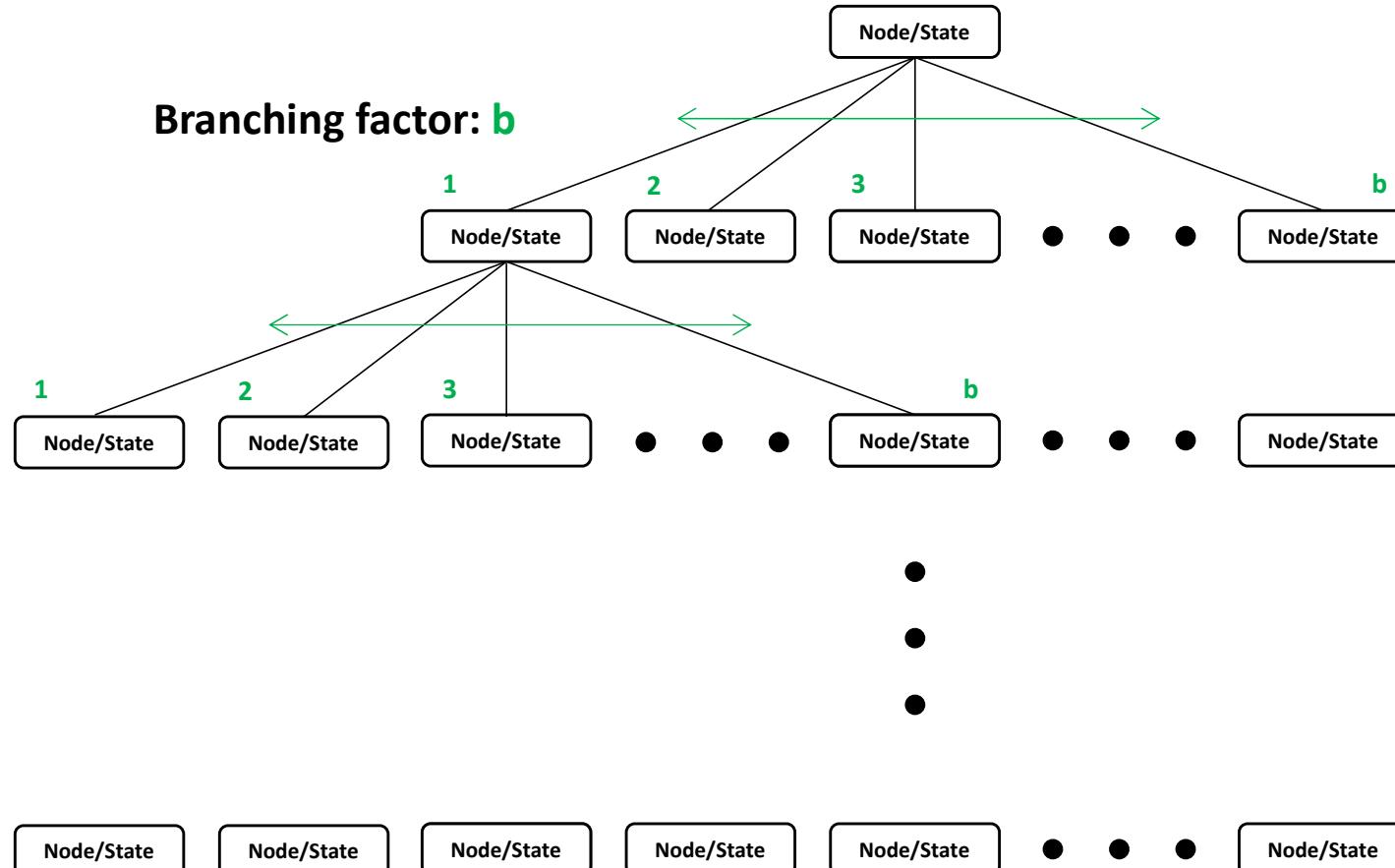
Solutions:

A, B, F, E

A, C, B, F, E

A, C, D, E

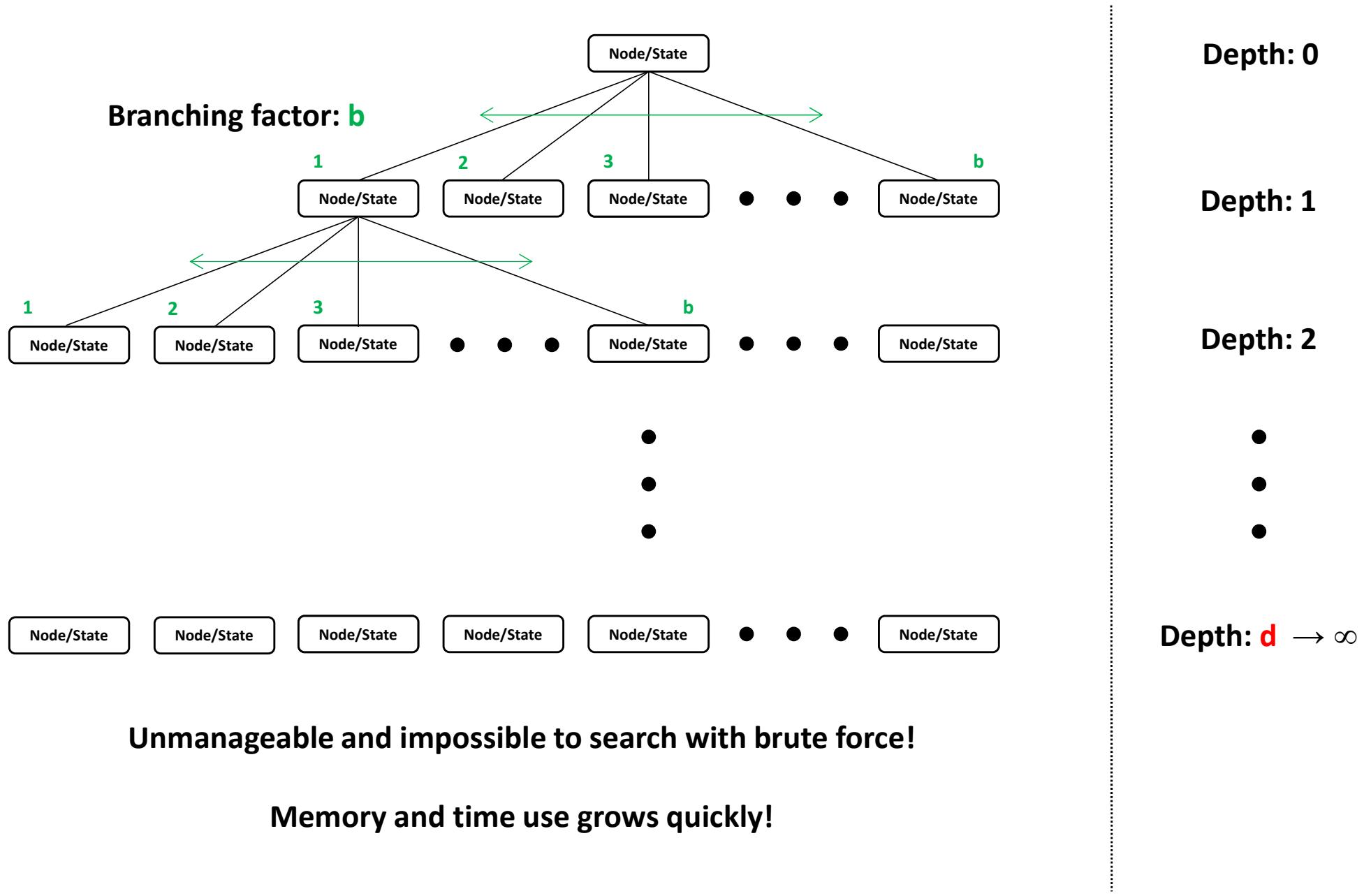
Search Tree Challenges: Size



Total number of nodes / states: $1 + b + b^2 + b^3 + \dots + b^d \rightarrow O(b^d)$

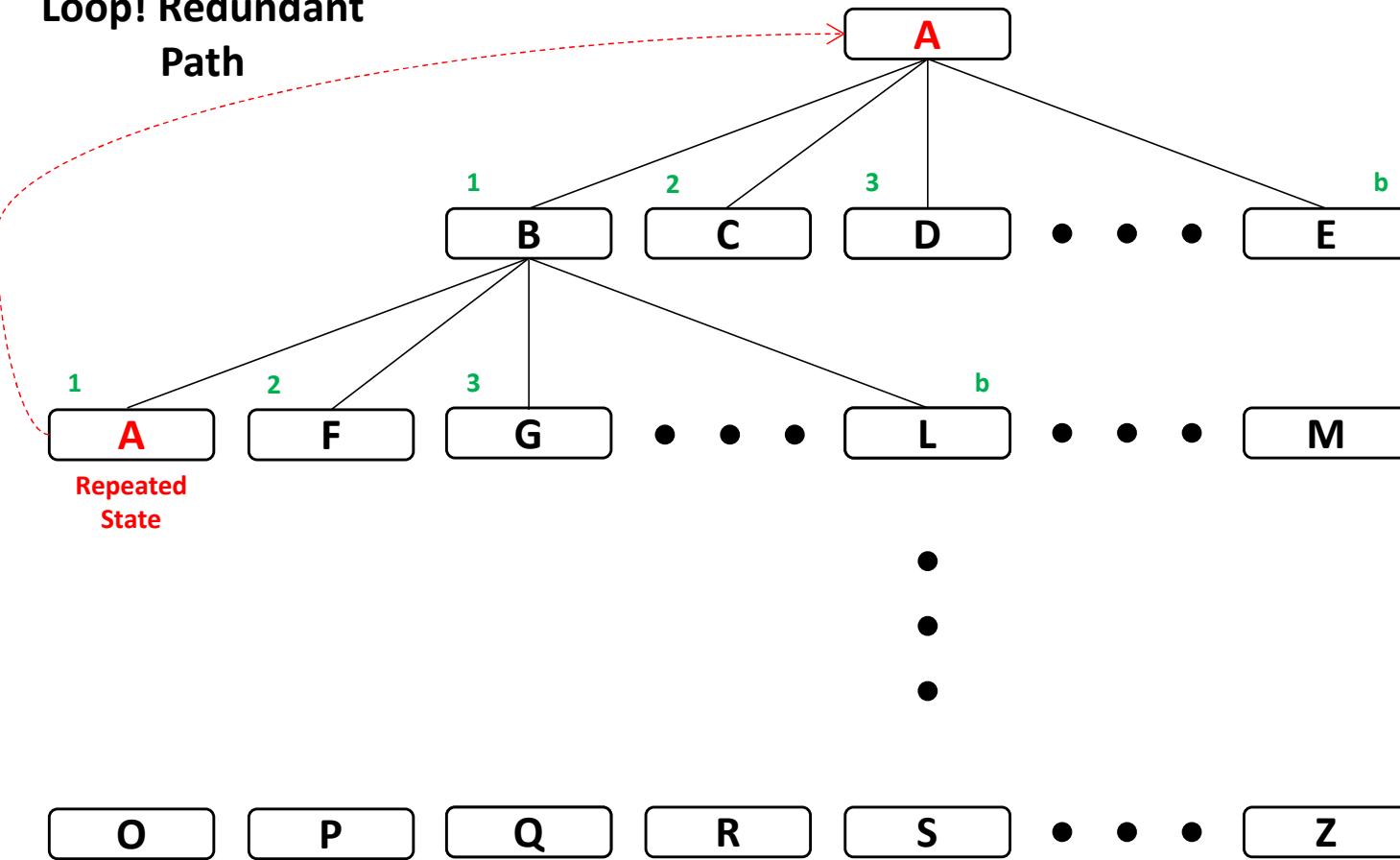
Quickly becomes unmanageable and impossible to search with brute force!

Search Tree Challenges: Infiniteness



Search Tree Challenges: Loops

Loop! Redundant Path

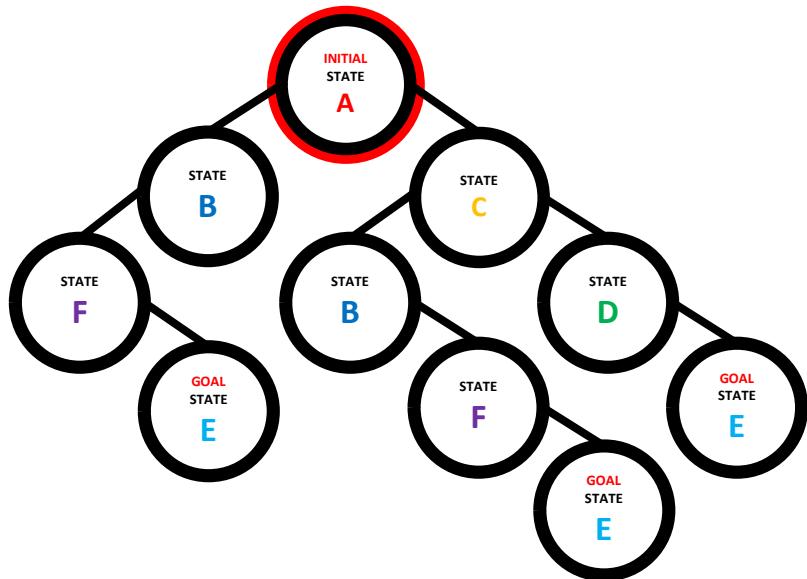


This would lead to an infinite state sequence repetition if not handled!

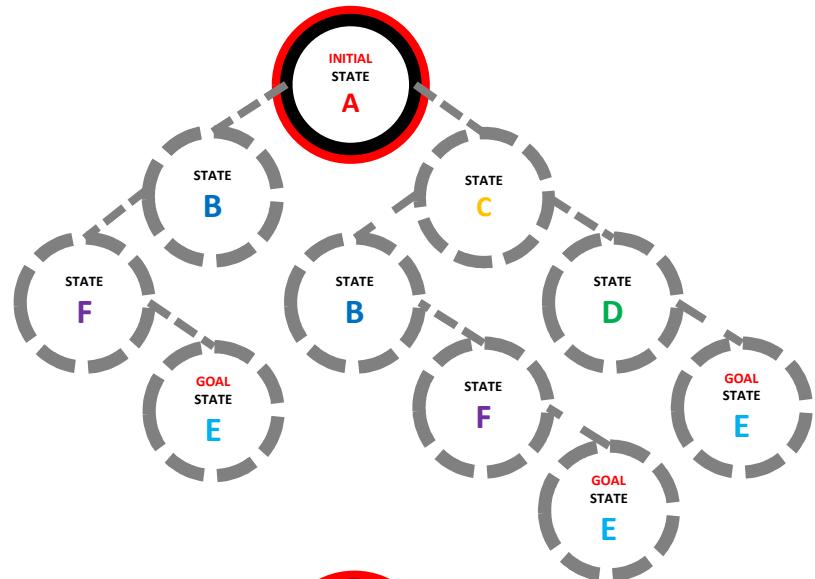
Memory and time use grows quickly!

Search Tree: Implementations

Build entire search tree

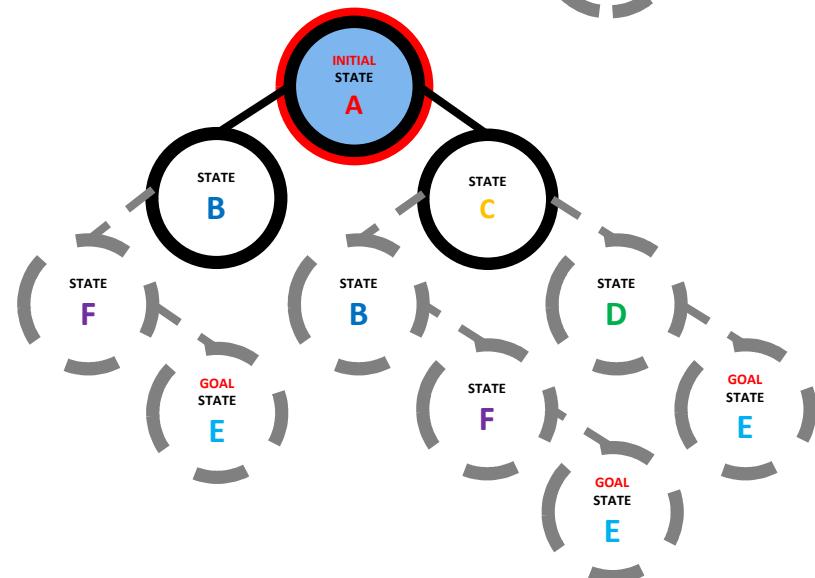


Expand/generate nodes as you go

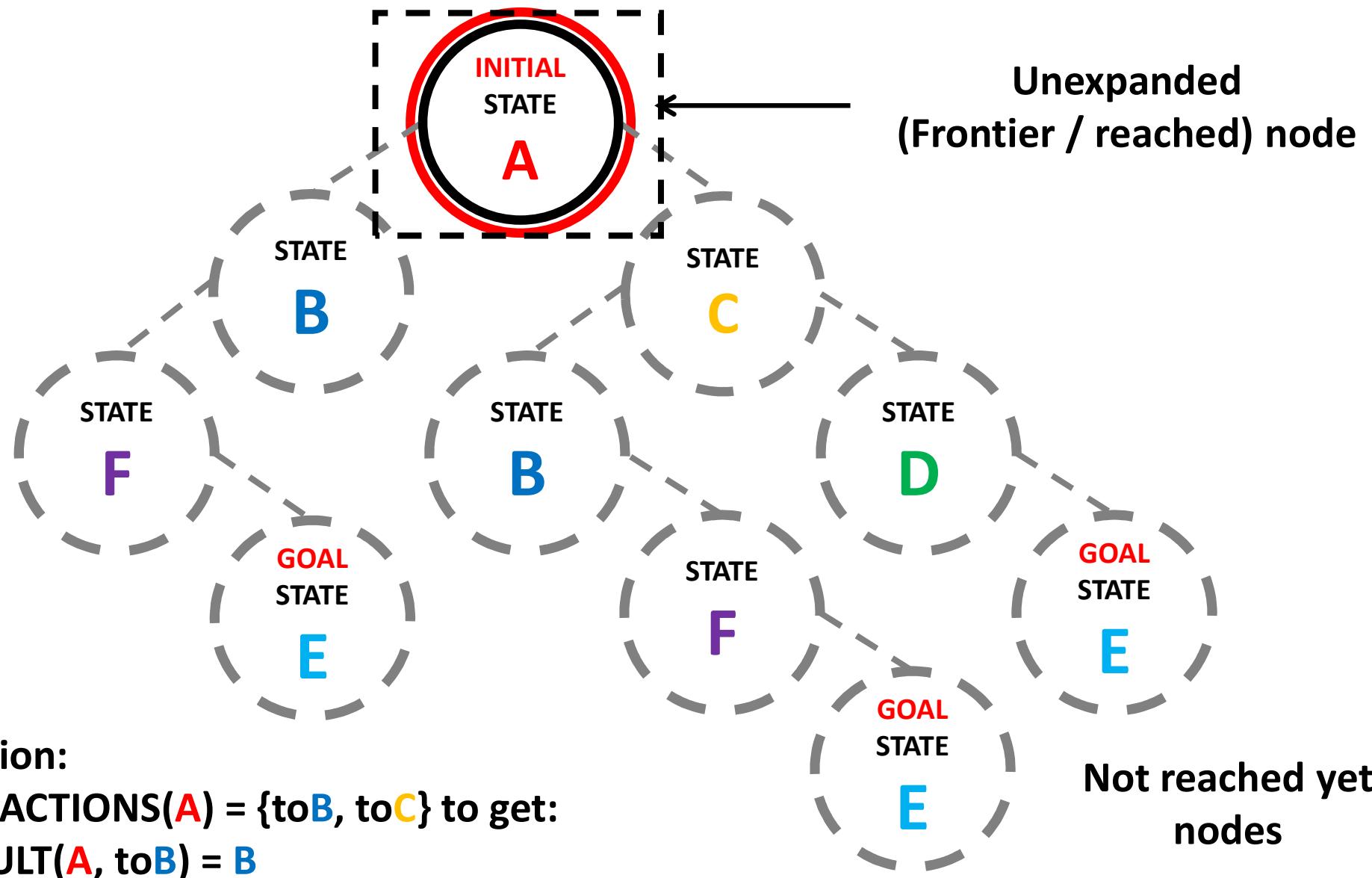


Challenges:

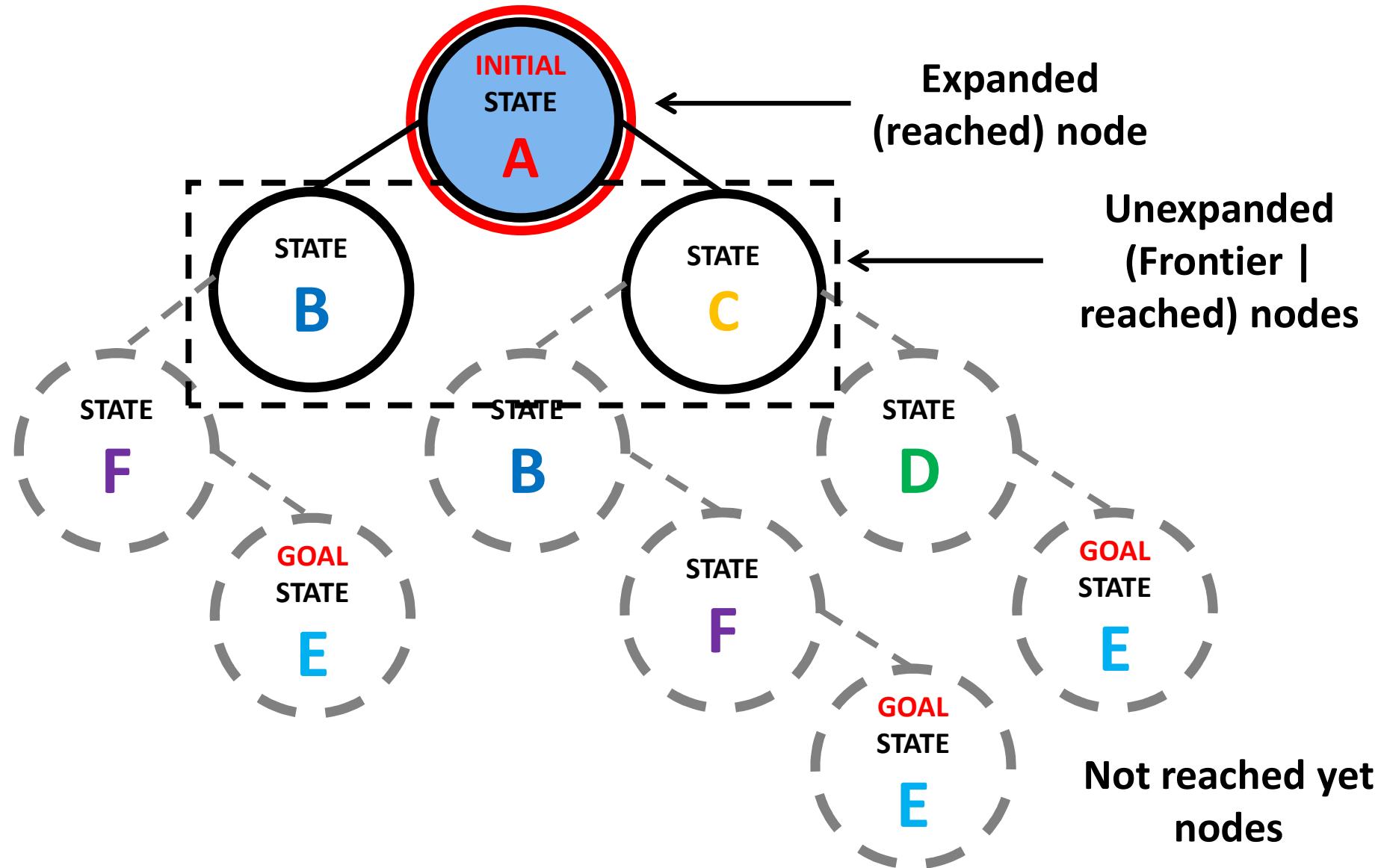
- memory requirements
- impossible for infinite number of states



Search Tree: Node Expansion



Search Tree: Node Expansion



Designing the Searching Problem

Analyze and
define the
Problem / Task

Model and build
the State Space

Select searching
algorithm

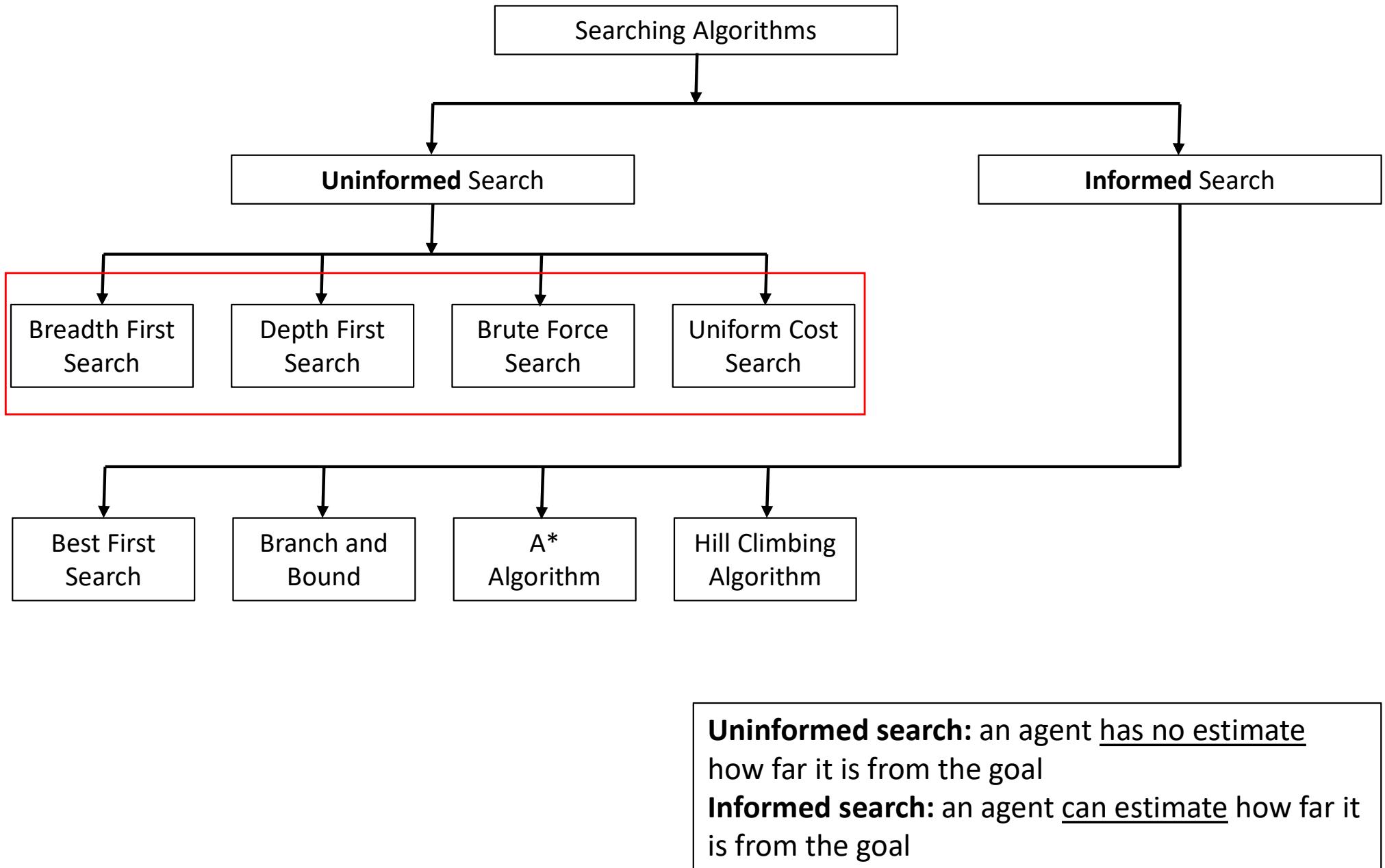
Search

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?

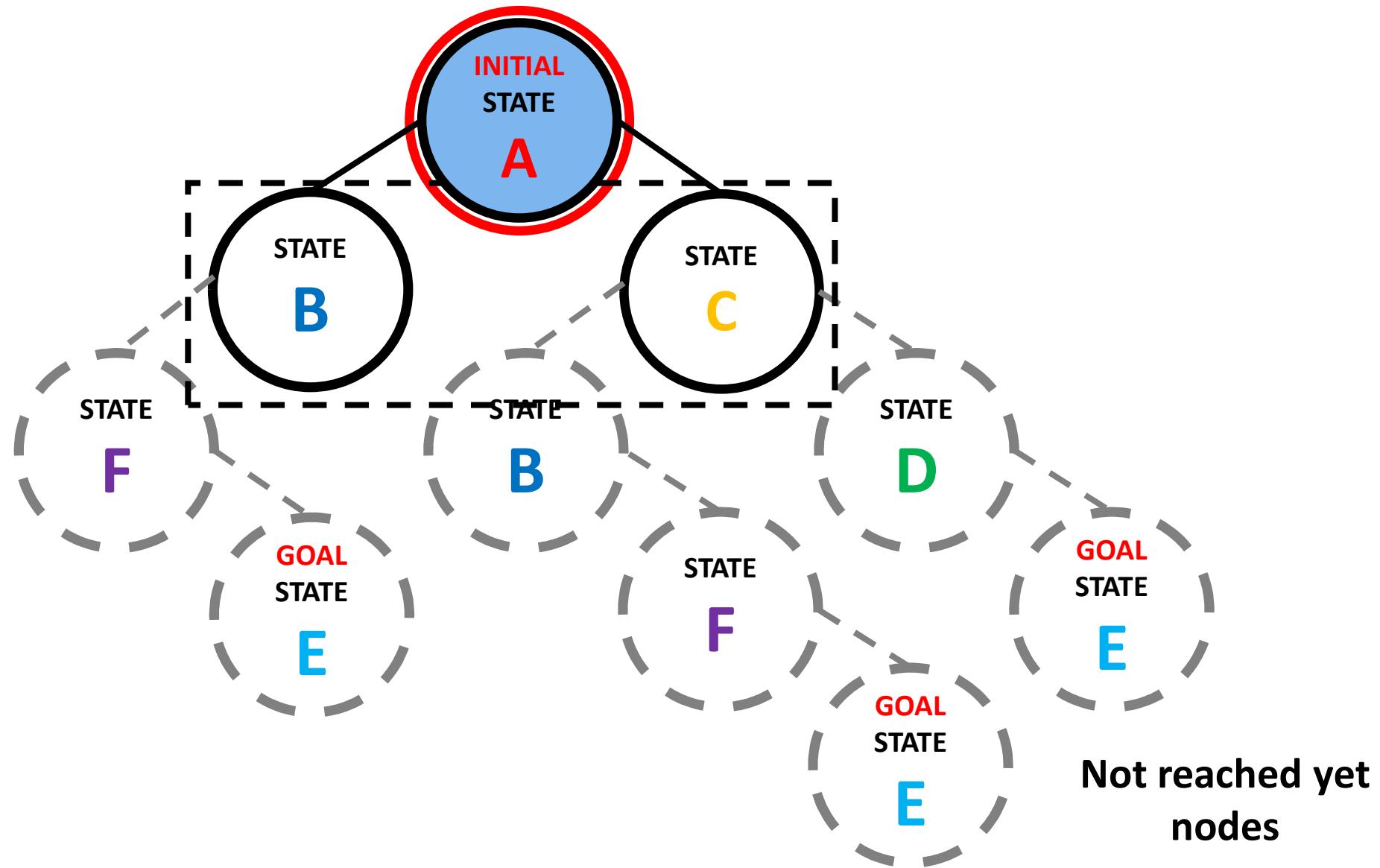
Selected Searching Algorithms



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:

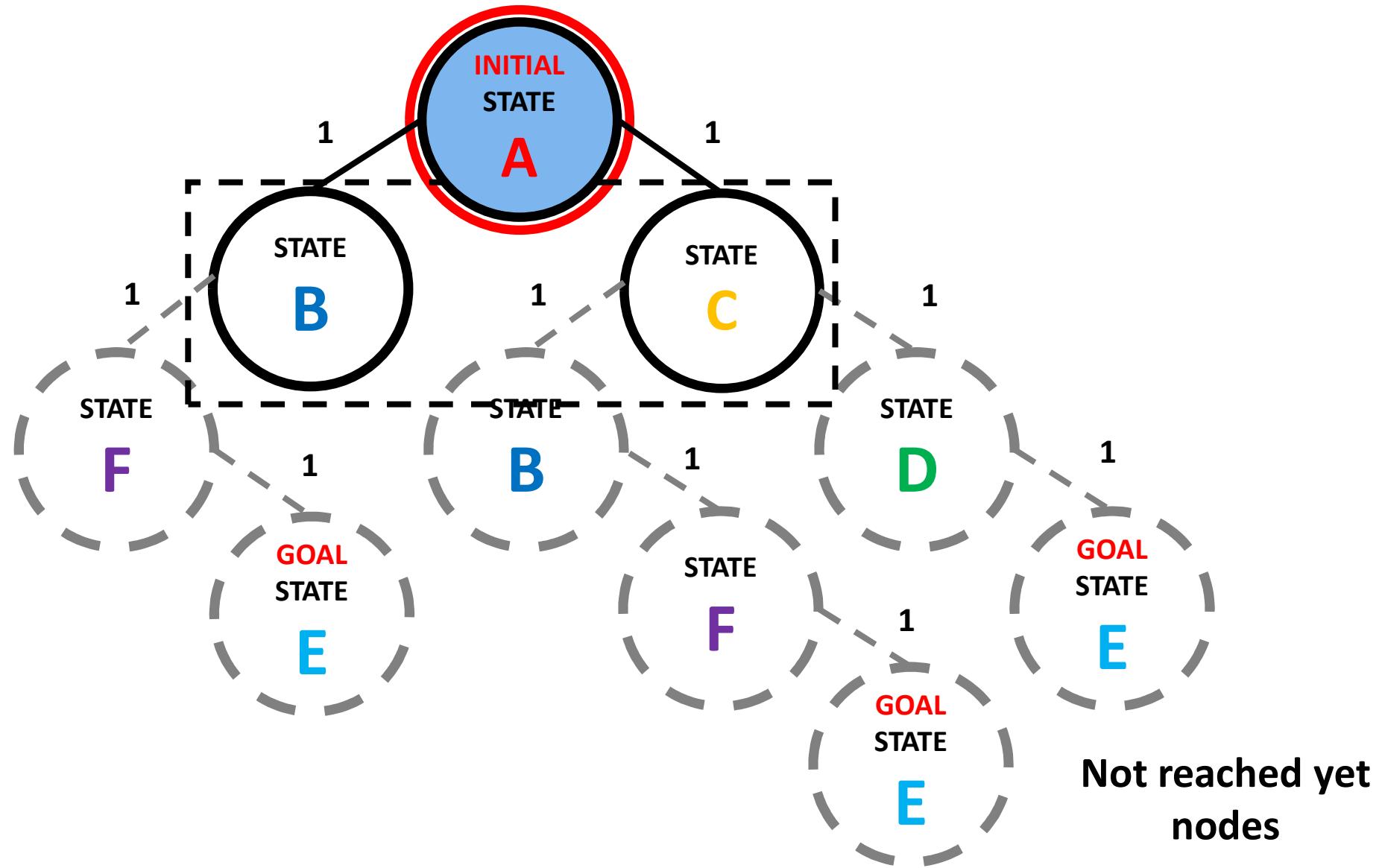
$$f(n) = f(\text{State } n)$$

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

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

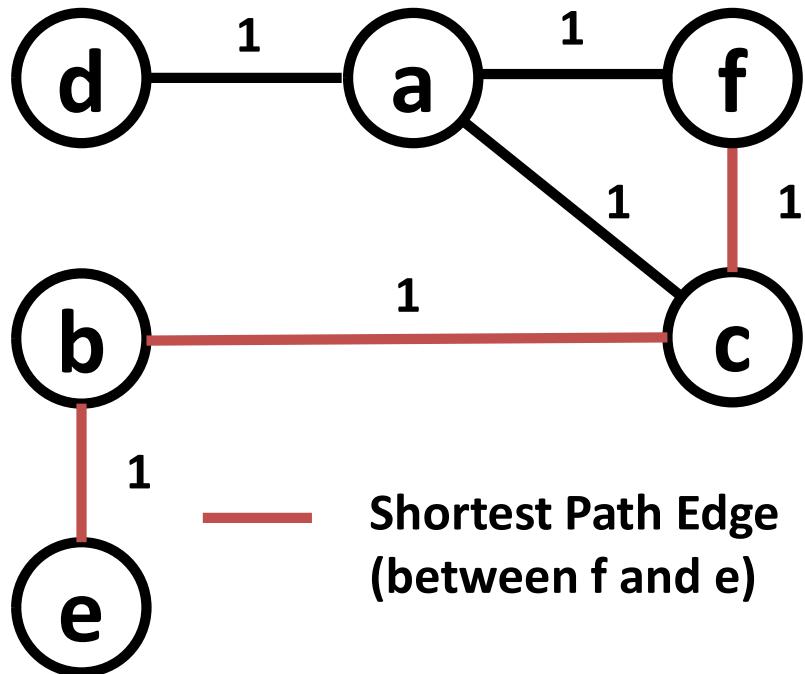
What about ties?

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:

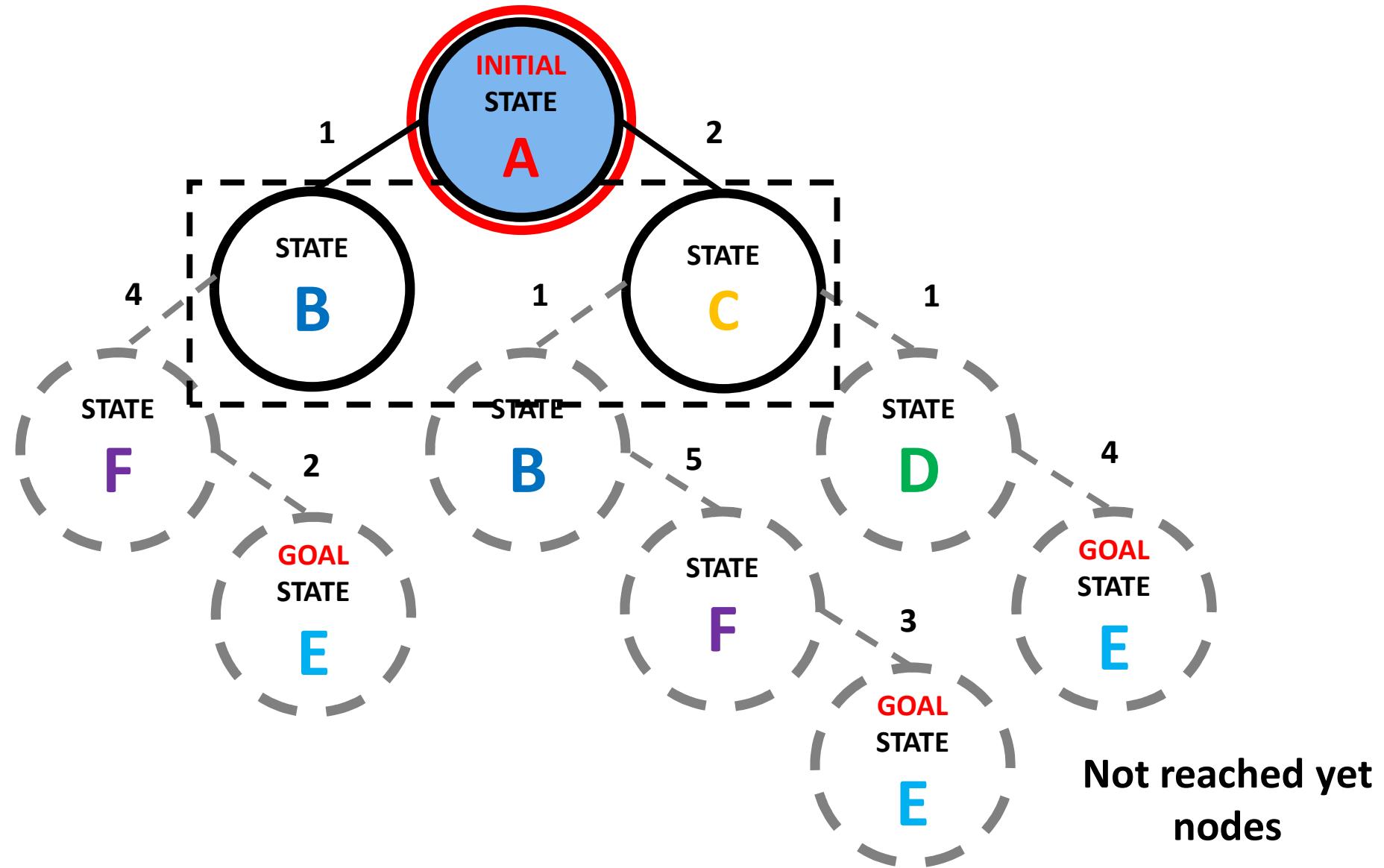
Given a weighted graph $G(V, E, w)$ and two vertices a, b in V , find the shortest path between vertices a and b (**all edge weights are equal**).

BFS and UCS: Pseudocode

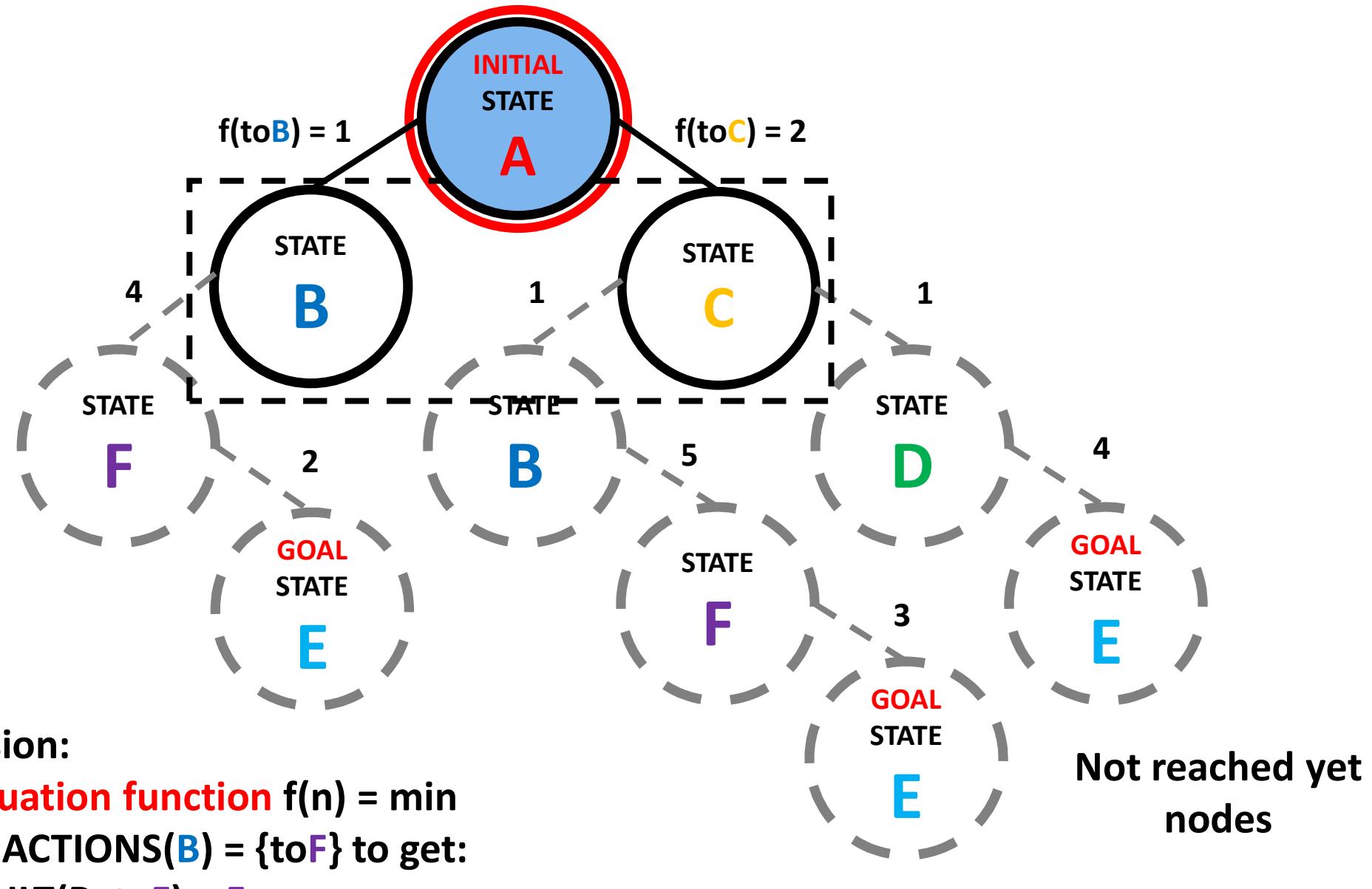
```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node  $\leftarrow$  NODE(problem.INITIAL)
  if problem.Is-GOAL(node.STATE) then return node
  frontier  $\leftarrow$  a FIFO queue, with node as an element
  reached  $\leftarrow \{problem.\text{INITIAL}\}
  while not Is-EMPTY(frontier) 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)
```

Search Tree: Variable Action Cost

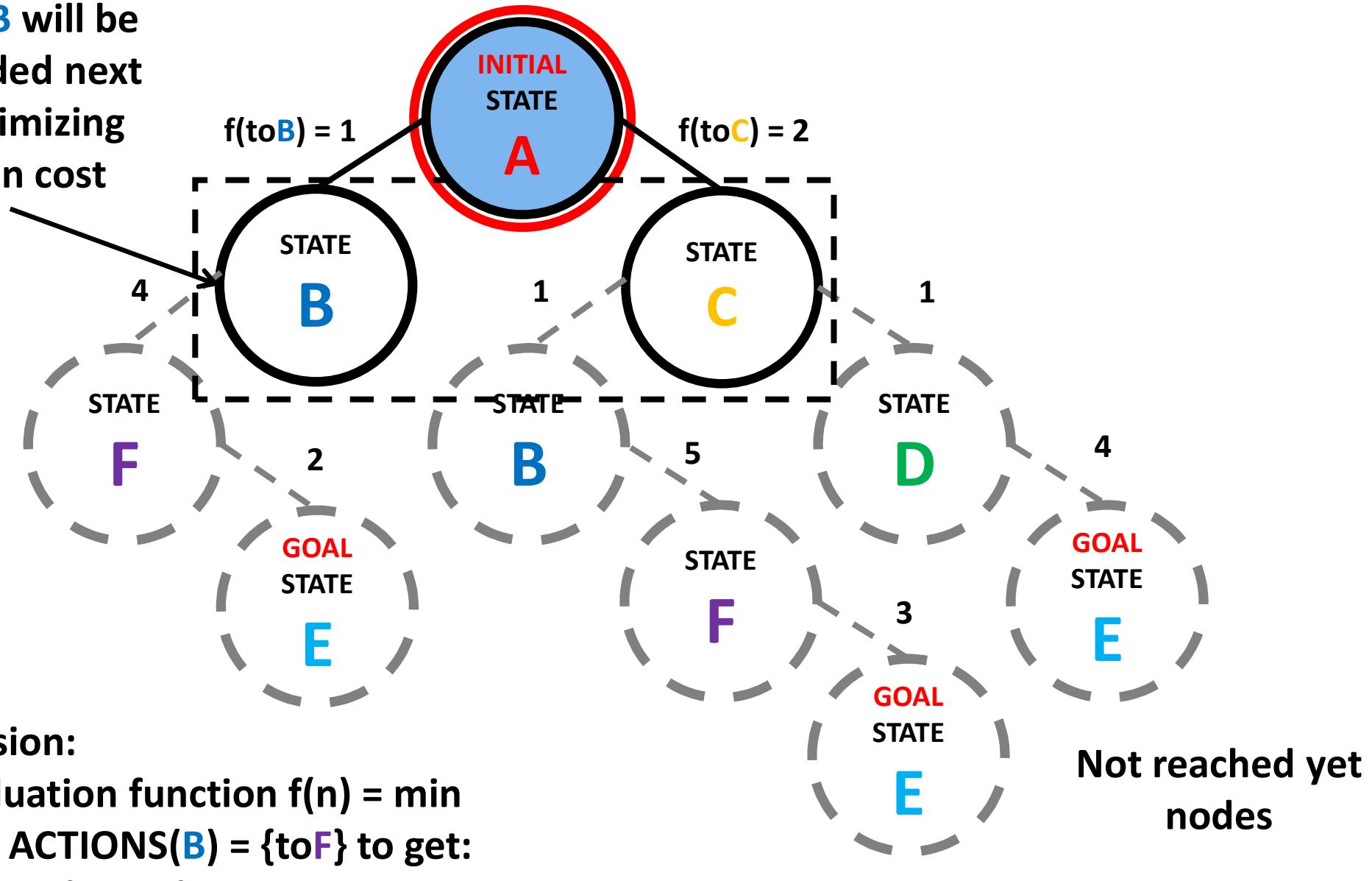


Search Tree: Variable Action Cost



Search Tree: Best-First Search

Node B will be expanded next if minimizing action cost

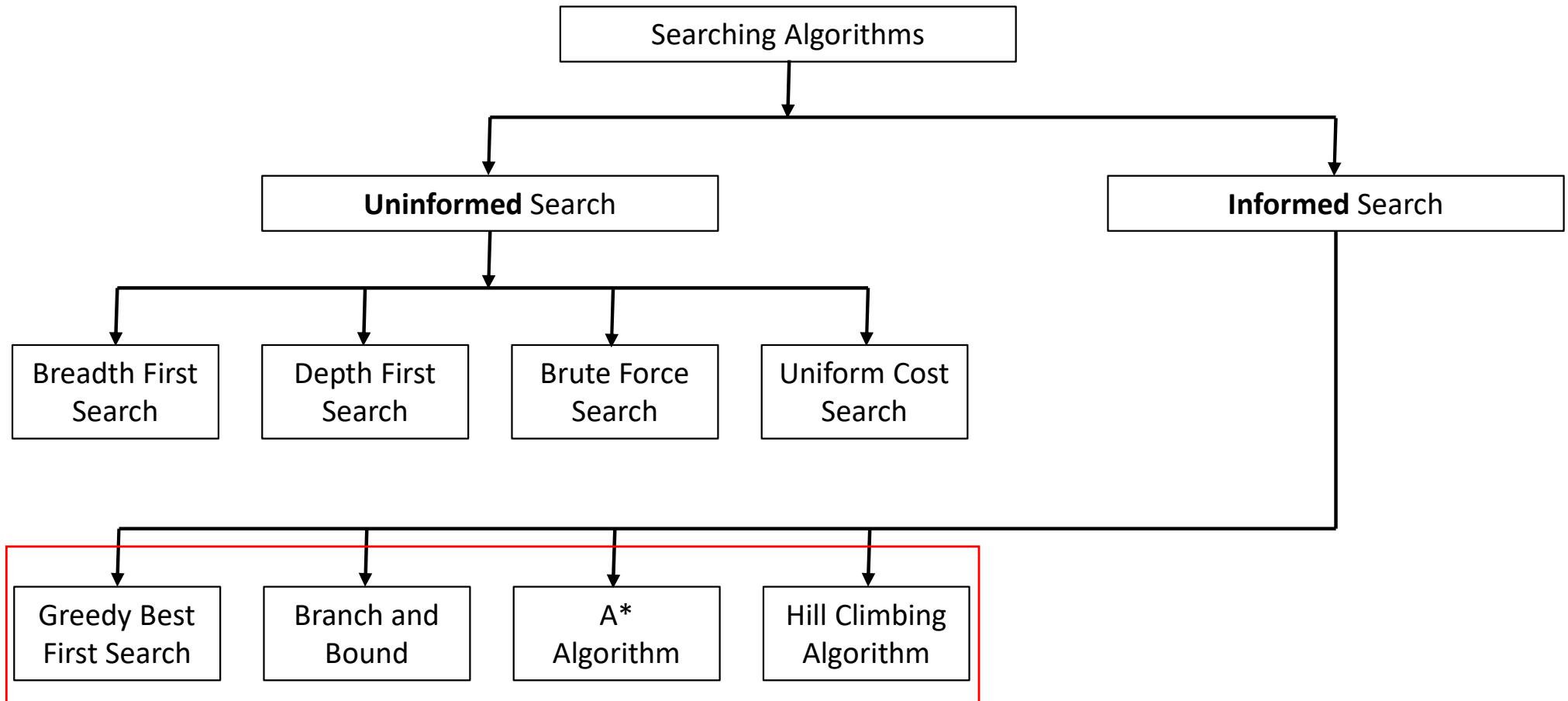


Uninformed Search Algorithms

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



Uninformed search: an agent has no estimate how far it is from the goal
Informed search: an agent can estimate how far it is from the goal

Informed Search and Heuristics

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

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

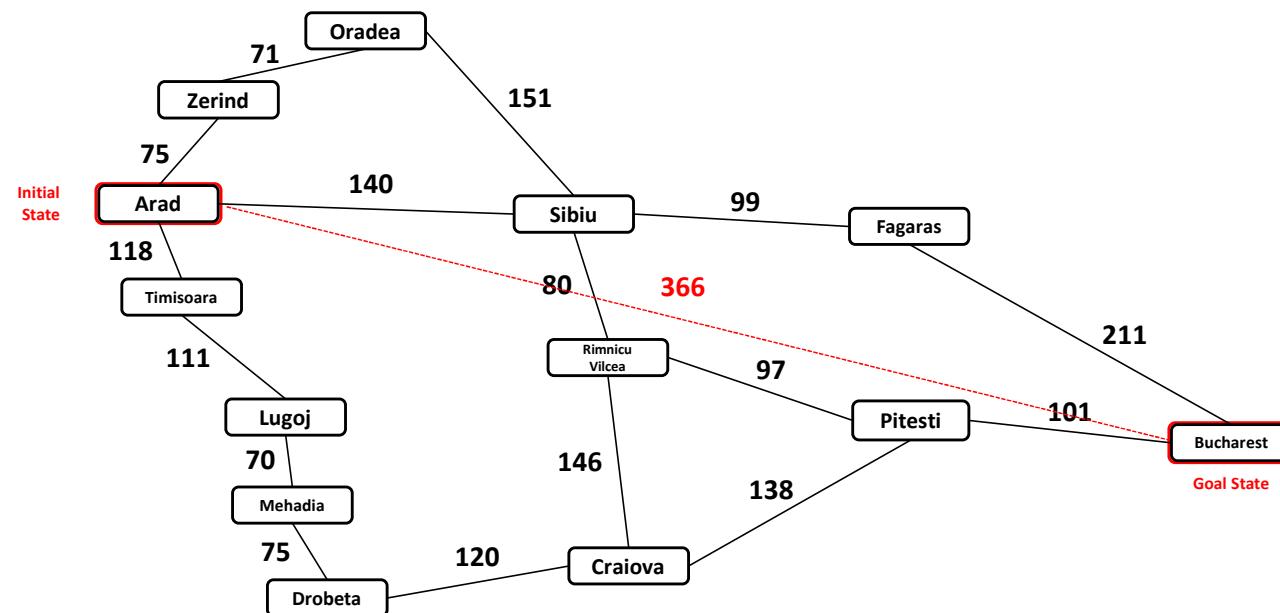
$$h(n) = n(\text{relevant information about State } n)$$

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

Romanian Roadtrip: Heuristics $h(n)$

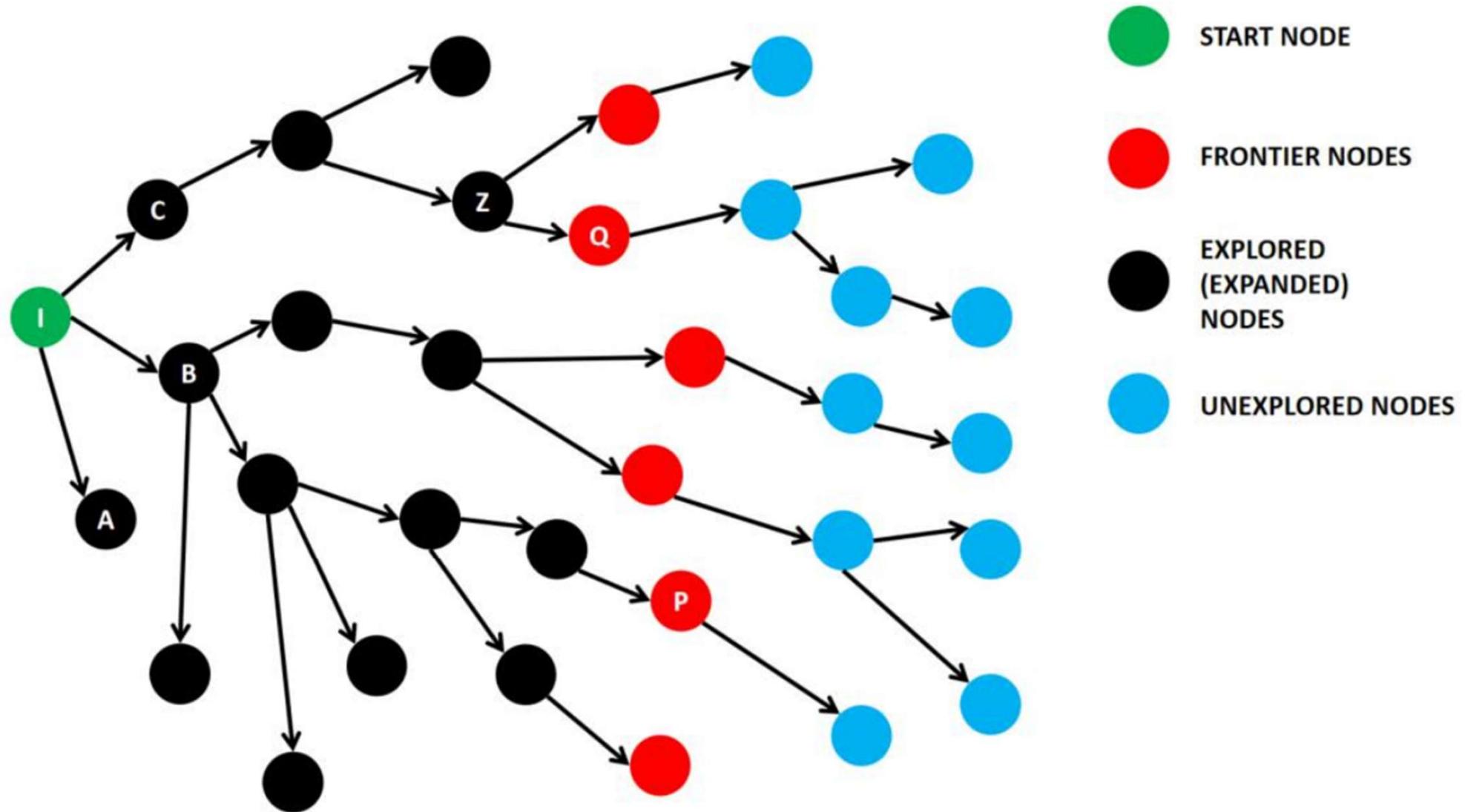
For this particular problem the heuristic function $h(n)$ is defined by a **straight-line (Euclidean) distance** between two states (cities).

“As the crows flies” in other words.



A* Algorithm

Search Tree: Frontier Nodes



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

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:

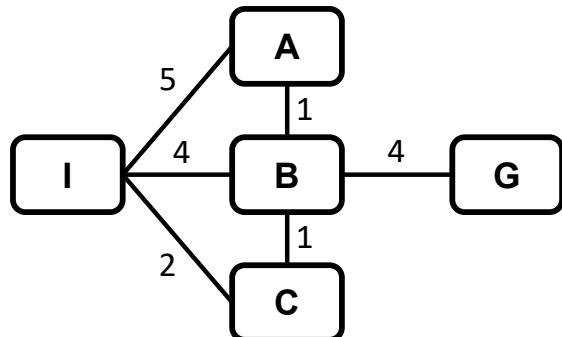
- Use the evaluation function $f(n)$ to pick next action
- Keep track of visited states
- Keep track of frontier states
- Evaluation function $f(n)$ choice controls their behavior

Best-First 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)
```

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

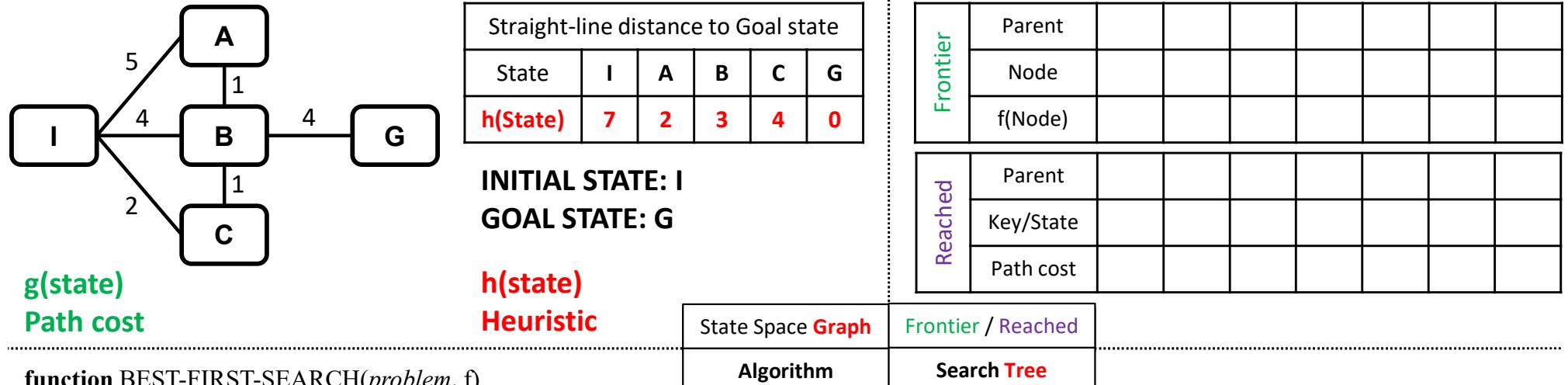
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

A* Search: Example



```
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
```

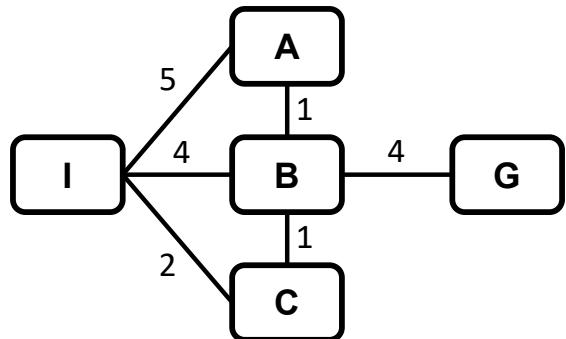
```
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
```

```
                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

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

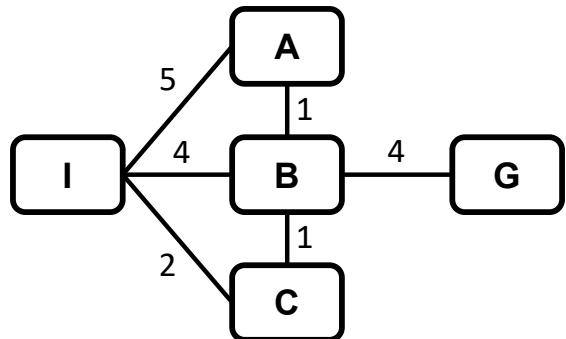
node \rightarrow

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							
$f(\text{Node})$							

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

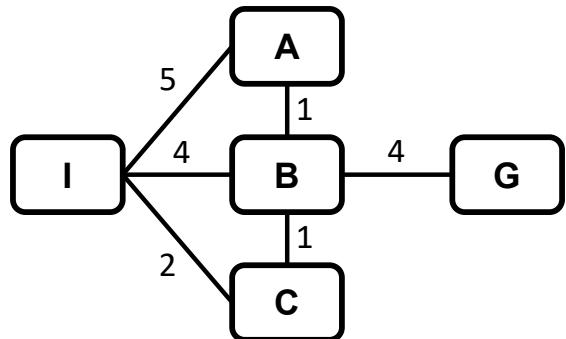


I



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					

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*

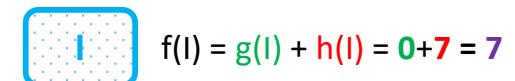
if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached[s].PATH-COST* **then**

reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

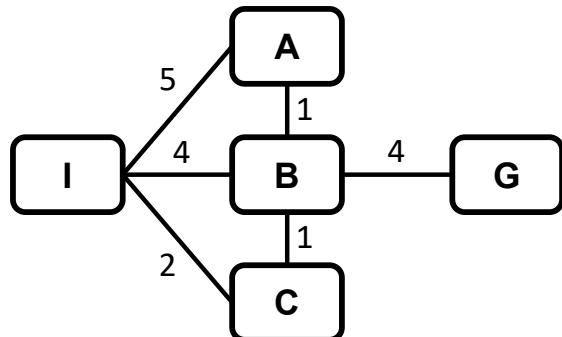


$$f(I) = g(I) + h(I) = 0 + 7 = 7$$



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

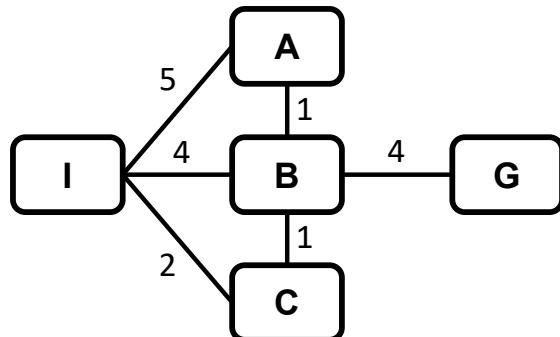
Frontier	Parent	----					
	Node	I					
	f(Node)	7					
Reached	Parent						
	Key/State						
	Path cost						

Reached	Parent						
	Key/State						
	Path cost						



Frontier node

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
$h(\text{State})$	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	I					
	$f(\text{Node})$	7					

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

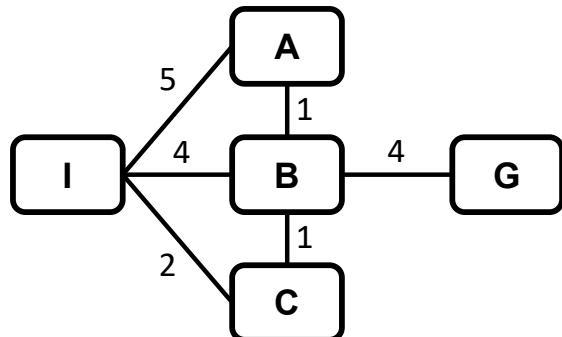


Path cost 0



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	

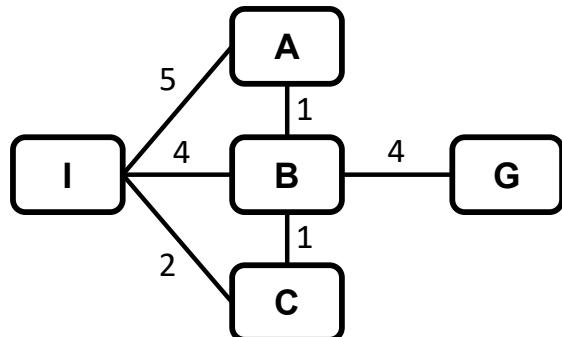
Frontier	Parent	----					
	Node	I					
	f(Node)	7					

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



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	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*

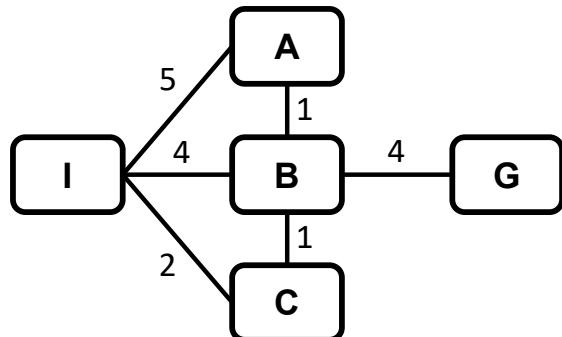
 add *child* to *frontier*

return failure



Frontier node

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*

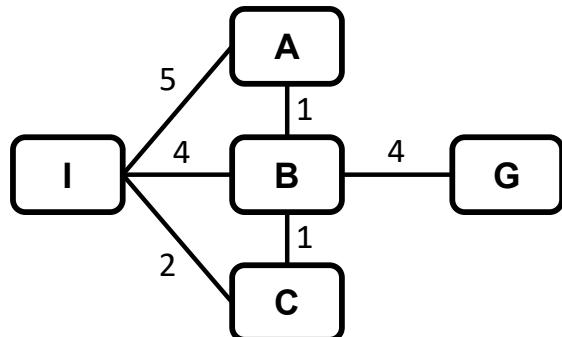
 add *child* to *frontier*

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

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

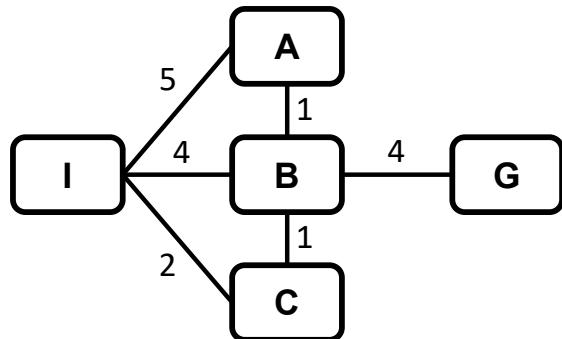
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier

  return failure
  
```

node →



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

```

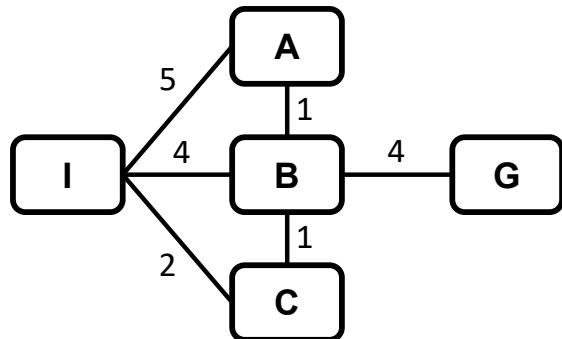
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    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 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

node \rightarrow

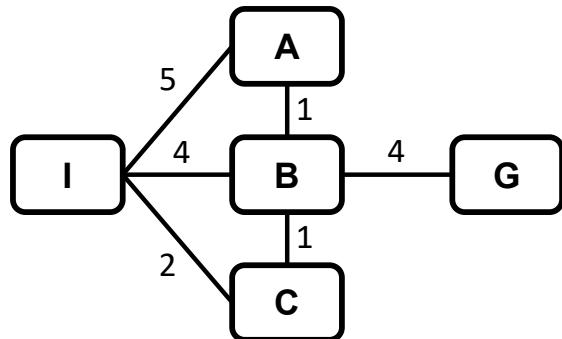


Frontier node

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**

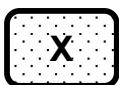
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

State Space Graph	Frontier / Reached
Algorithm	Search Tree

node \rightarrow

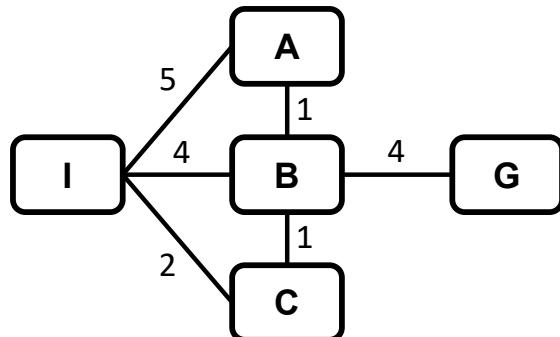


Frontier node

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**

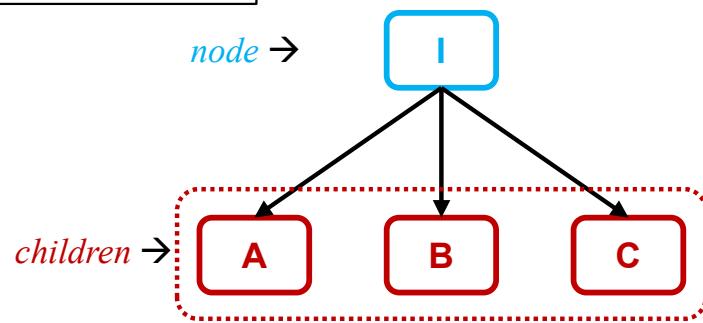
reached[s] \leftarrow *child*

 add *child* to *frontier*

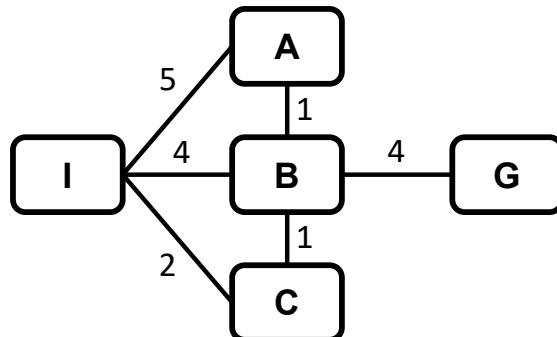
return failure

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**

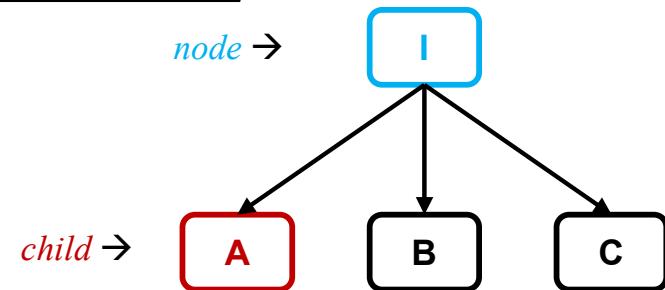
reached[s] \leftarrow *child*

 add *child* to *frontier*

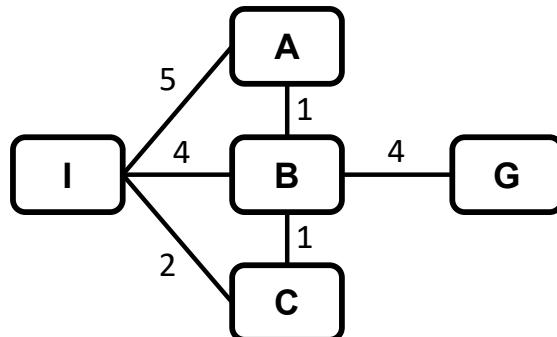
return failure

	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*

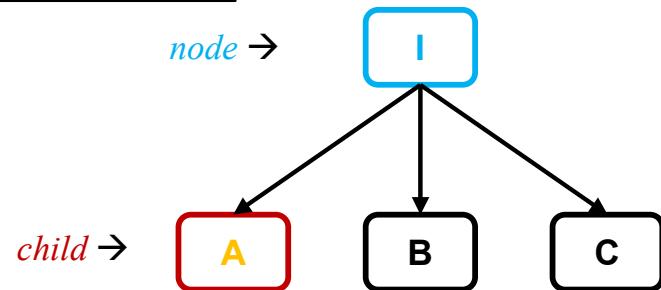
 add *child* to *frontier*

return failure

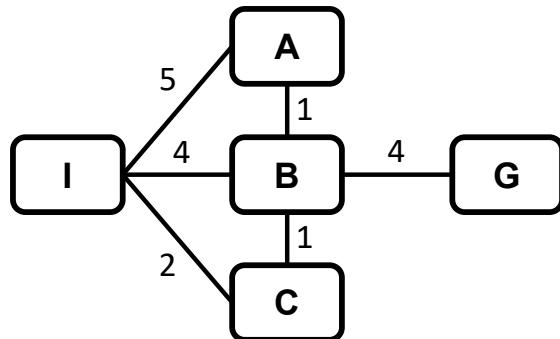
Frontier	Parent						
	Node						
	$f(\text{Node})$						

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

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*

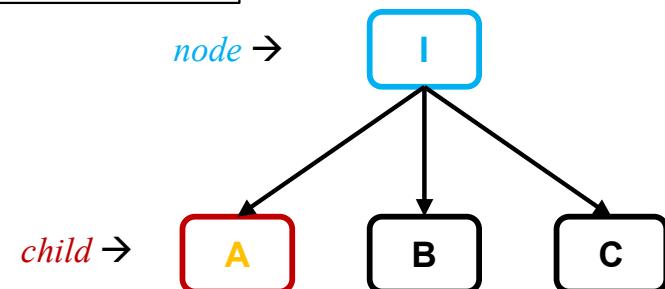
 add *child* to *frontier*

return failure

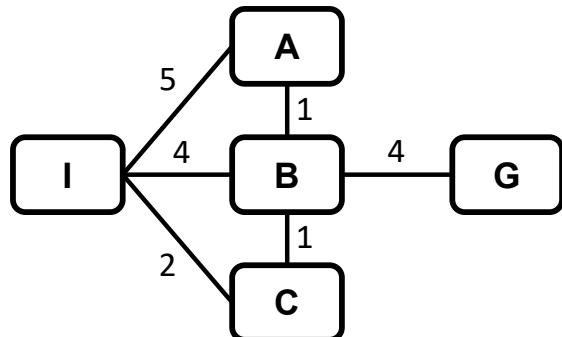
Frontier	Parent						
	Node						
	<i>f</i> (Node)						

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

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**

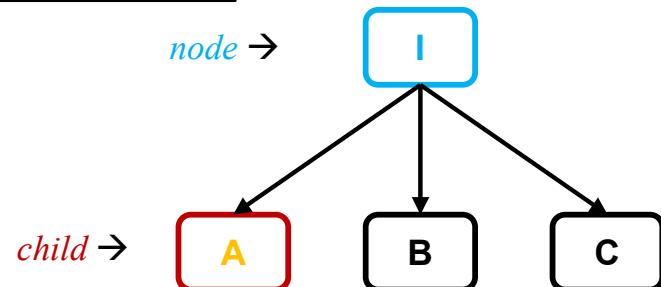
reached[s] \leftarrow *child*

 add *child* to *frontier*

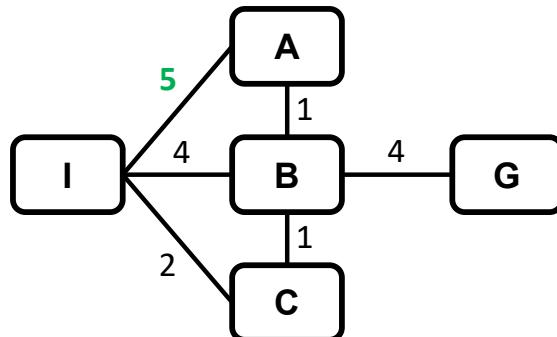
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent						
	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**

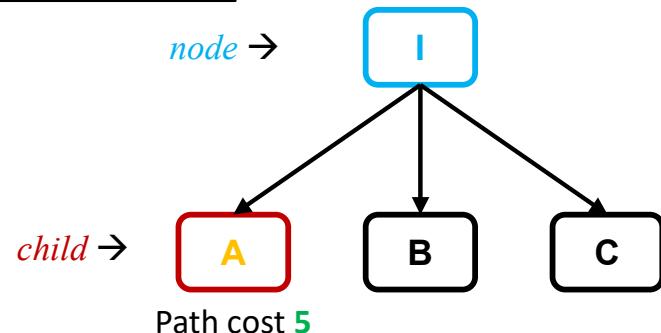
reached[s] \leftarrow *child*

 add *child* to *frontier*

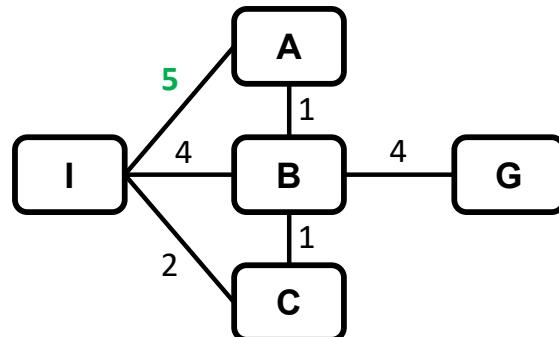
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent						
	Node						
	f(Node)						
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*

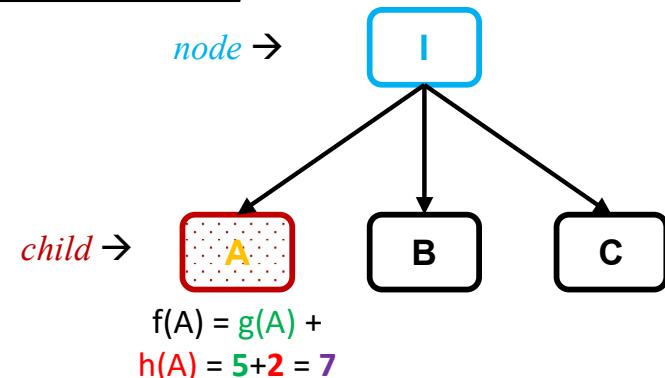
 add *child* to *frontier*

 return failure

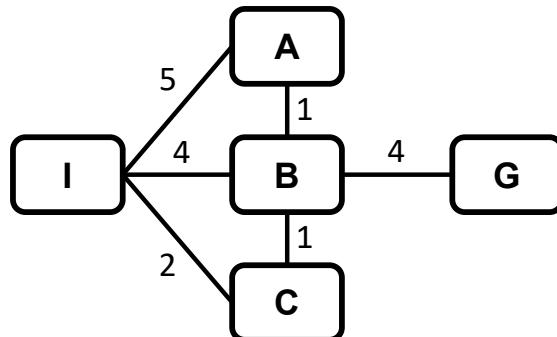
Frontier	Parent	I					
Node	A						
f(Node)	7						

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(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*

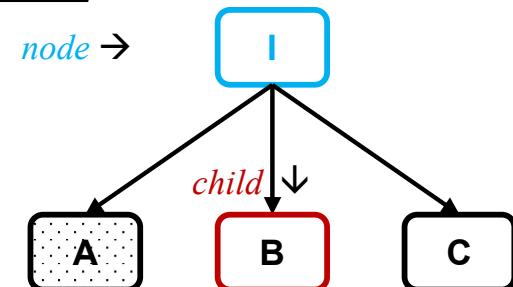
 add *child* to *frontier*

 return failure

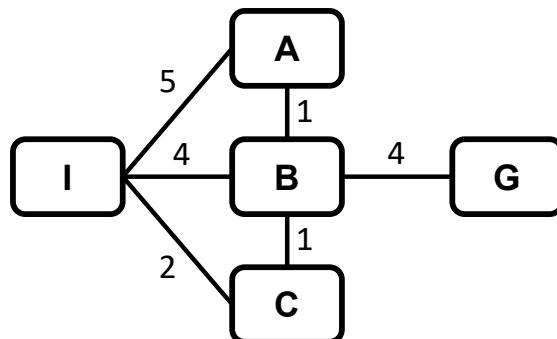
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

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*

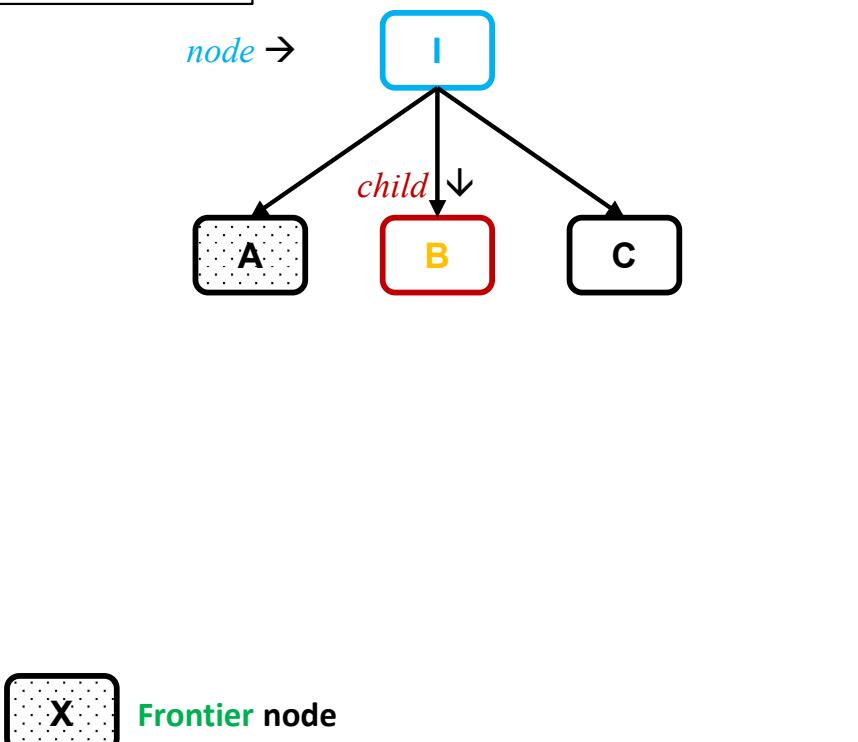
 add *child* to *frontier*

return failure

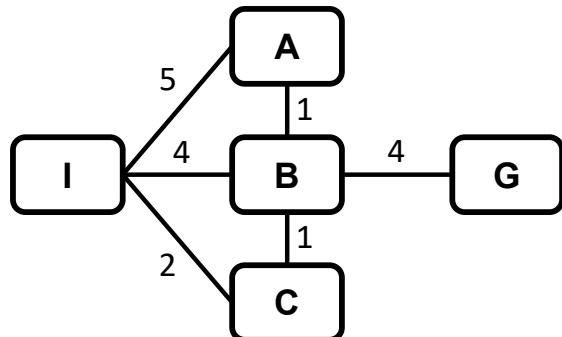
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

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*

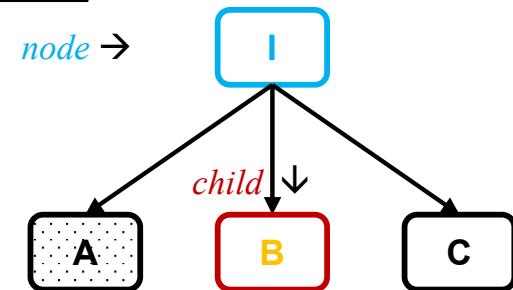
 add *child* to *frontier*

return failure

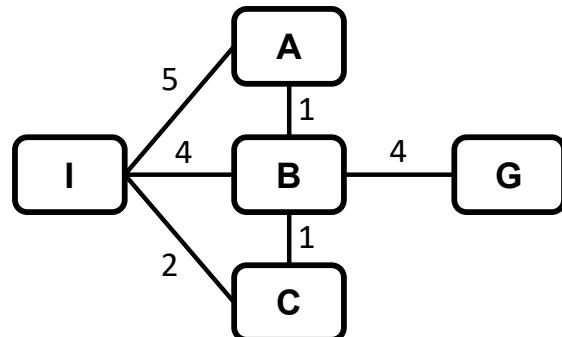
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	I					
Node	A						
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*

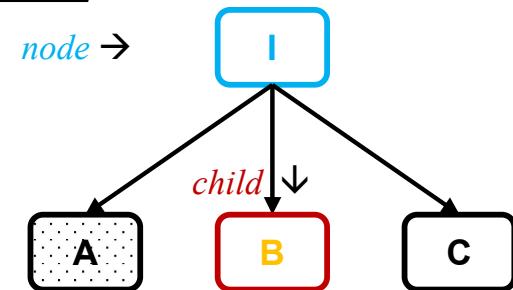
 add *child* to *frontier*

return failure

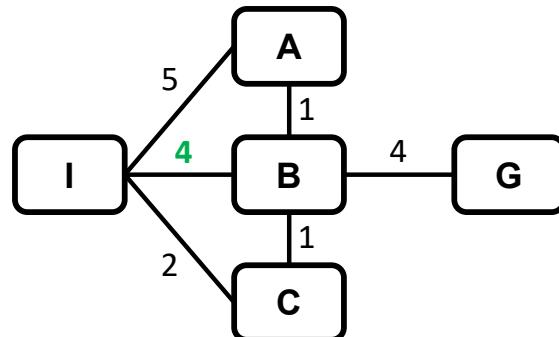
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	I					
	Node	A					
	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

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

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**

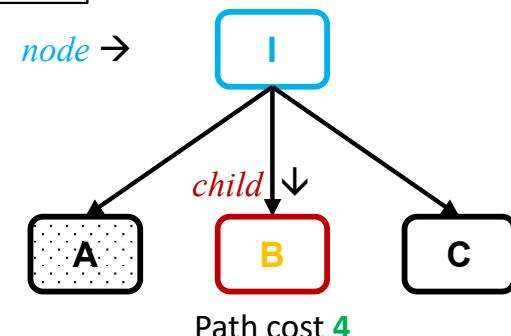
s \leftarrow *child.STATE*

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

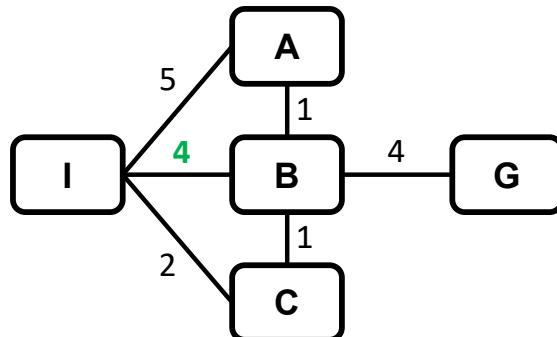
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



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*

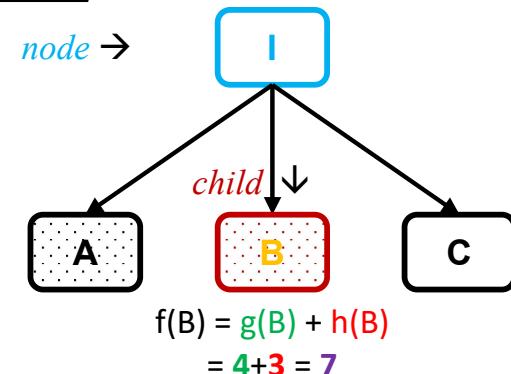
 add *child* to *frontier*

 return failure

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

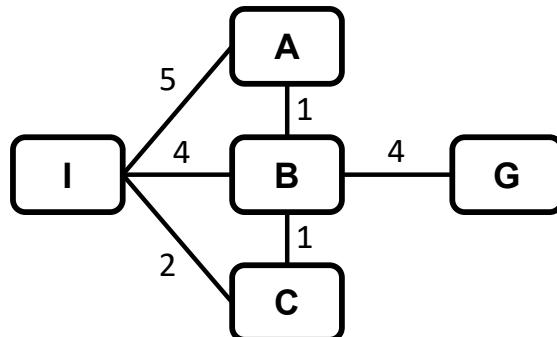
Reached	Parent	---	I	I			
	Key/State	I	A	B			
	Path cost	0	5	4			

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(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*

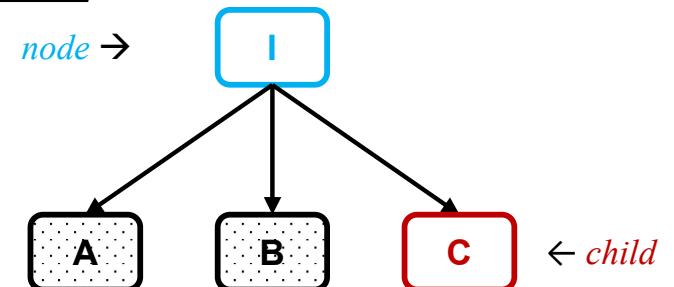
 add *child* to *frontier*

 return failure

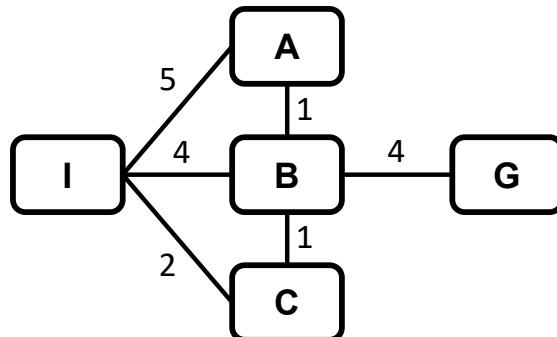
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

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**

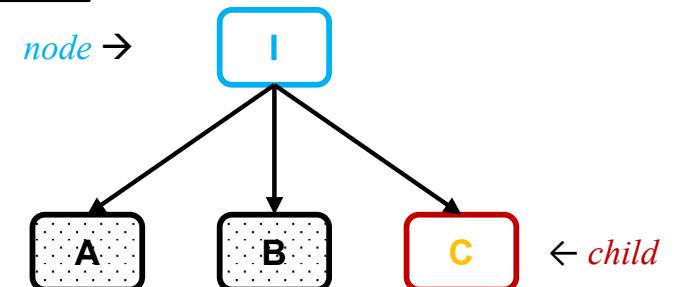
reached[s] \leftarrow *child*

 add *child* to *frontier*

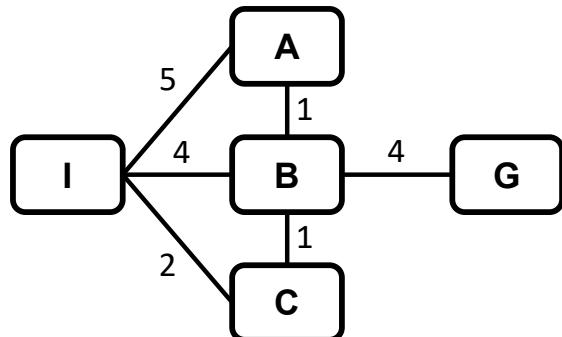
return failure

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
$h(\text{State})$	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			
	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**

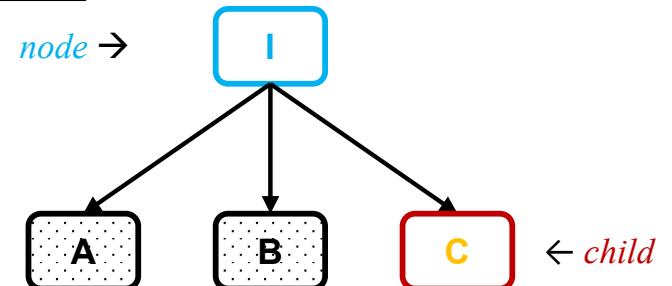
s \leftarrow *child.STATE*

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

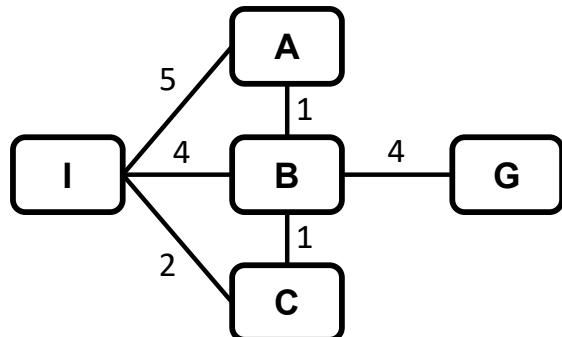
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure



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**

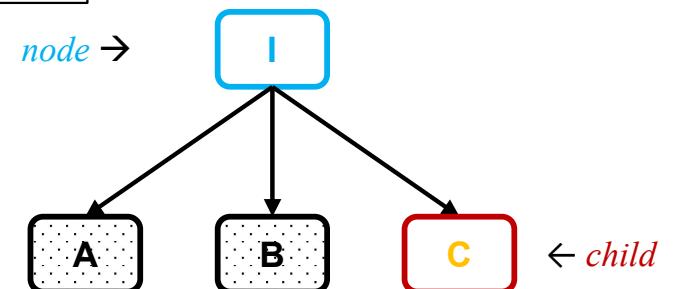
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

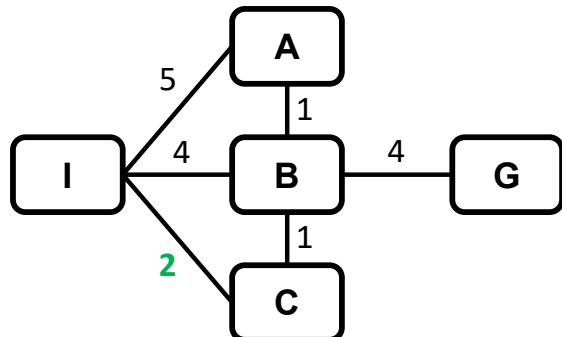
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

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			



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*

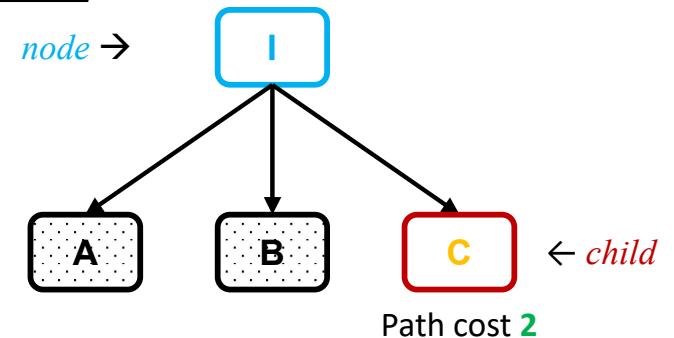
 add *child* to *frontier*

return failure

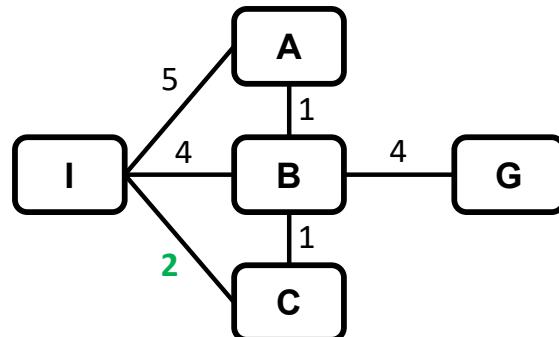
State Space Graph		Frontier / Reached			
Algorithm		Search Tree			

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

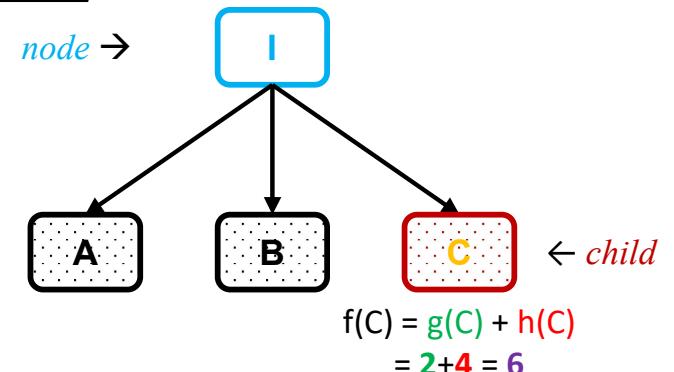
s \leftarrow *child.STATE*

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

reached[s] \leftarrow *child*

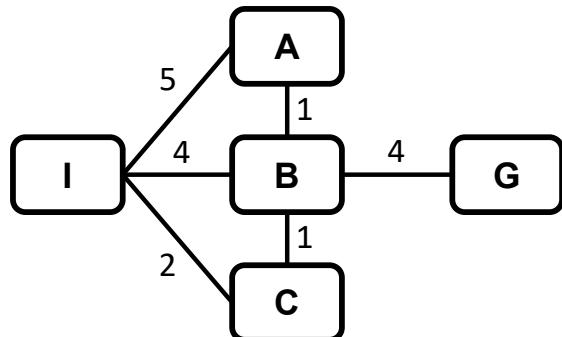
 add *child* to *frontier*

 return failure



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*

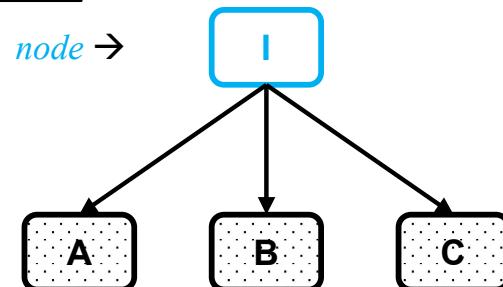
 add *child* to *frontier*

return failure

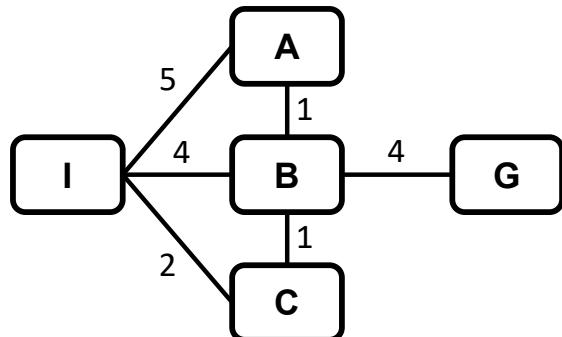
Frontier	Parent	I	I	I			
	Node	C	B	A			
	f(Node)	6	7	7			

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(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**

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*

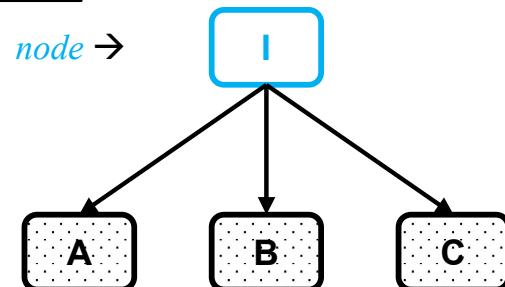
 add *child* to *frontier*

return failure

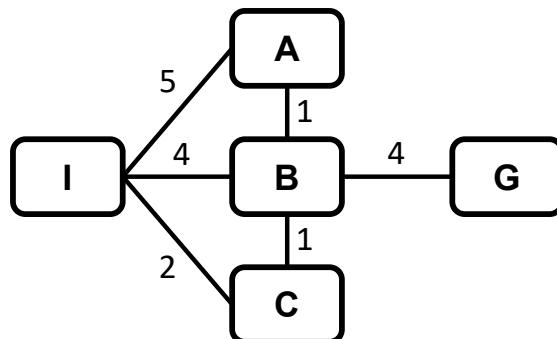
Frontier	Parent	I	I	I			
Node	C	B	A				
f(Node)	6	7	7				

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(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**

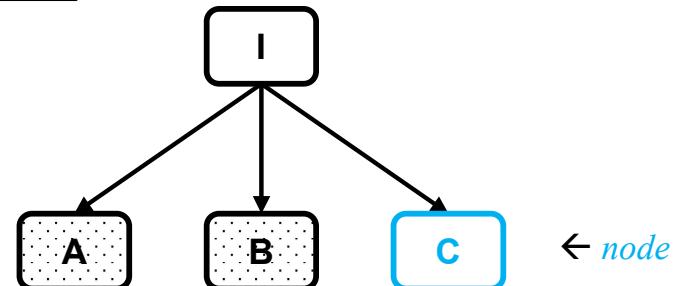
reached[s] \leftarrow *child*

 add *child* to *frontier*

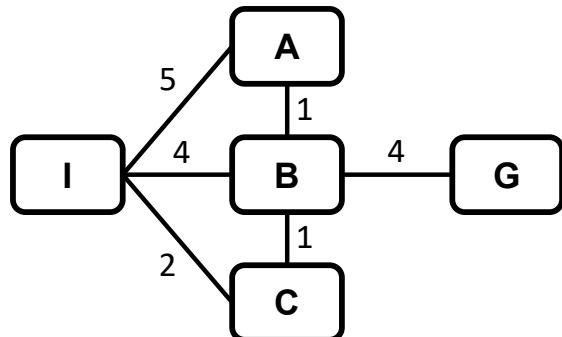
return failure

	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

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**

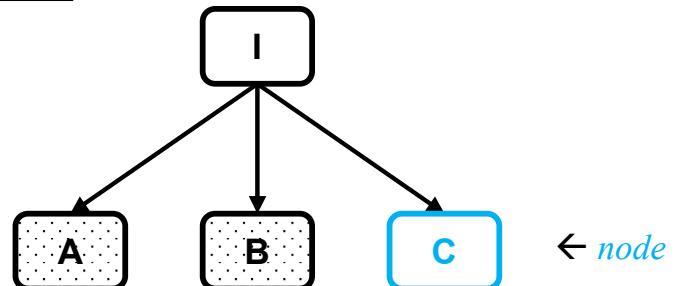
s \leftarrow *child.STATE*

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

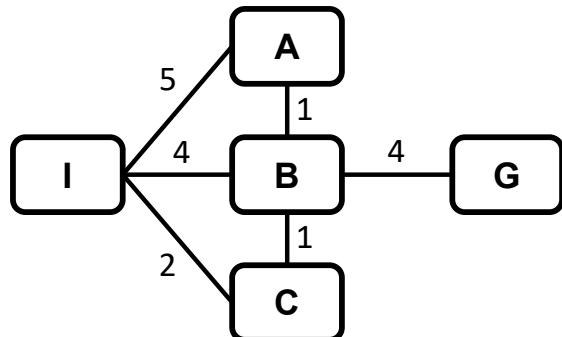
reached[s] \leftarrow *child*

 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*

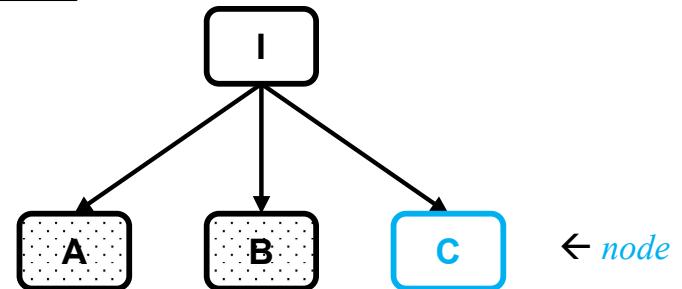
 add *child* to *frontier*

return failure

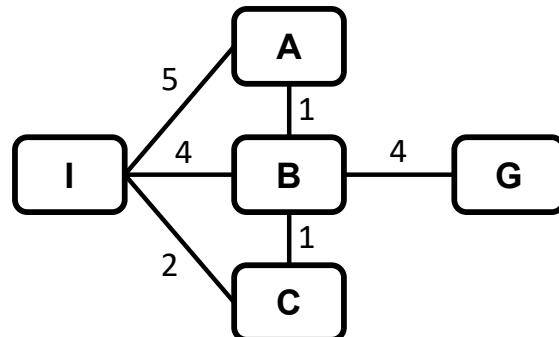
Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

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



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*

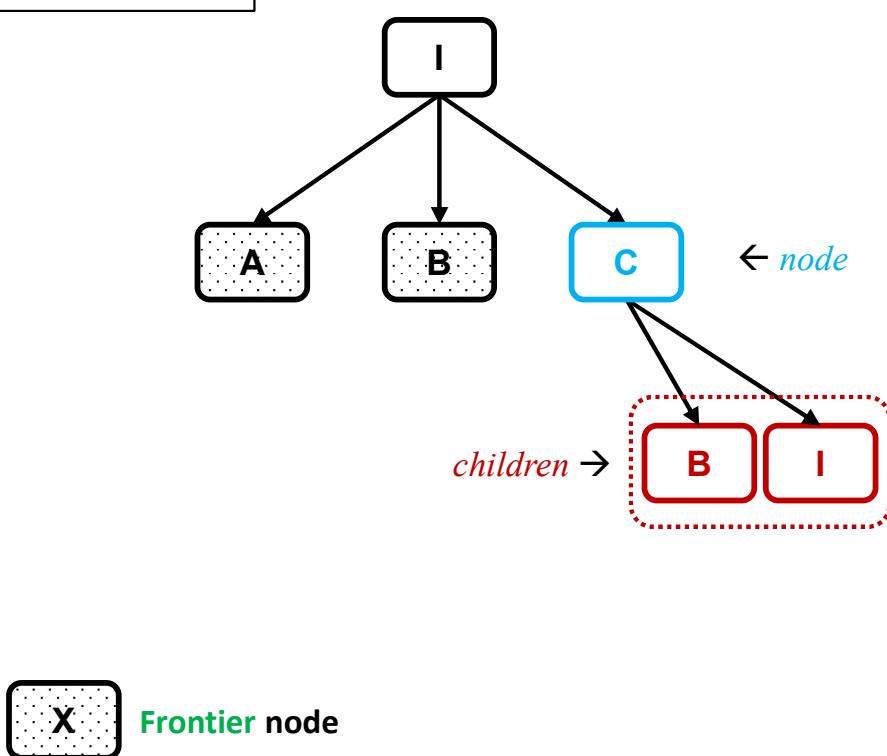
 add *child* to *frontier*

 return failure

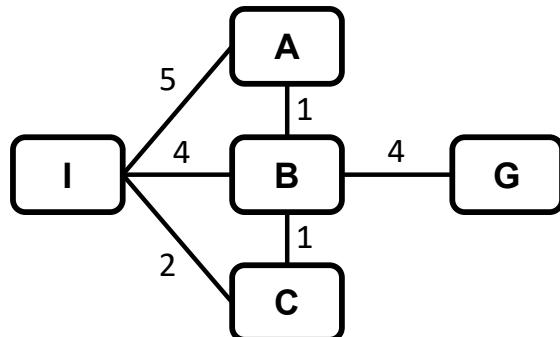
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

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*

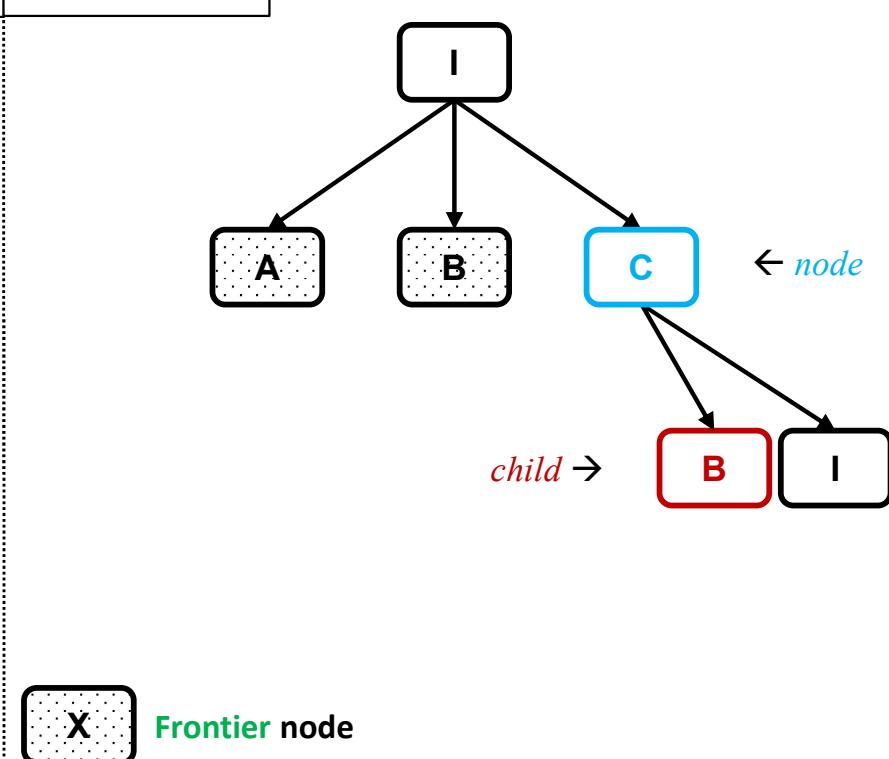
 add *child* to *frontier*

 return failure

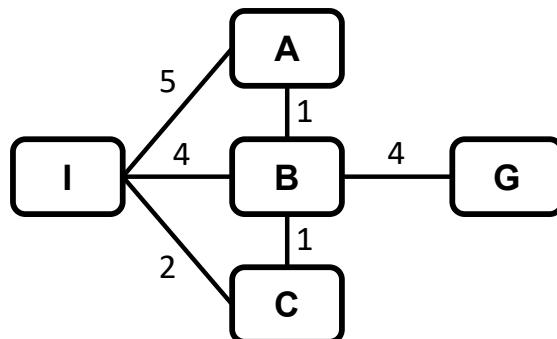
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

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

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

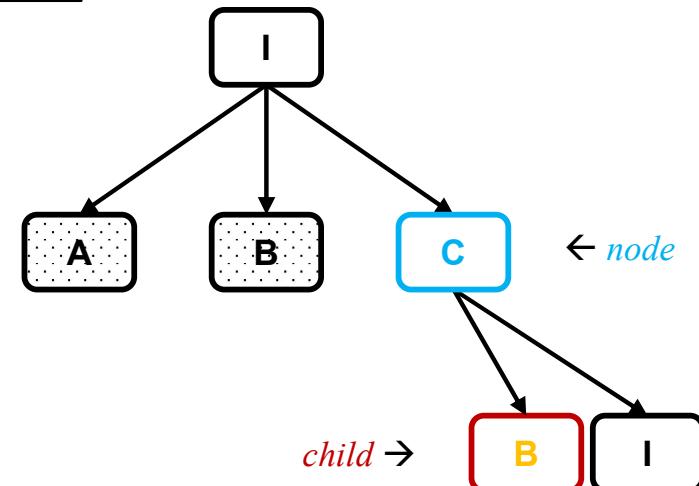
s \leftarrow *child.STATE*

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

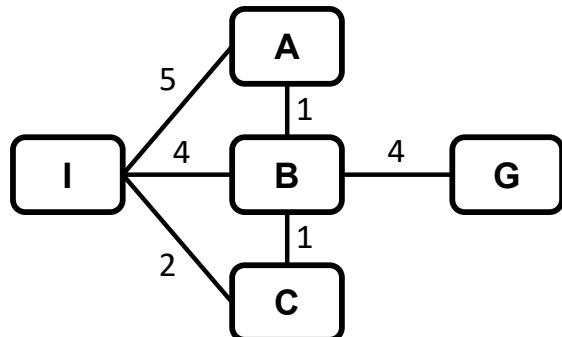
reached[s] \leftarrow *child*

 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*

 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*

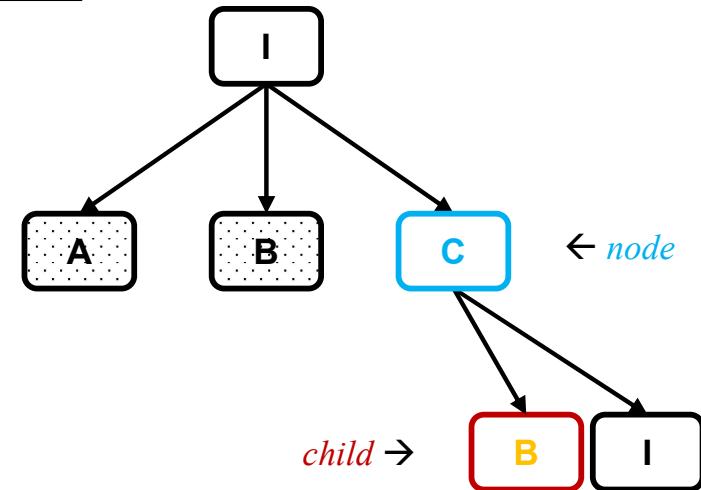
 add *child* to *frontier*

 return failure

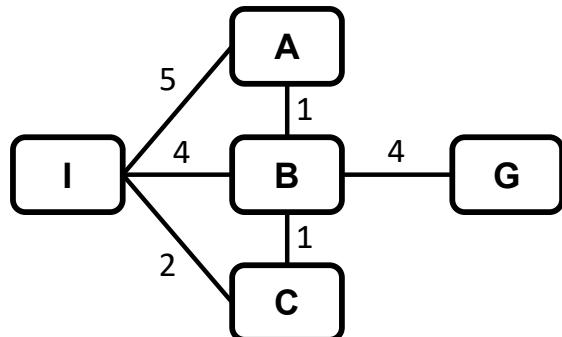
Frontier	Parent	I	I				
Node	B	A					
f(Node)	7	7					

Reached	Parent	---	I	I	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				
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

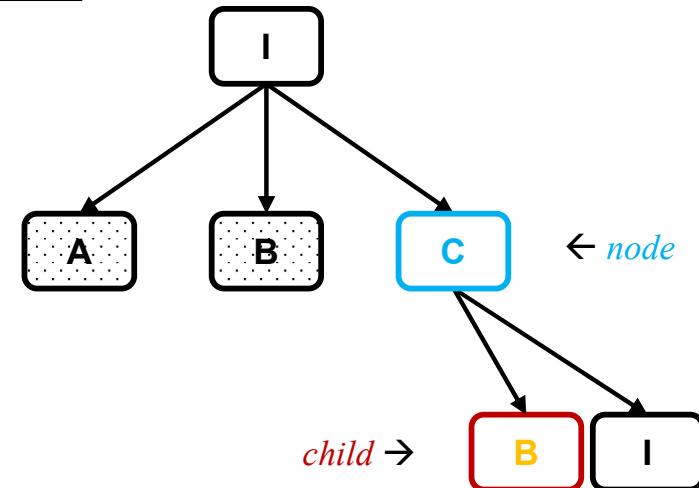
s \leftarrow *child.STATE*

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

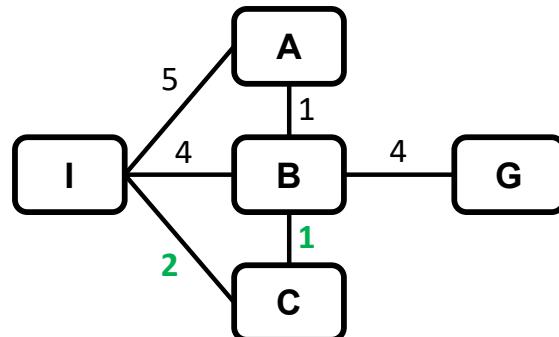
reached[s] \leftarrow *child*

 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

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*

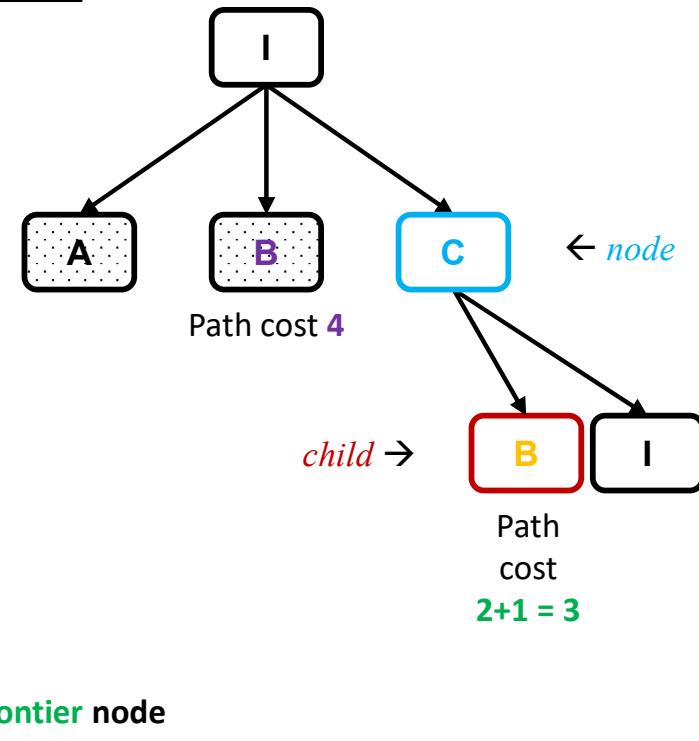
 add *child* to *frontier*

 return failure

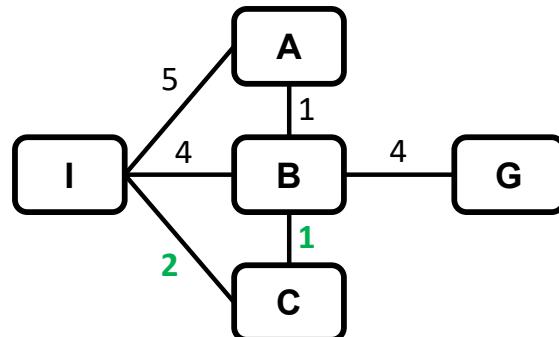
Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

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*

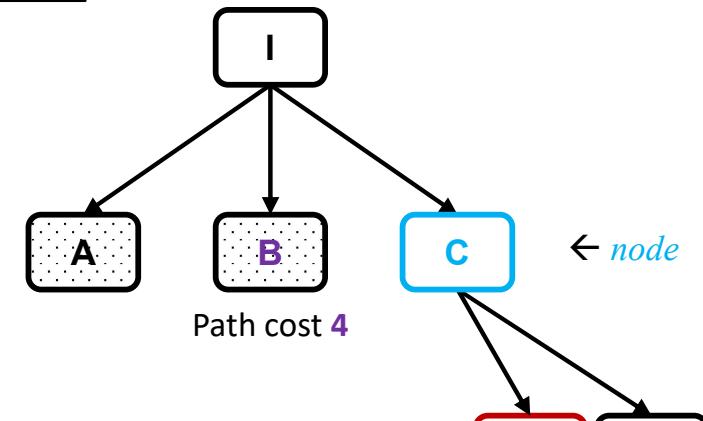
 add *child* to *frontier*

 return failure

Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

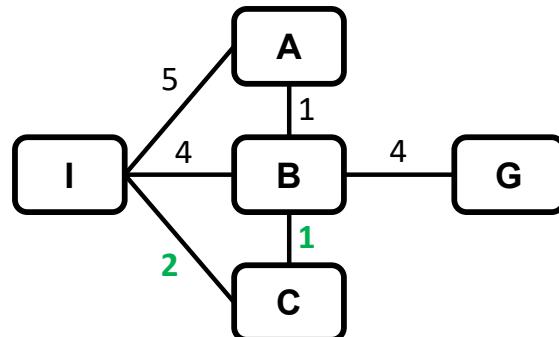
Reached	Parent	---	I	I	I		
Key/State	I	A	B	C			
Path cost	0	5	4	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



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

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

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

reached[s] \leftarrow *child*

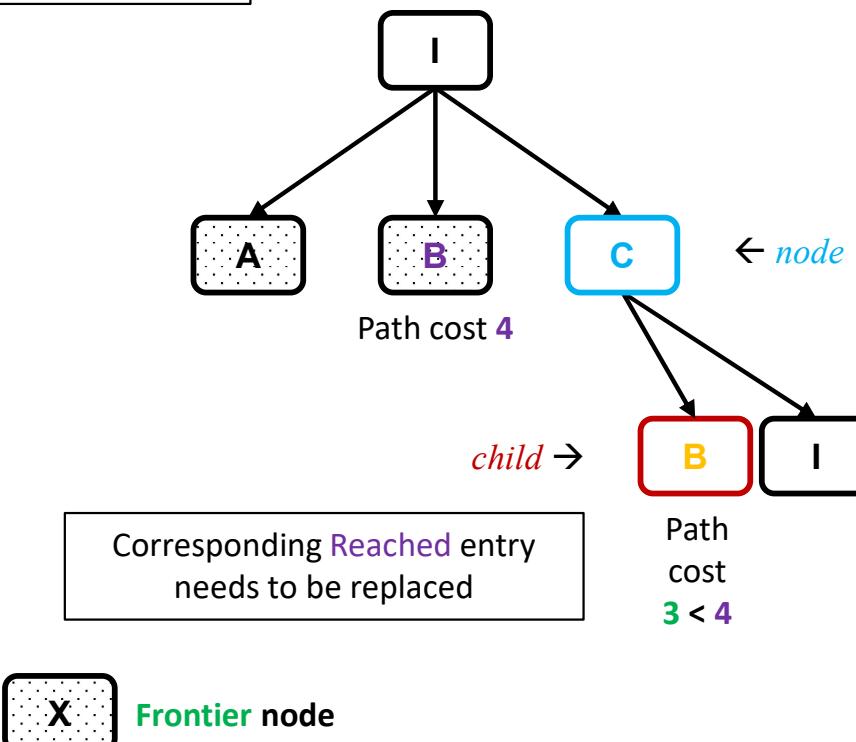
 add *child* to *frontier*

 return failure

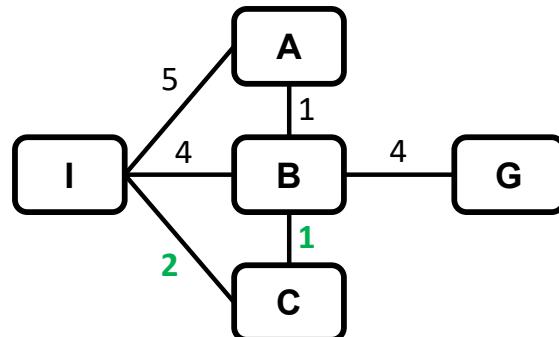
Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

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

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

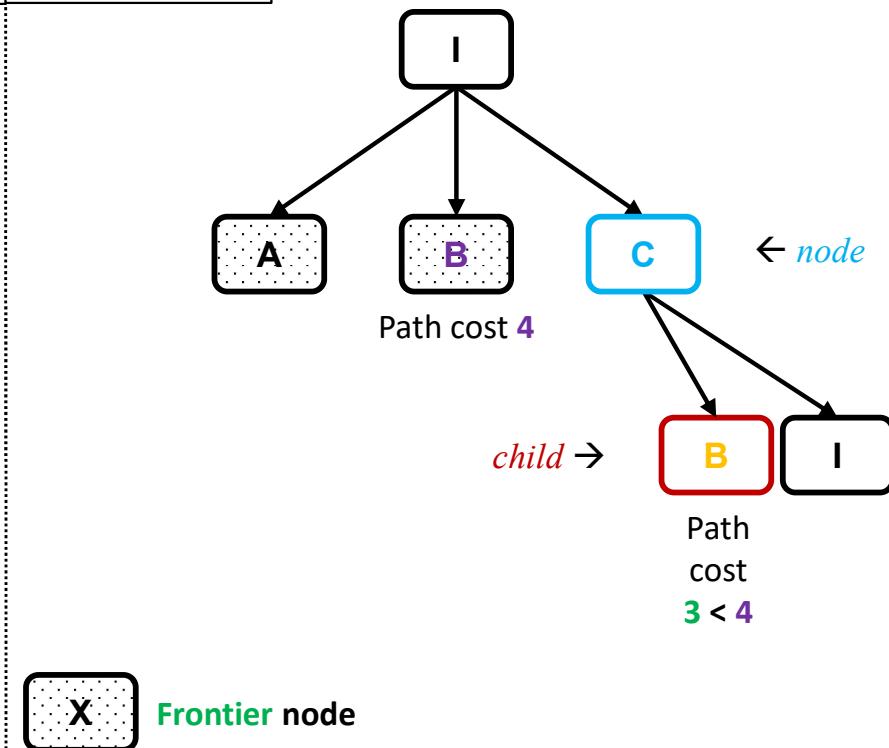
s \leftarrow *child.STATE*

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

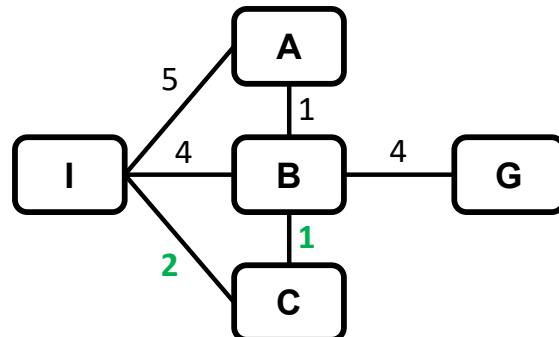
reached[s] \leftarrow *child*

 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*

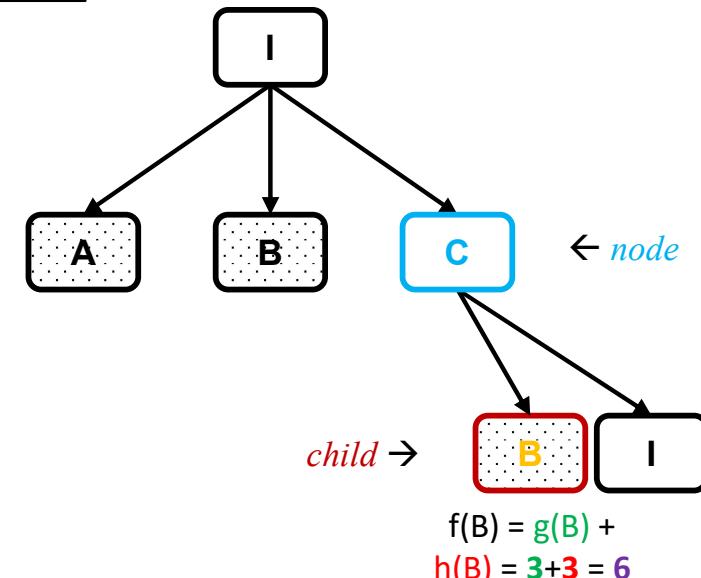
 add *child* to *frontier*

 return failure

Frontier	Parent	C	I	I			
Node	B		B	A			
<i>f</i> (Node)	6		7	7			

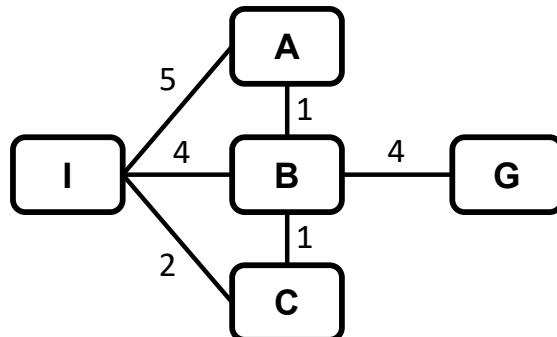
Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



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*

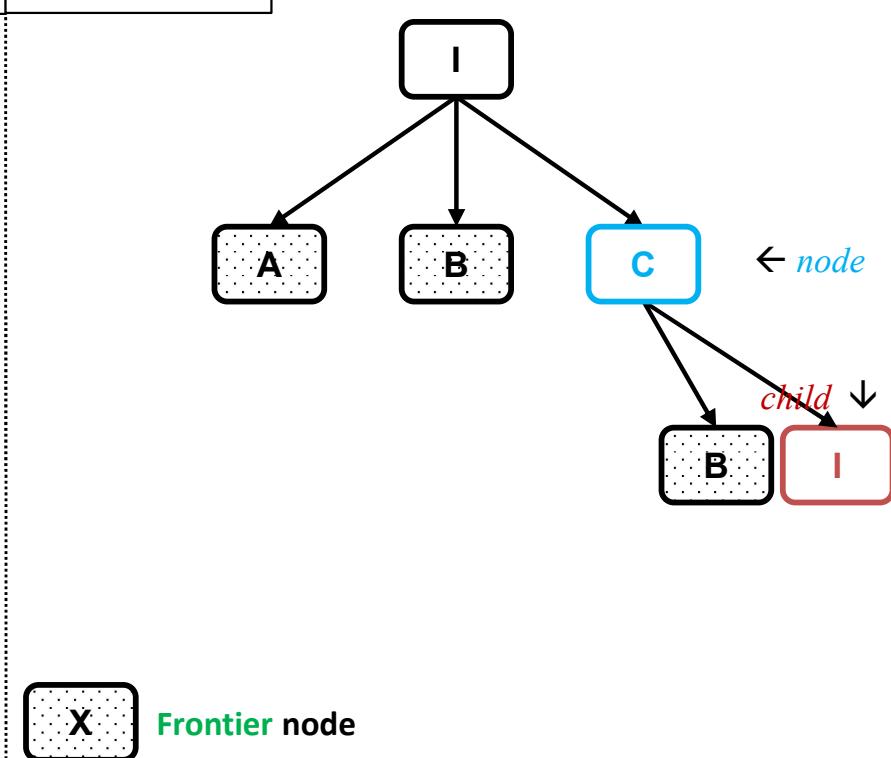
 add *child* to *frontier*

return failure

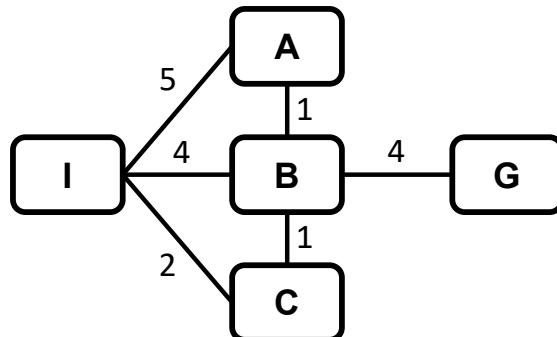
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	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**

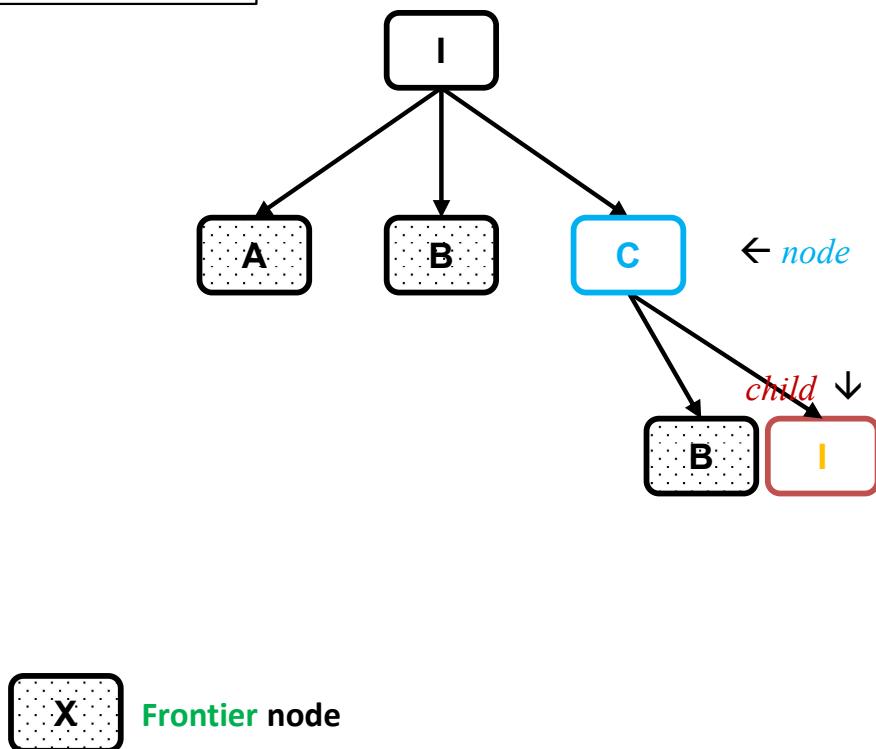
reached[s] \leftarrow *child*

 add *child* to *frontier*

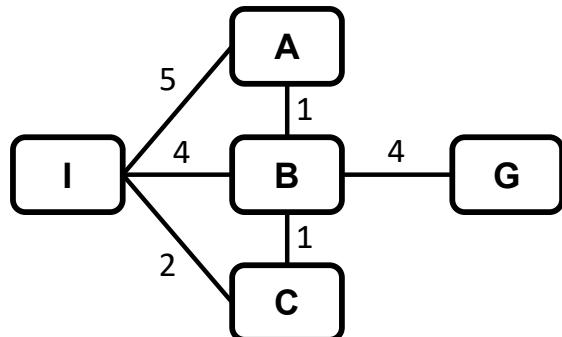
return failure

	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**

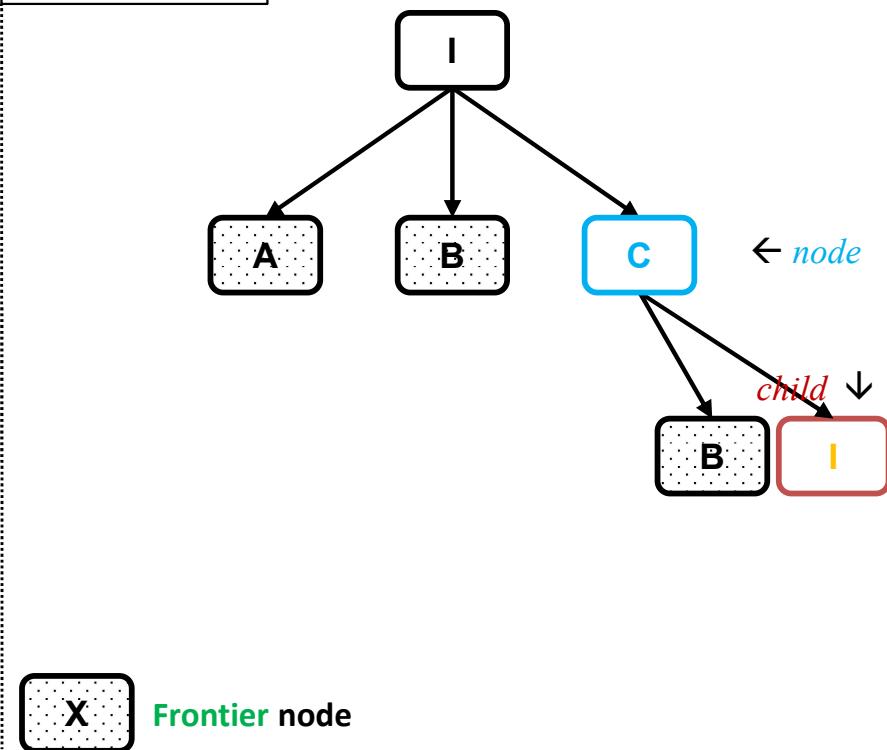
reached[s] \leftarrow *child*

 add *child* to *frontier*

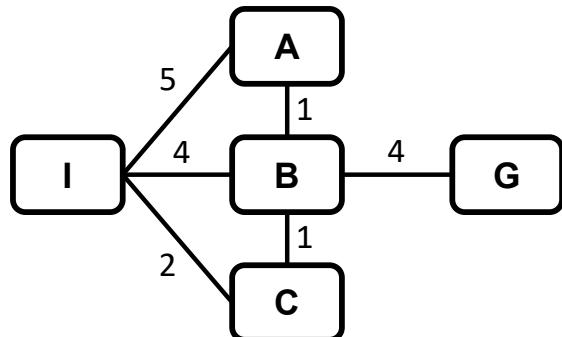
return failure

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*

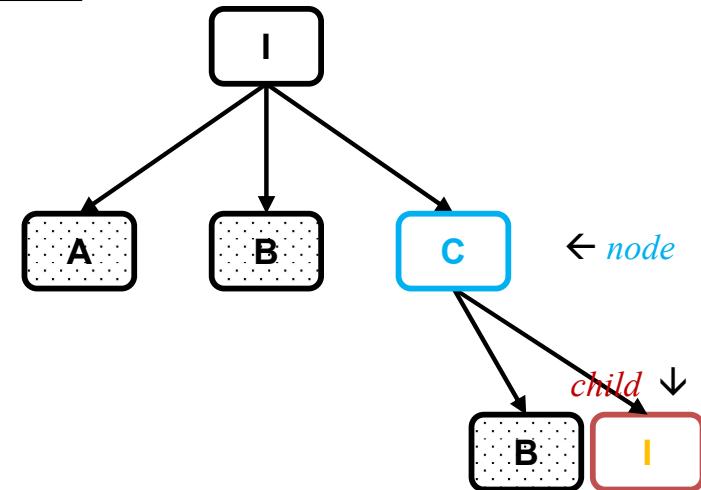
 add *child* to *frontier*

 return failure

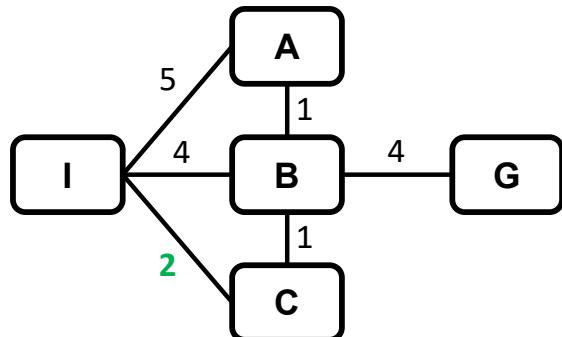
Frontier	Parent	C	I	I			
Node	B	B	A				
f(Node)	6	7	7				

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*

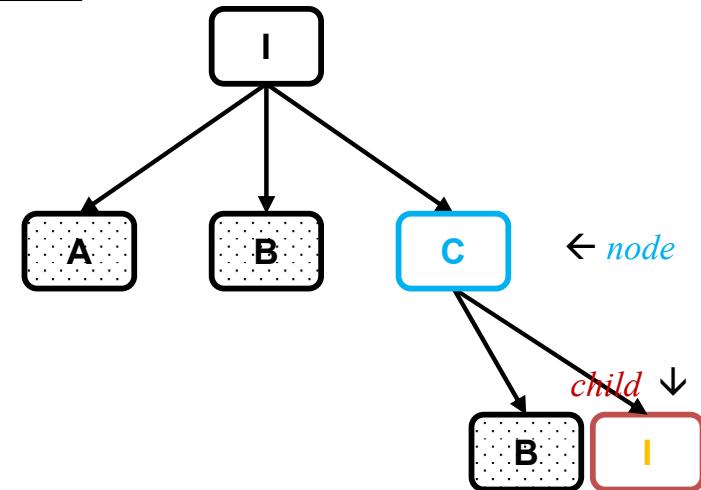
 add *child* to *frontier*

 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			

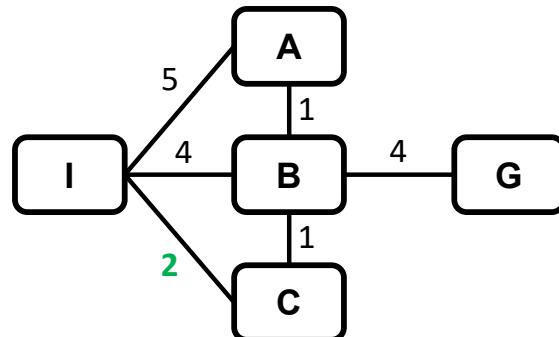
State Space Graph	Frontier / Reached
Algorithm	Search Tree



Path cost
 $2+2 = 4$

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*

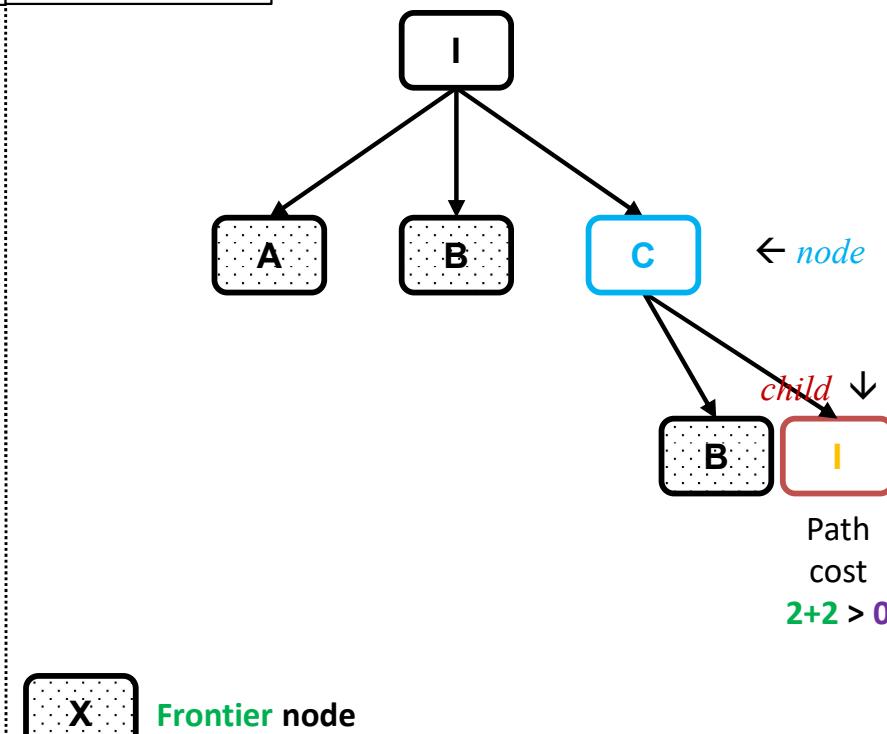
 add *child* to *frontier*

 return failure

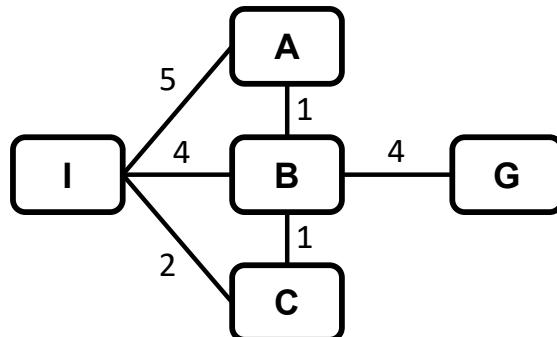
Frontier	Parent	C	I	I			
Node	B	B	A				
<i>f</i> (Node)	6	7	7				

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*

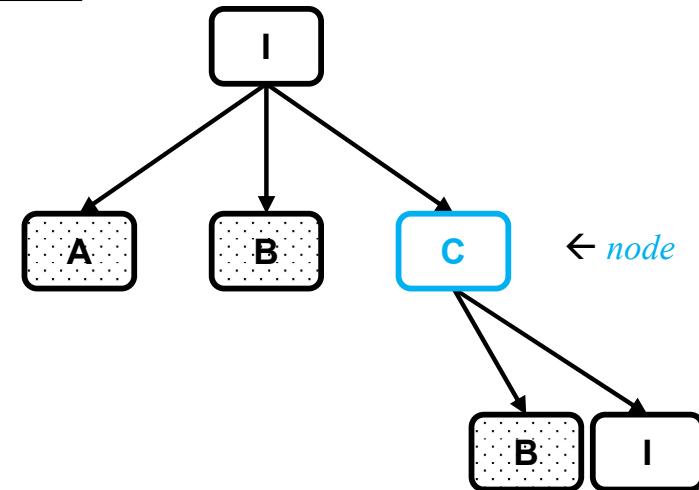
 add *child* to *frontier*

return failure

Frontier	Parent	C	I	I			
Node	B	B	A				
<i>f</i> (Node)	6	7	7				

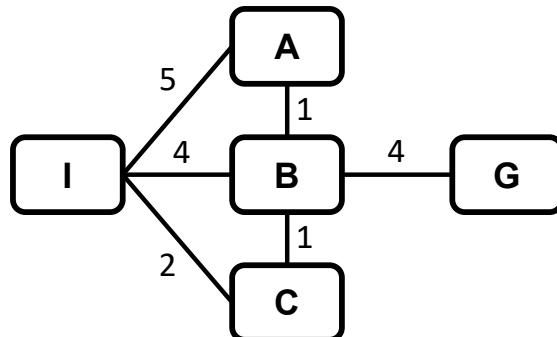
Reached	Parent	---	I	C	I		
Key/State	I	A	B	C			
Path cost	0	5	3	2			

State Space Graph	Frontier / Reached
Algorithm	Search Tree



X 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**

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*

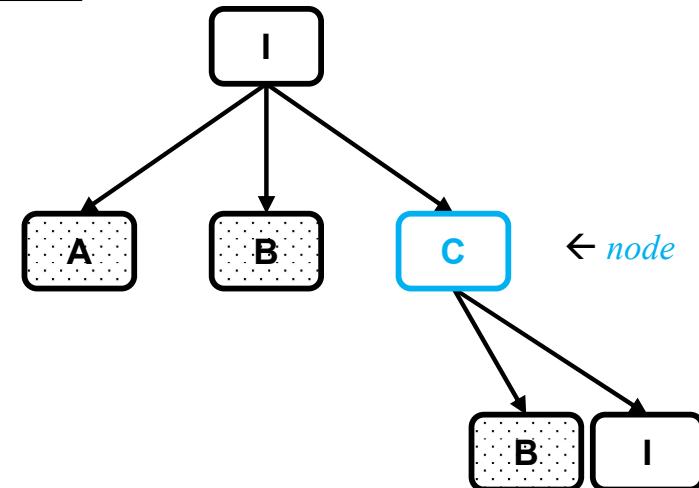
 add *child* to *frontier*

return failure

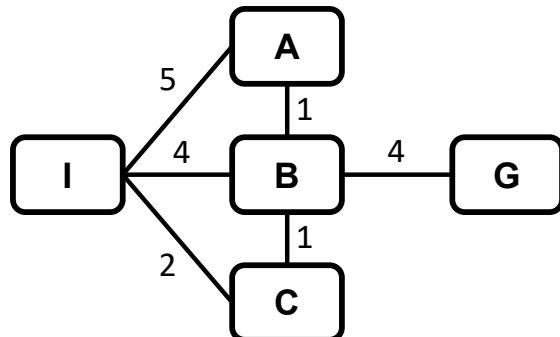
Frontier	Parent	C	I	I			
Node	B	B	A				
f(Node)	6	7	7				

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(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**

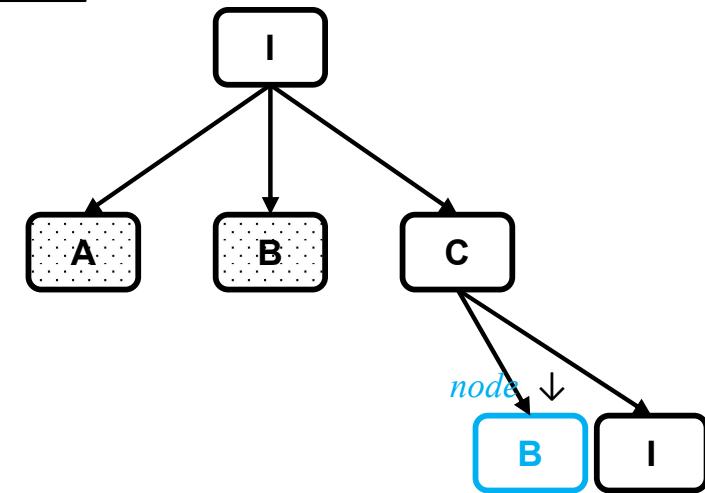
reached[s] \leftarrow *child*

 add *child* to *frontier*

return failure

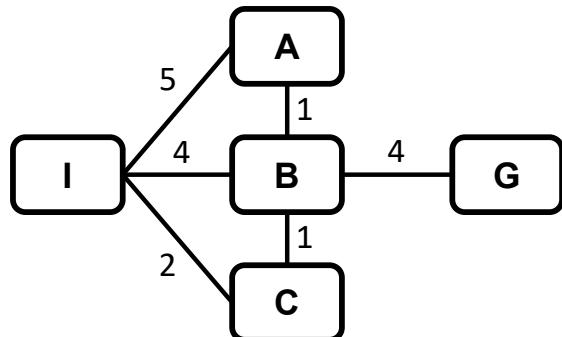
State Space Graph		Frontier / Reached
Algorithm		Search Tree

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		



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**

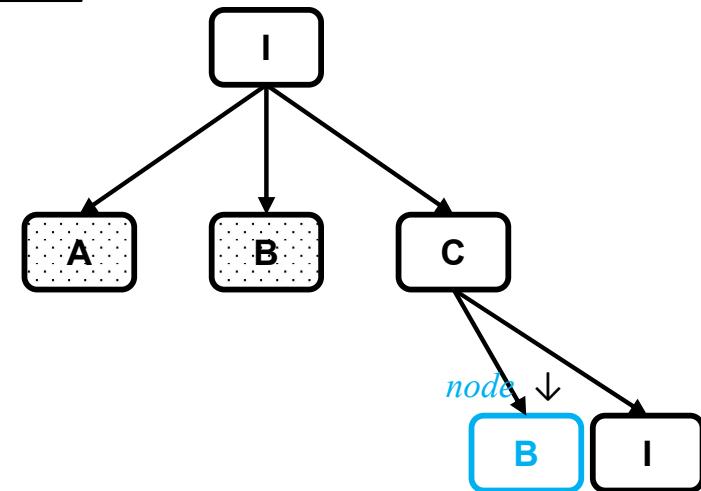
reached[s] \leftarrow *child*

 add *child* to *frontier*

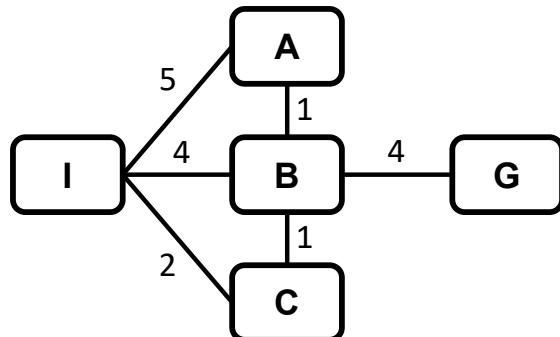
return failure

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**

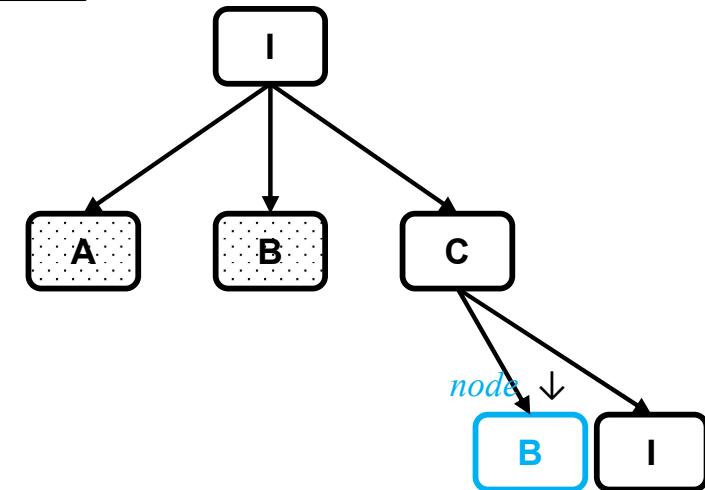
reached[s] \leftarrow *child*

 add *child* to *frontier*

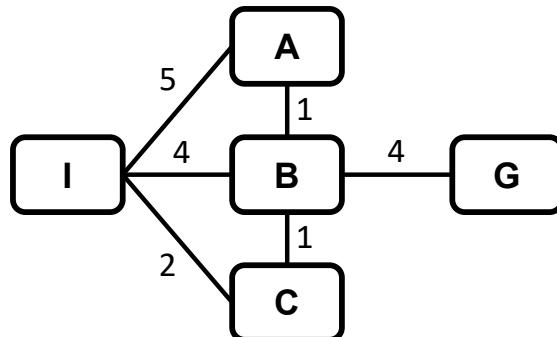
return failure

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*

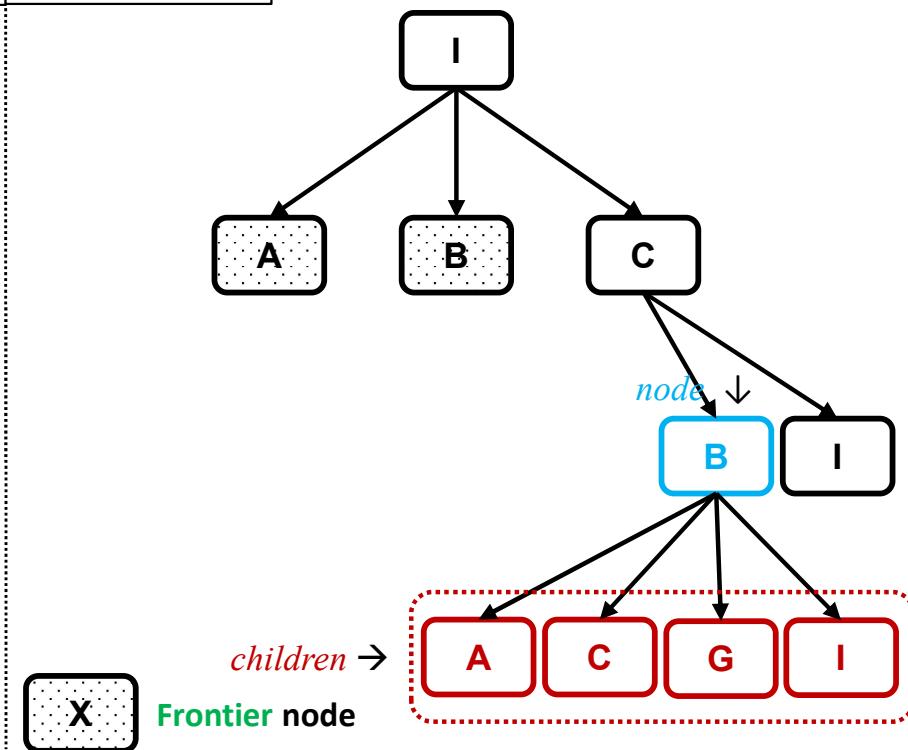
 add *child* to *frontier*

 return failure

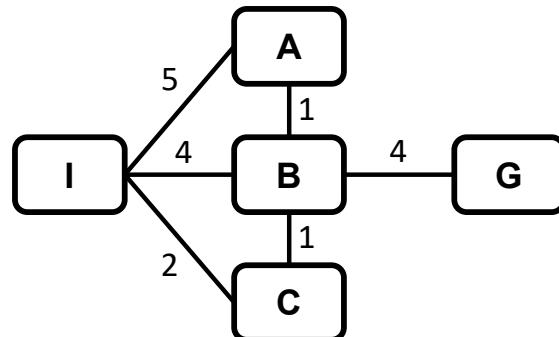
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

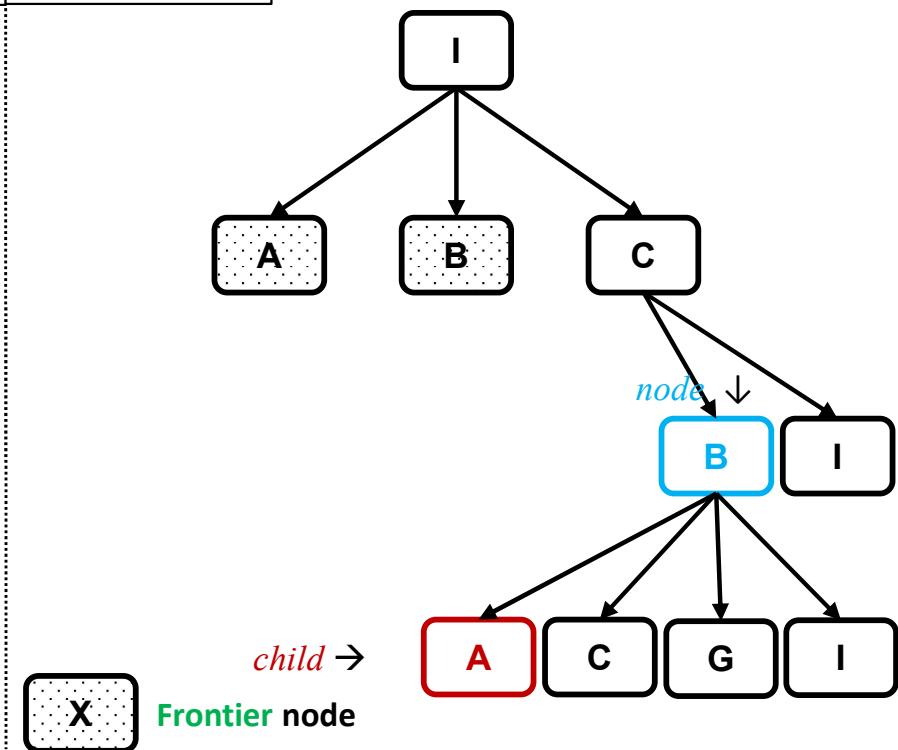
 add *child* to *frontier*

 return failure

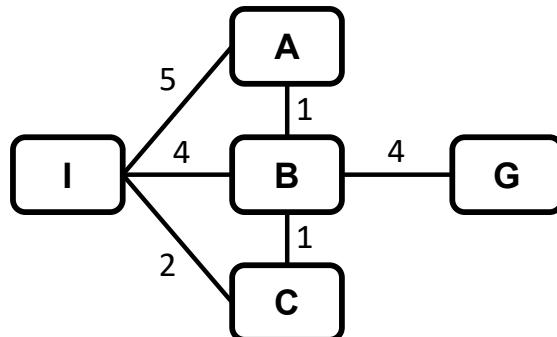
Frontier	Parent	I	I			
Node	B	A				
$f(\text{Node})$	7	7				

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

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*

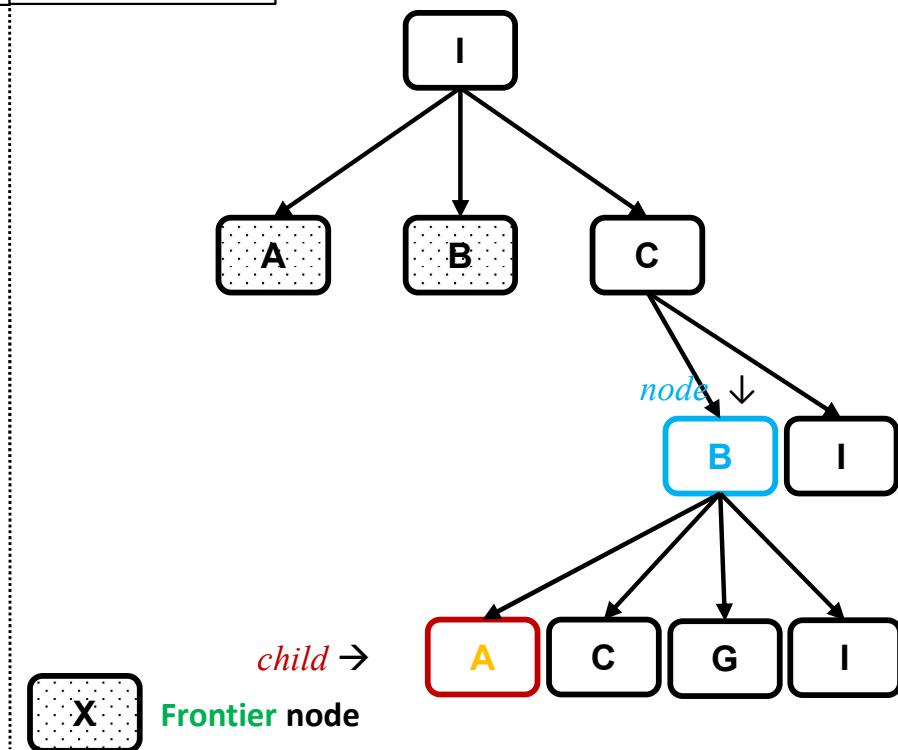
 add *child* to *frontier*

 return failure

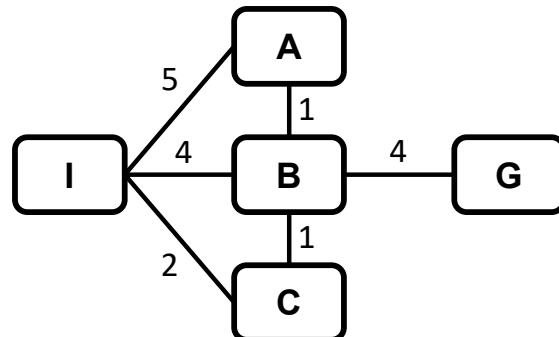
	State Space Graph	Frontier / Reached
Algorithm		Search Tree

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*

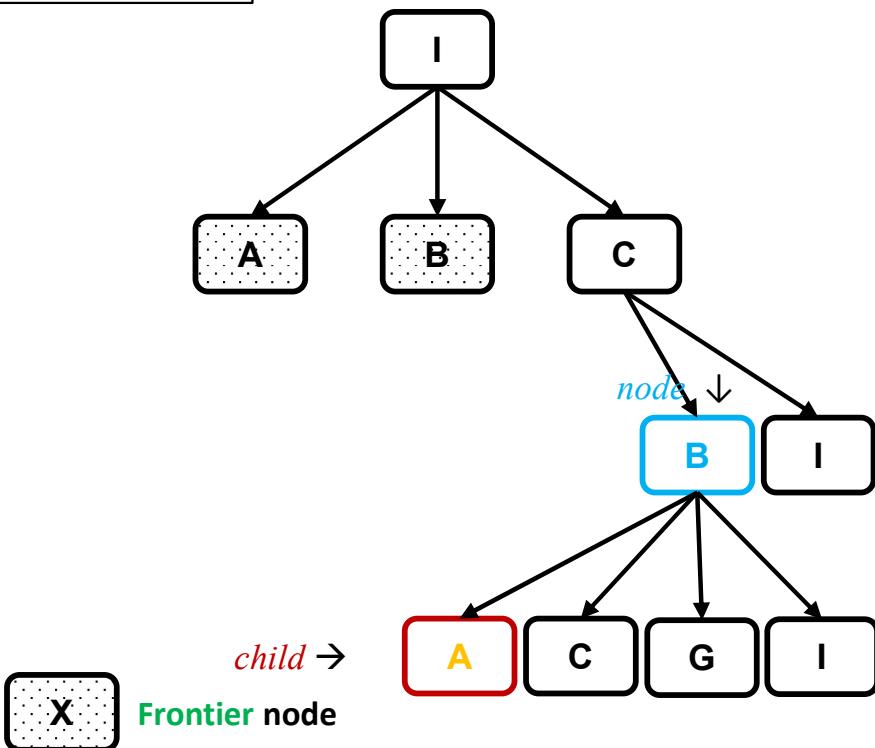
 add *child* to *frontier*

 return failure

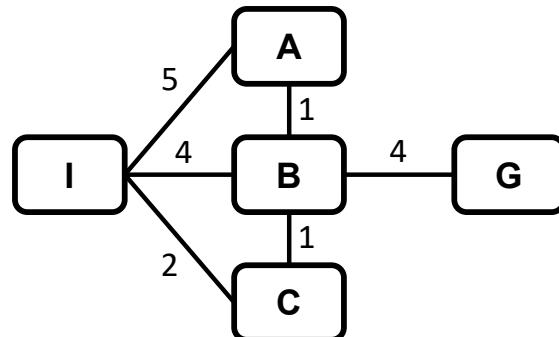
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

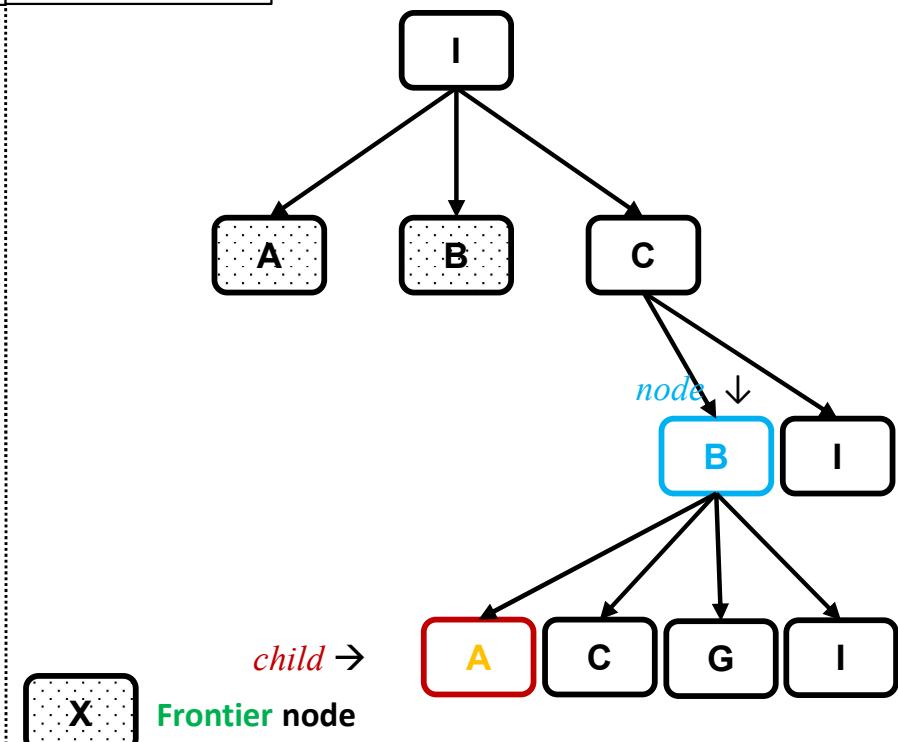
 add *child* to *frontier*

return failure

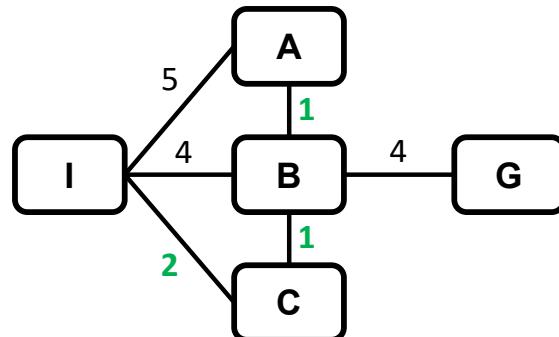
Frontier	Parent	I	I			
	Node	B	A			
	<i>f</i> (Node)	7	7			

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* ~~< i reached[s].PATH-COST~~ then

reached[*s*] \leftarrow *child*

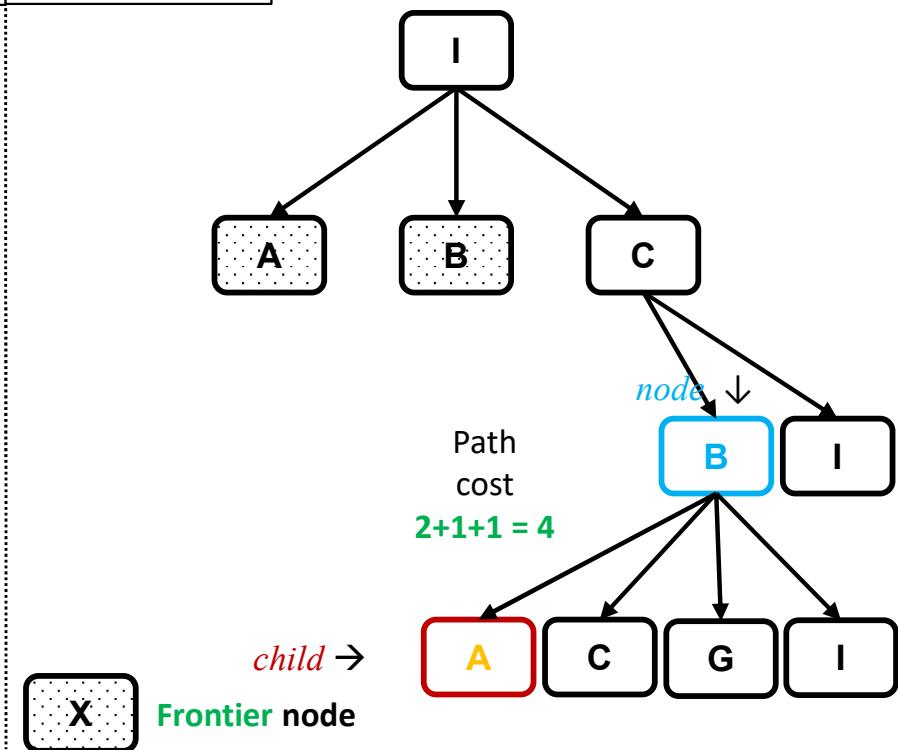
 add *child* to *frontier*

 return failure

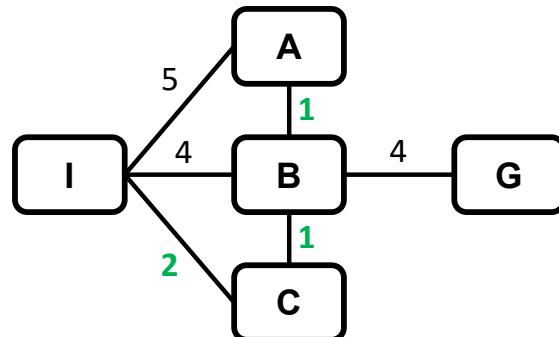
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

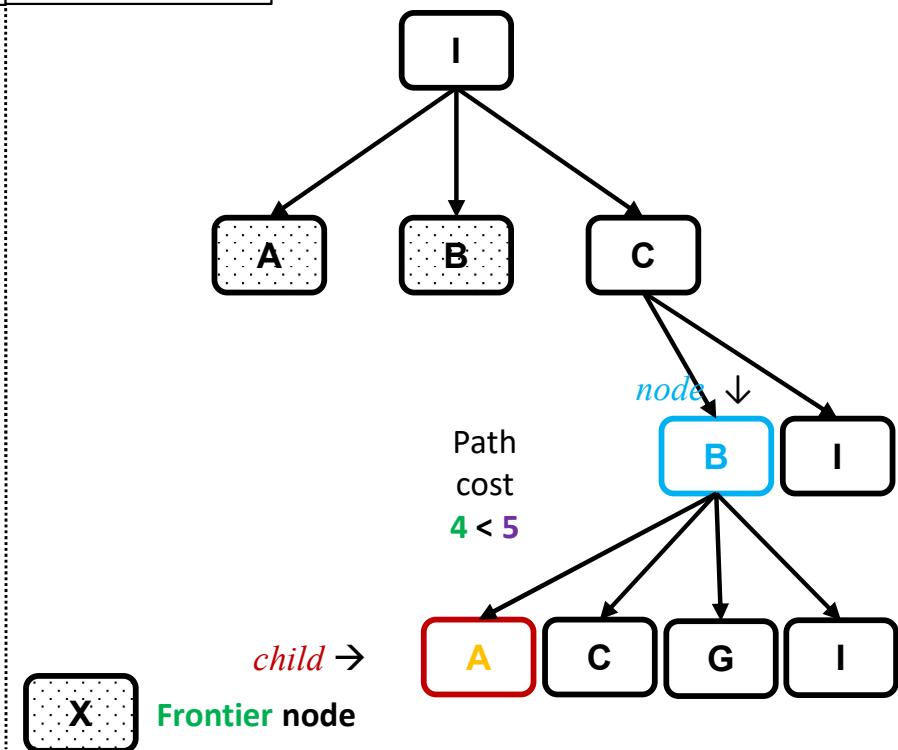
 add *child* to *frontier*

 return failure

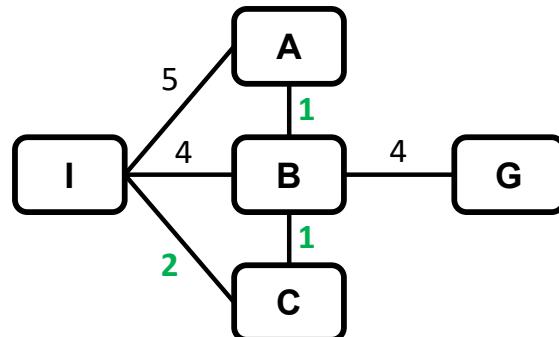
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

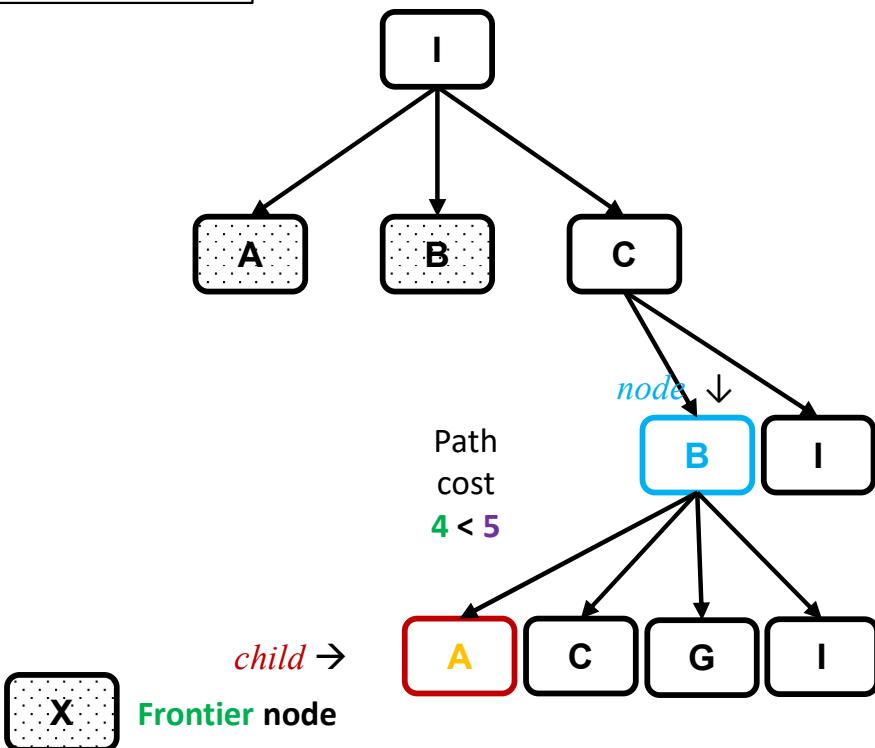
 add *child* to *frontier*

 return failure

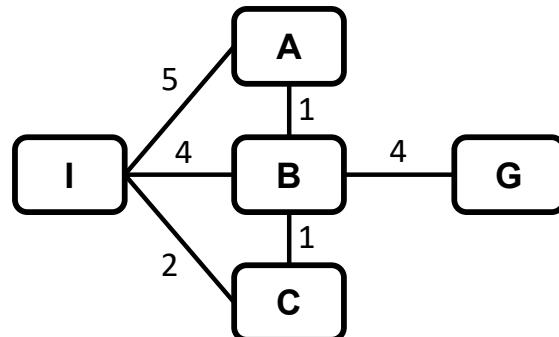
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

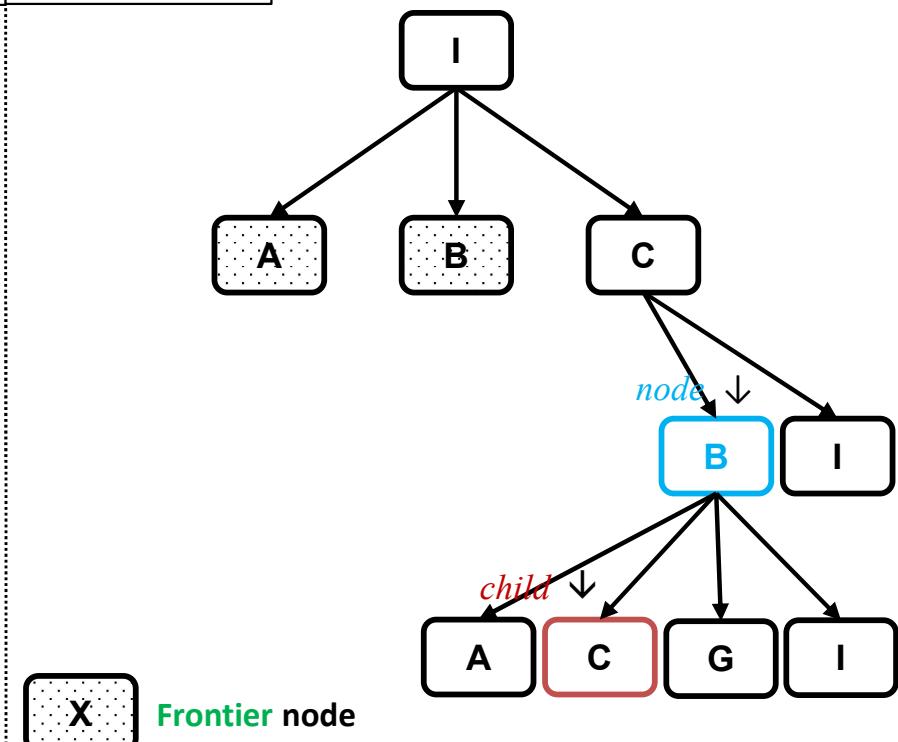
 add *child* to *frontier*

return failure

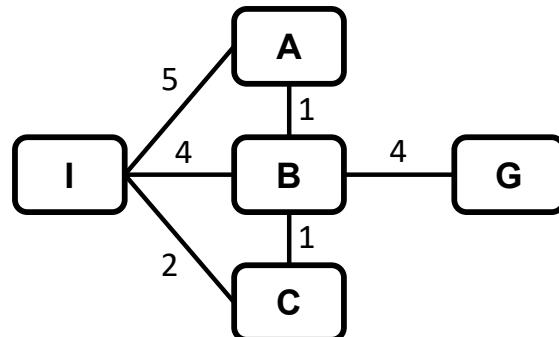
Frontier	Parent	I	I			
	Node	B	A			
	<i>f</i> (Node)	7	7			

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*

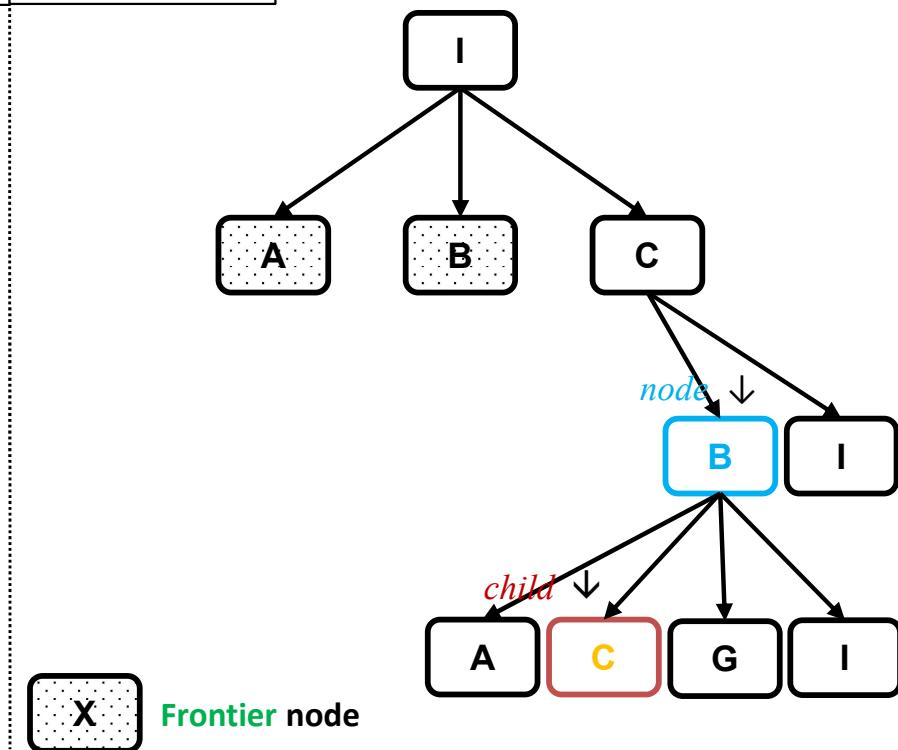
 add *child* to *frontier*

return failure

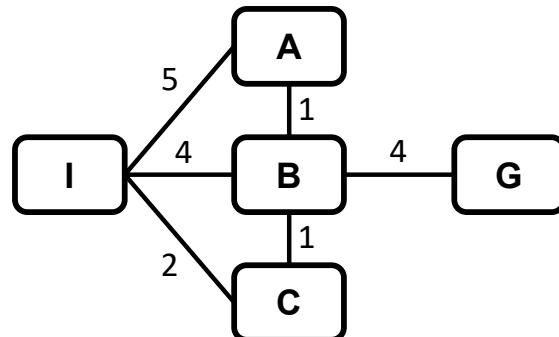
State Space Graph	Frontier / Reached
Algorithm	Search Tree

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*

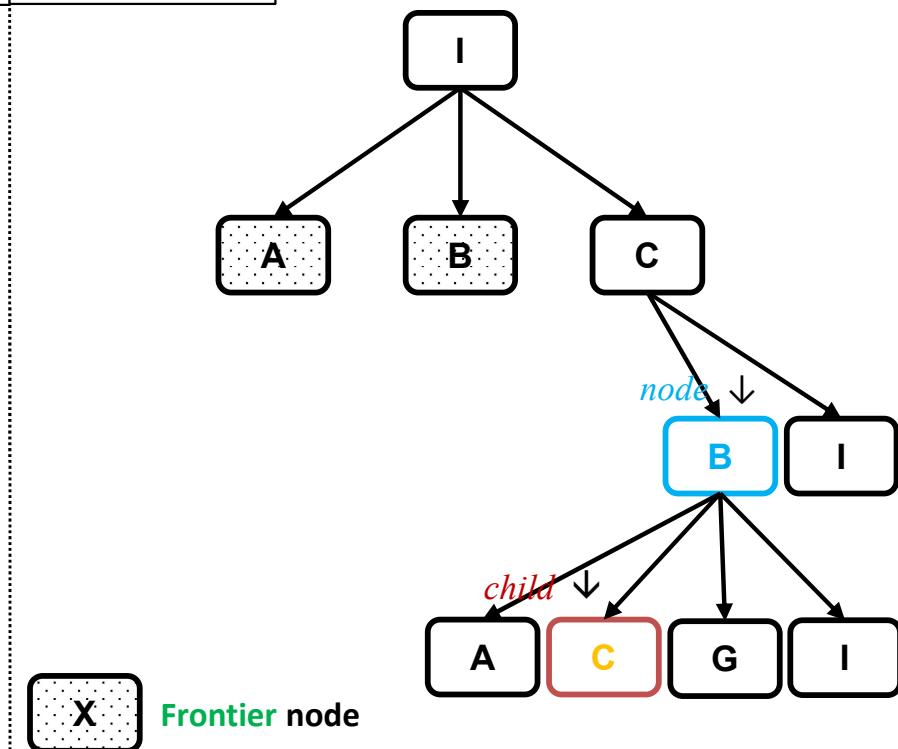
 add *child* to *frontier*

return failure

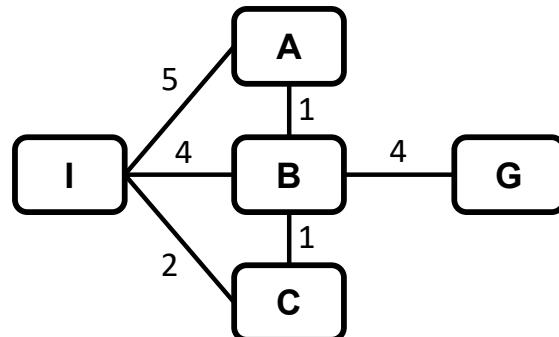
State Space Graph	Frontier / Reached
Algorithm	Search Tree

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

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

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

reached[s] \leftarrow *child*

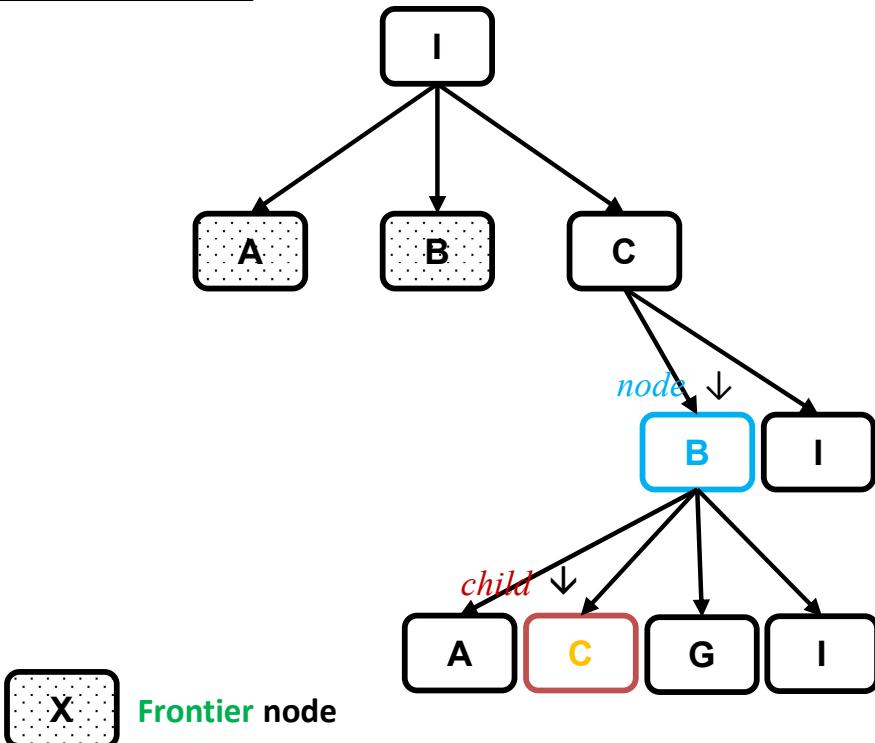
 add *child* to *frontier*

 return failure

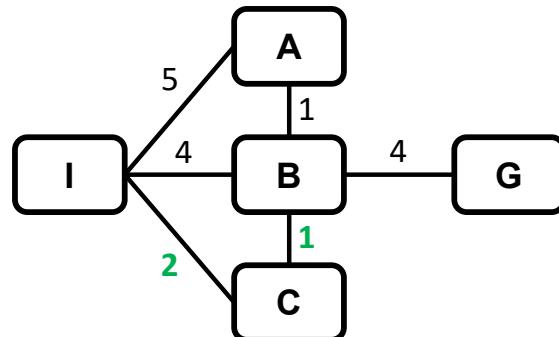
Frontier	Parent	I	I			
Node	B	A				
$f(\text{Node})$	7	7				

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

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*

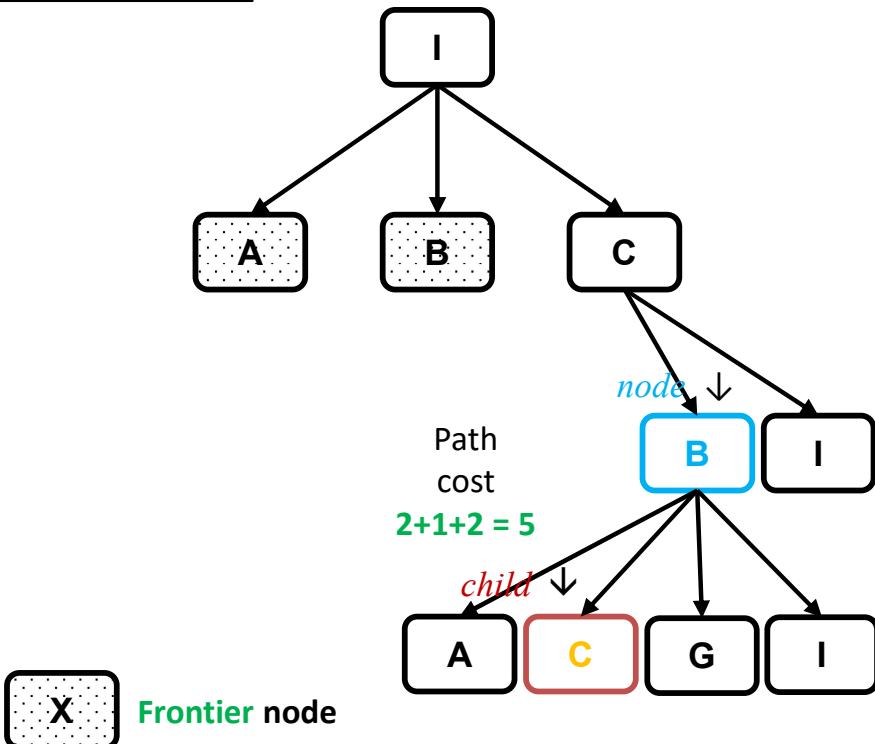
 add *child* to *frontier*

 return failure

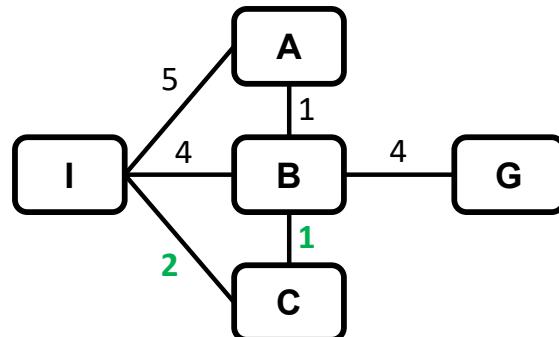
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

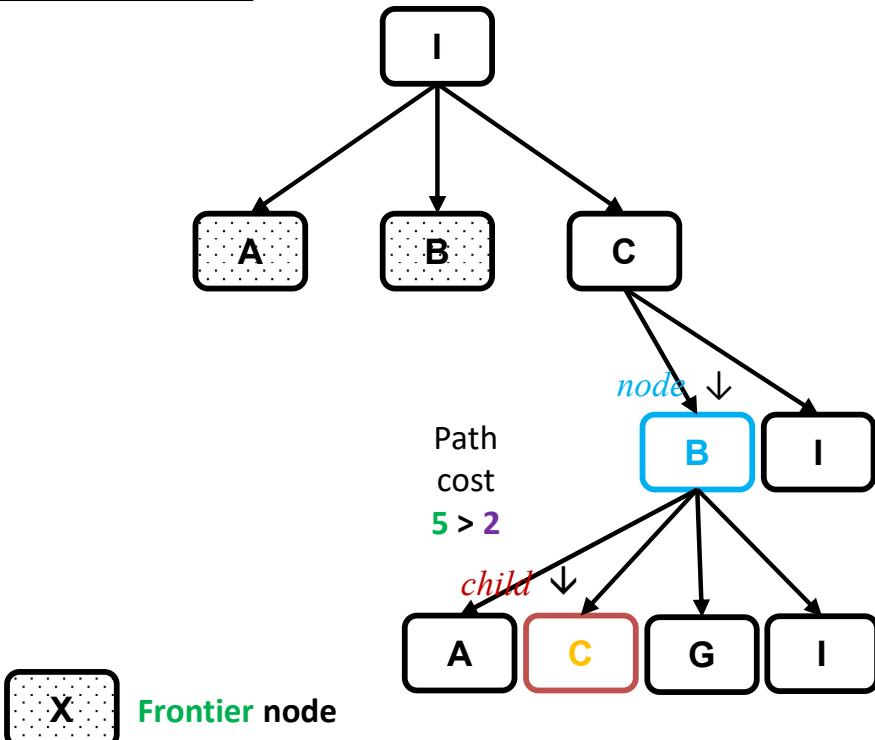
 add *child* to *frontier*

 return failure

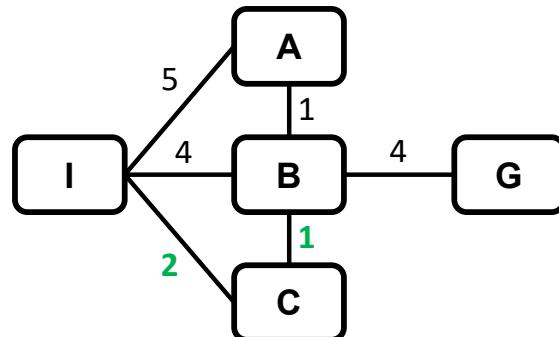
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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

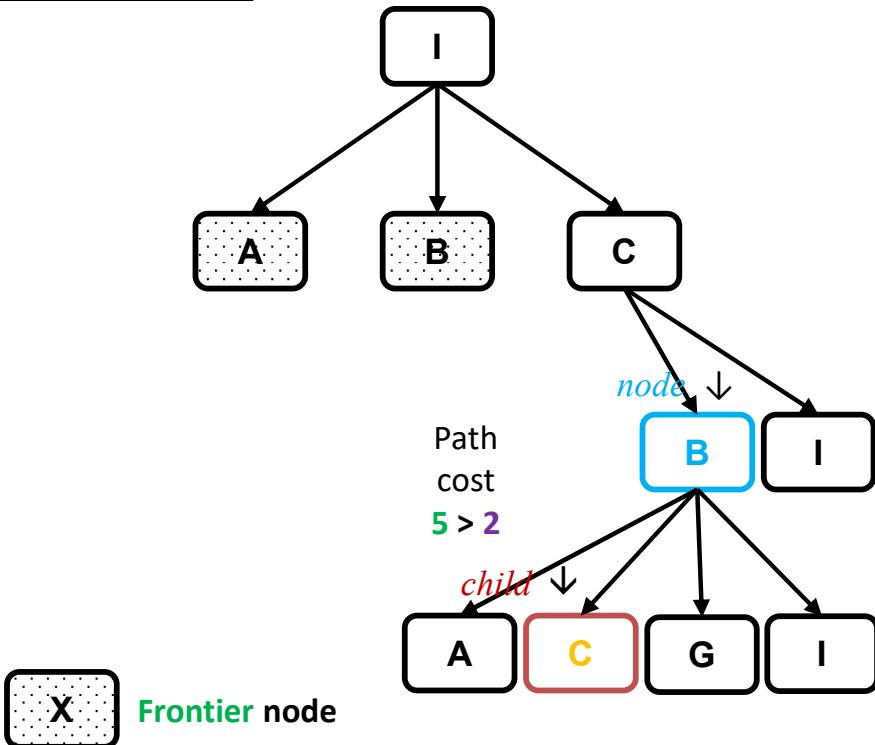
 add *child* to *frontier*

return failure

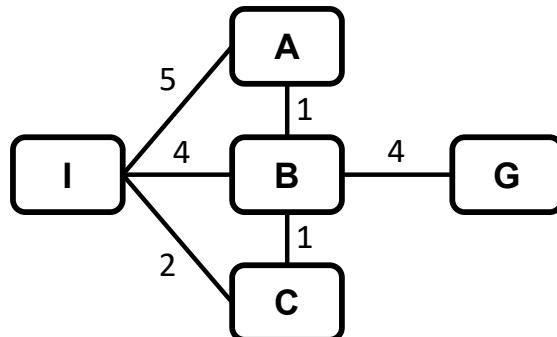
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

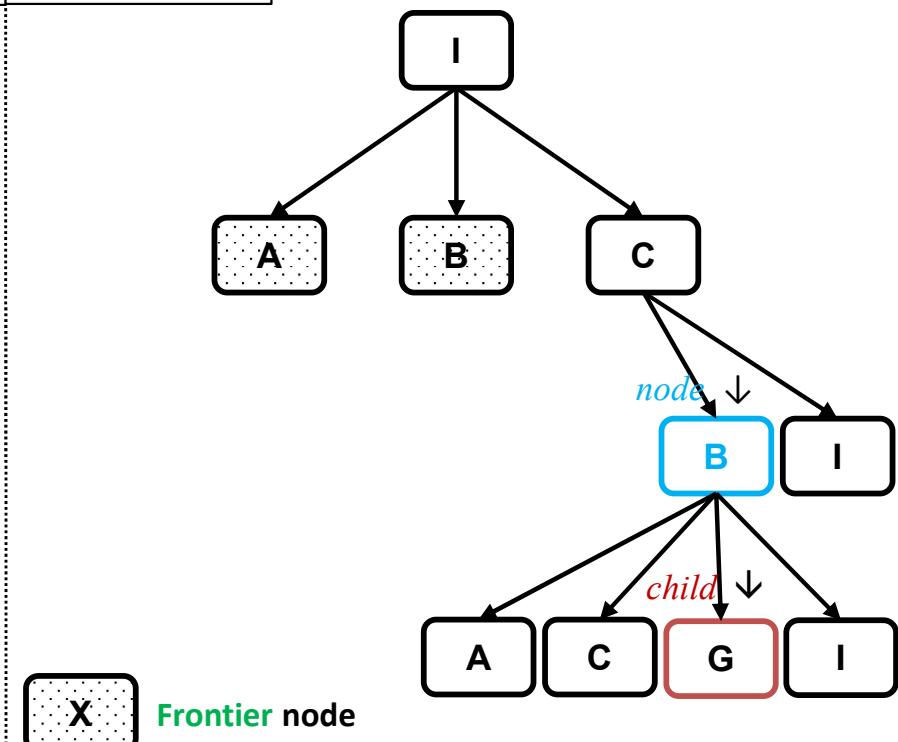
 add *child* to *frontier*

return failure

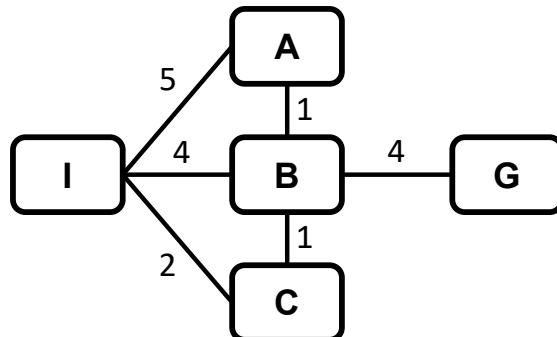
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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**

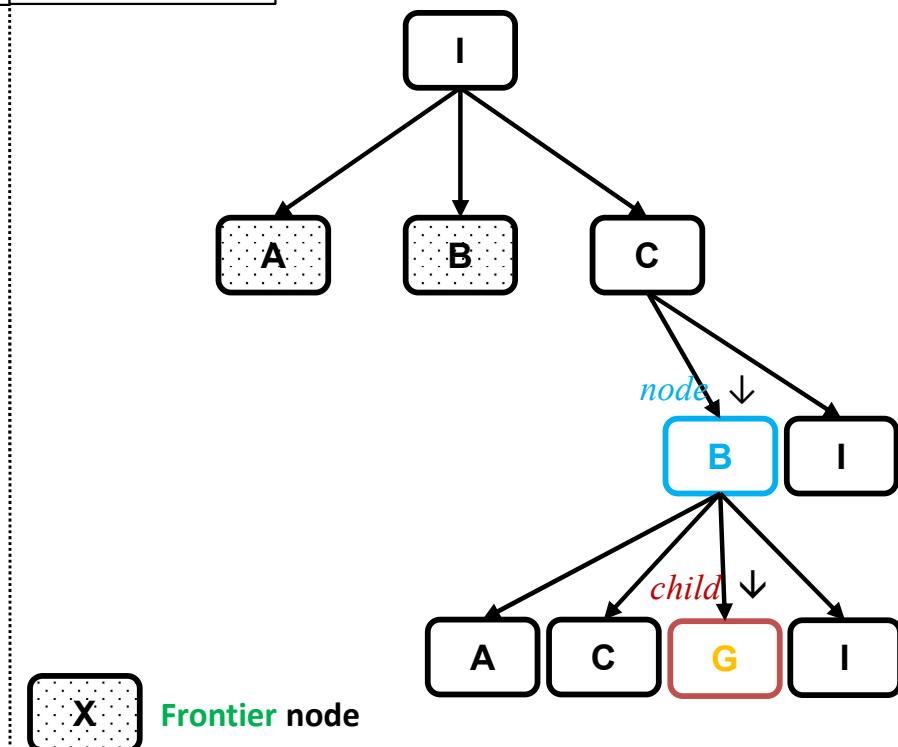
reached[s] \leftarrow *child*

 add *child* to *frontier*

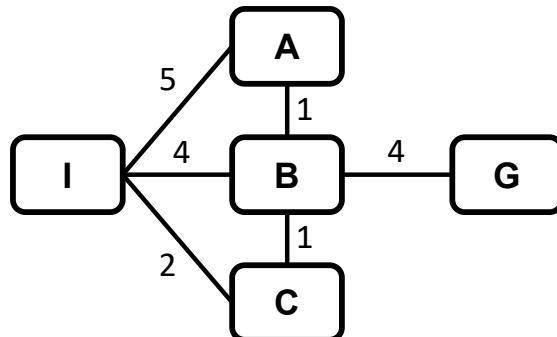
return failure

	State Space Graph	Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	<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*

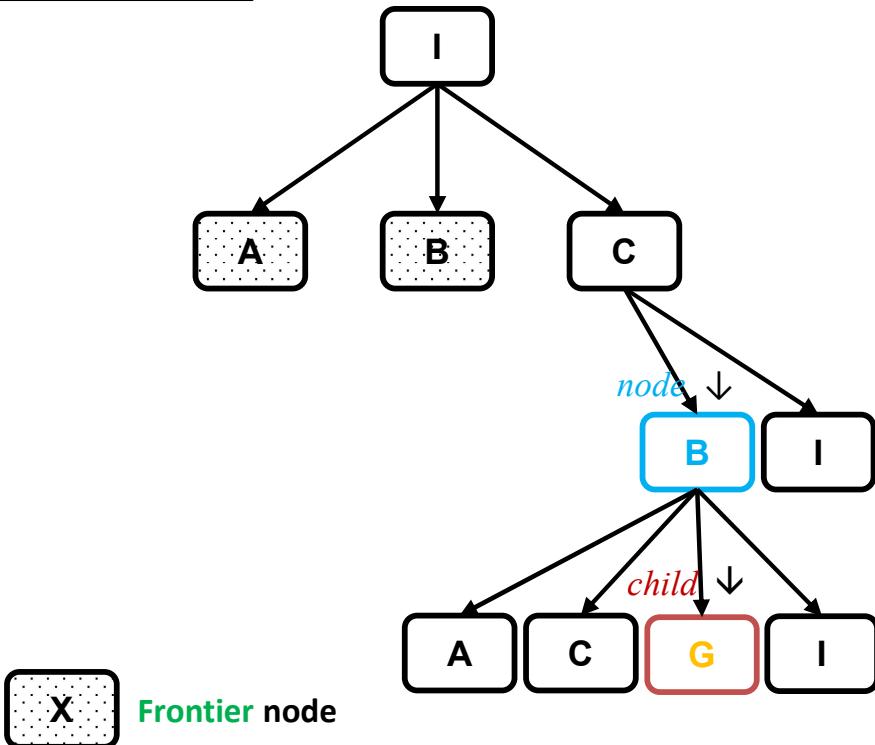
 add *child* to *frontier*

 return failure

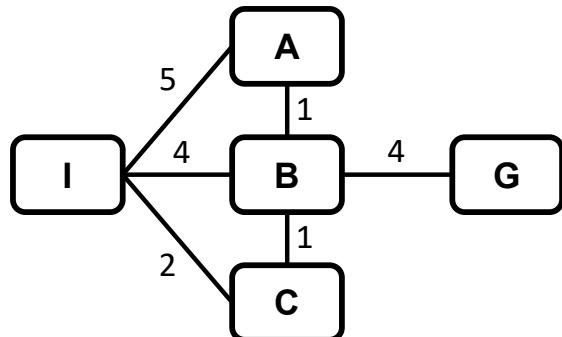
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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*

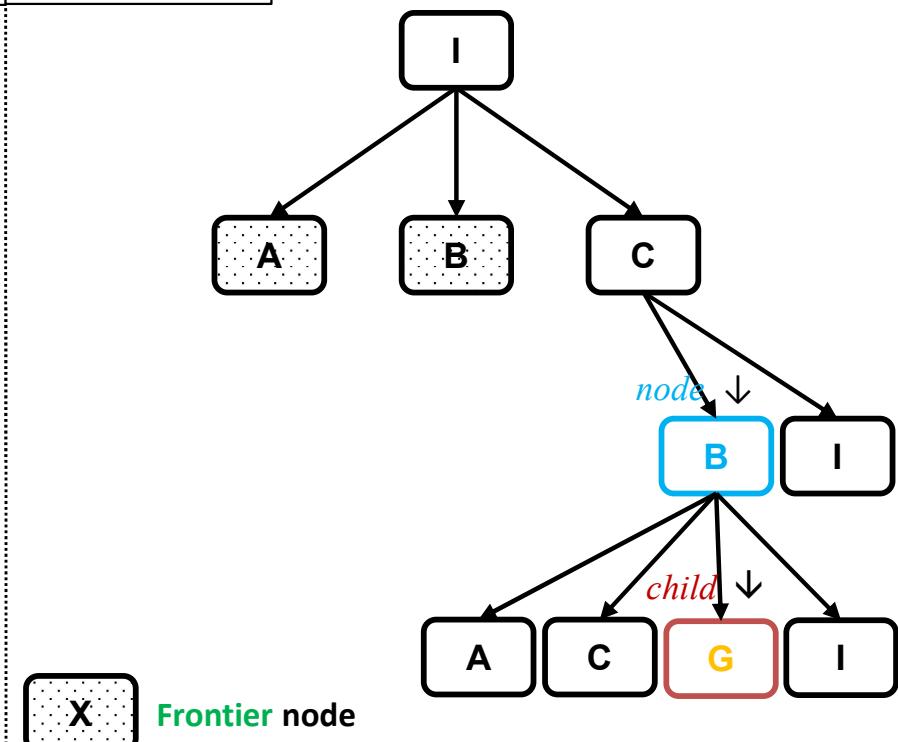
 add *child* to *frontier*

return failure

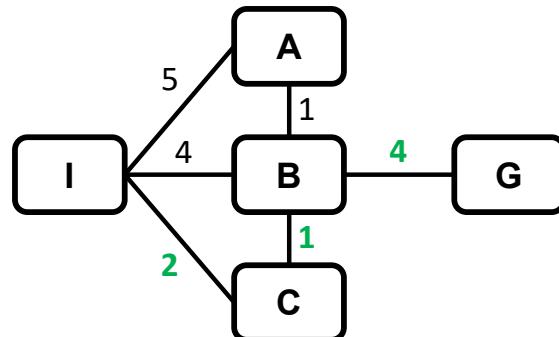
Frontier	Parent	I	I			
Node	B	A				
<i>f</i> (Node)	7	7				

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

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**

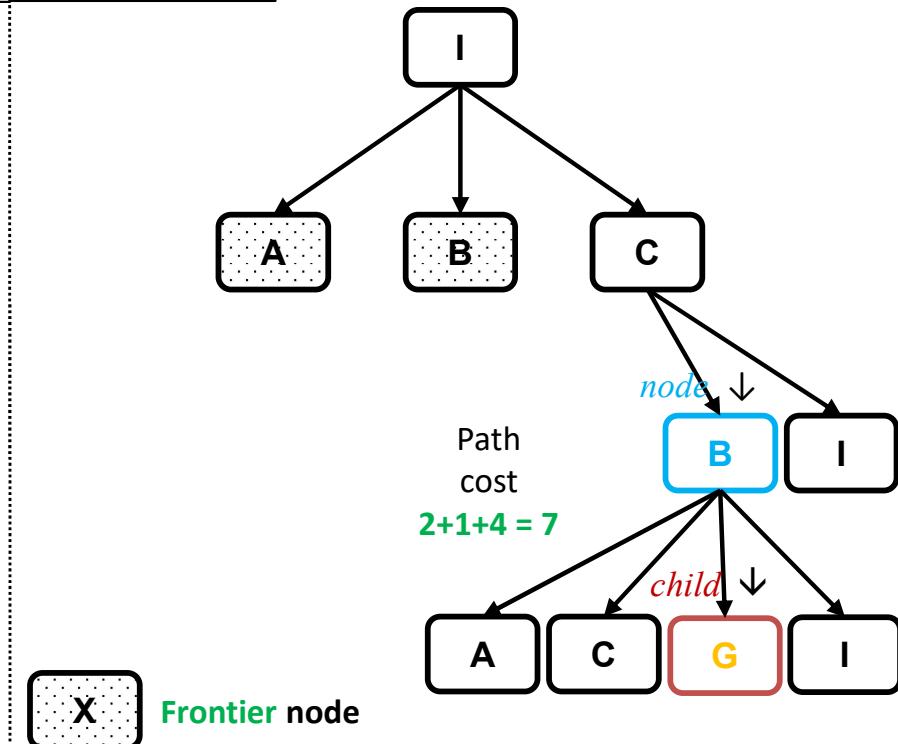
s \leftarrow *child.STATE*

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

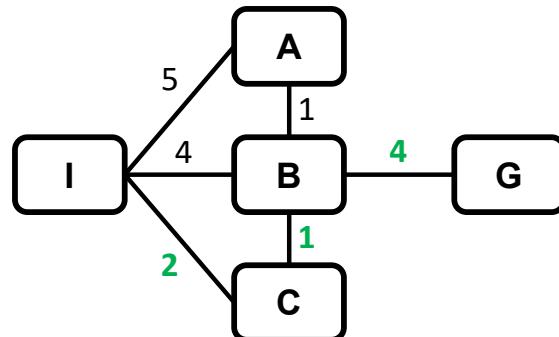
reached[s] \leftarrow *child*

 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

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

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**

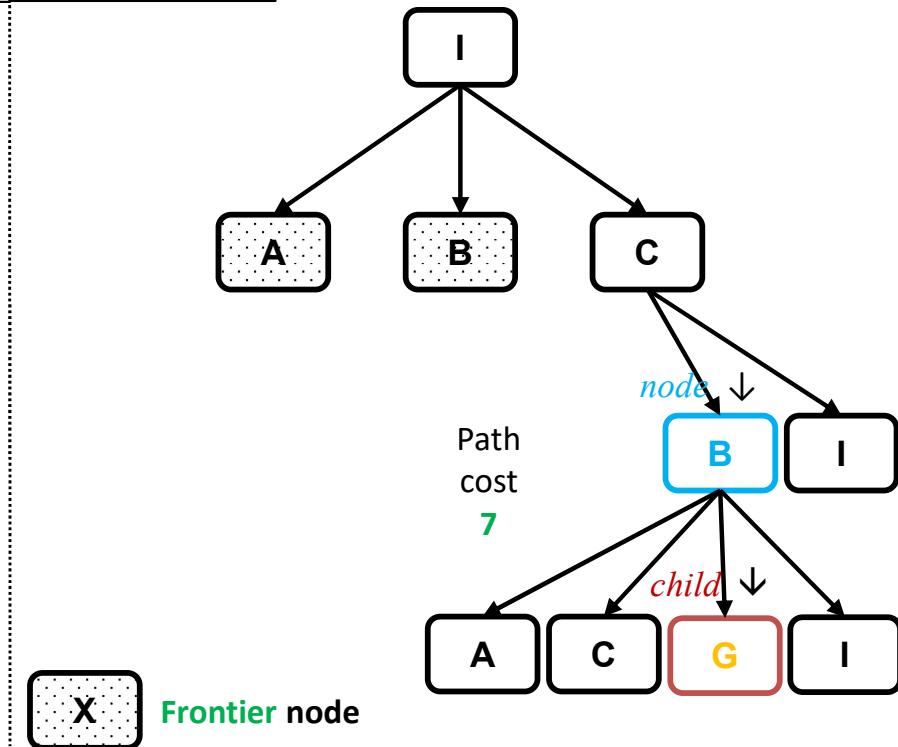
s \leftarrow *child.STATE*

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

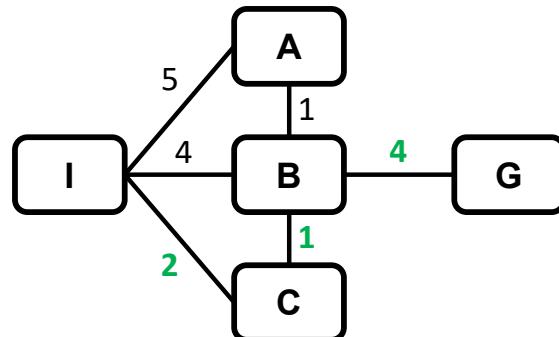
reached[*s*] \leftarrow *child*

 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

node \leftarrow POP(*frontier*)

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

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

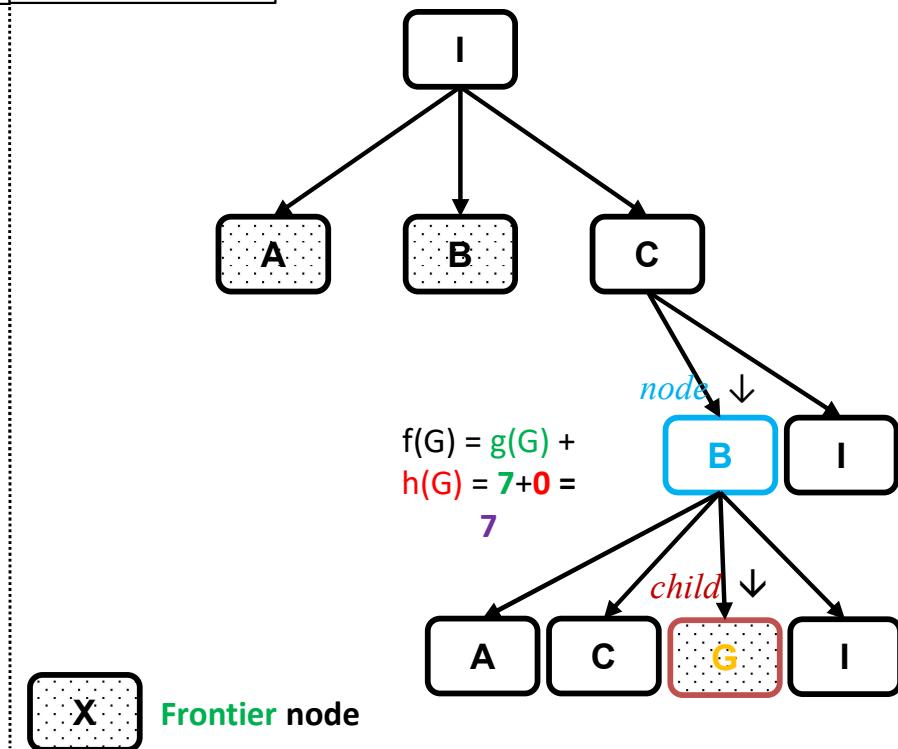
s \leftarrow *child.STATE*

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

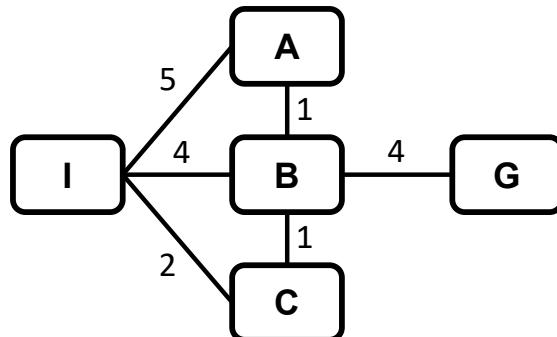
reached[s] \leftarrow *child*

 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*

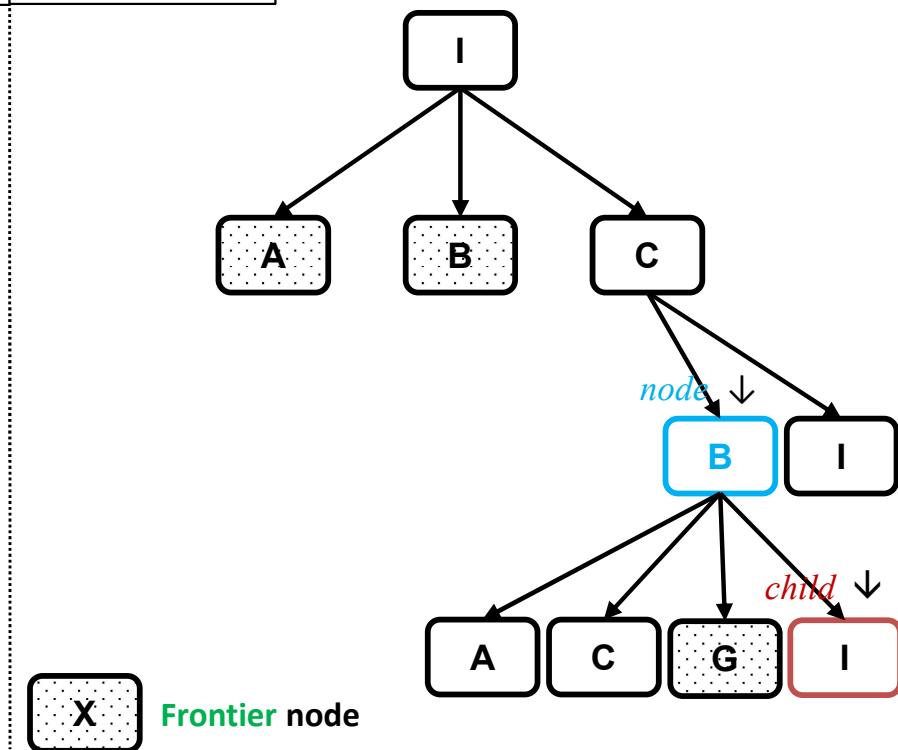
 add *child* to *frontier*

return failure

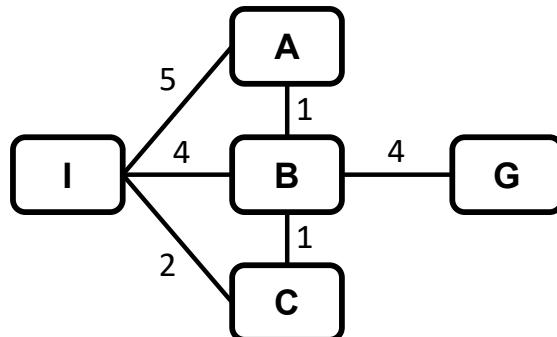
State Space Graph		Frontier / Reached
Algorithm	Search Tree	

Frontier	Parent	B	I	I			
	Node	G	B	A			
	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		



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				
$f(\text{Node})$	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**

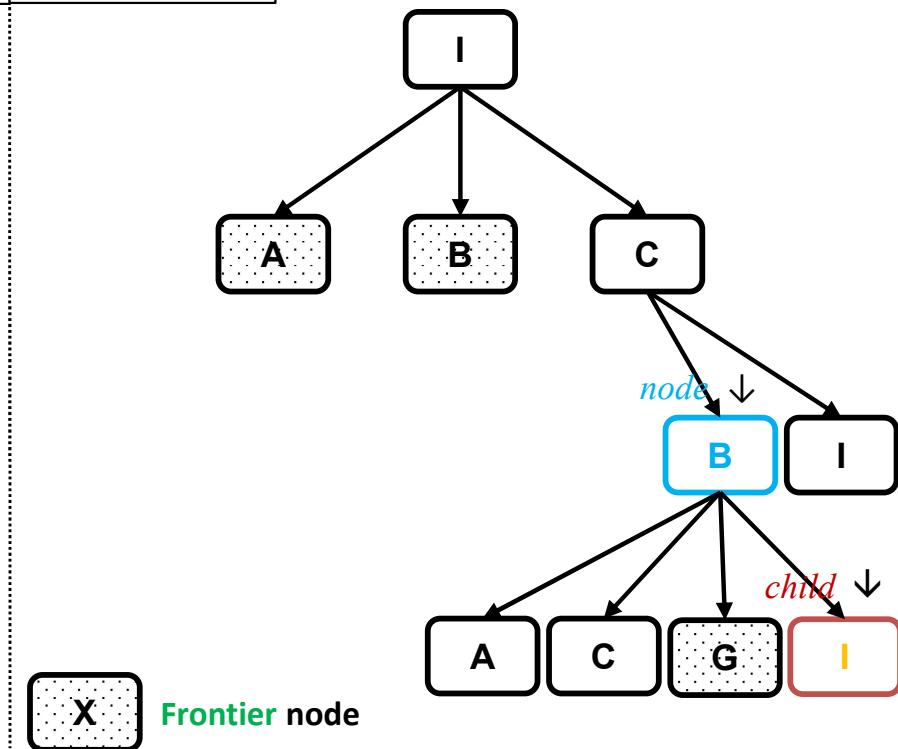
s \leftarrow *child.STATE*

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

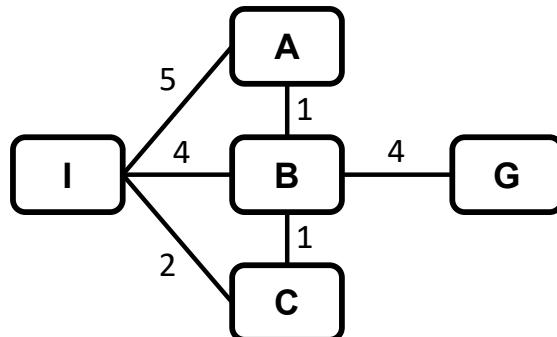
reached[*s*] \leftarrow *child*

 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**

node \leftarrow POP(*frontier*)

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

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

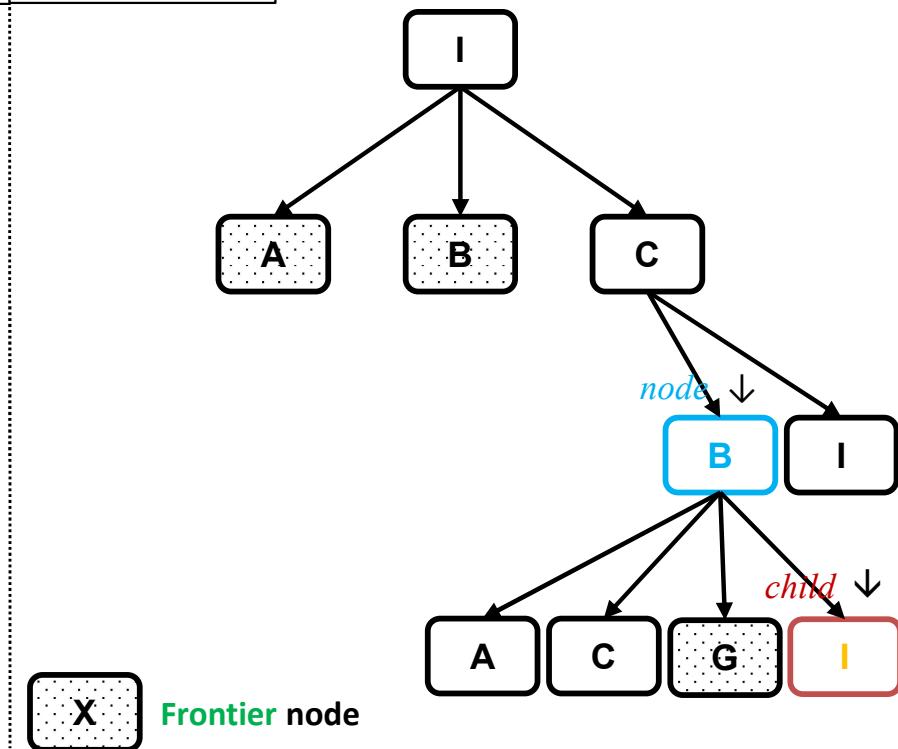
s \leftarrow *child.STATE*

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

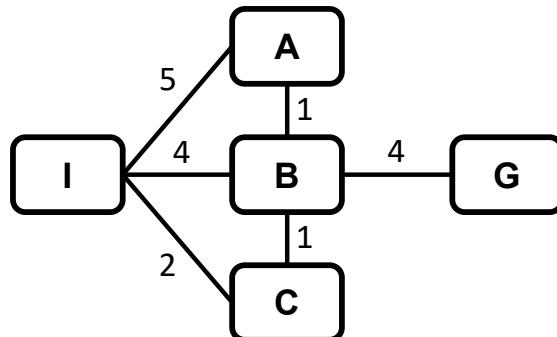
reached[*s*] \leftarrow *child*

 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**

node \leftarrow POP(*frontier*)

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

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

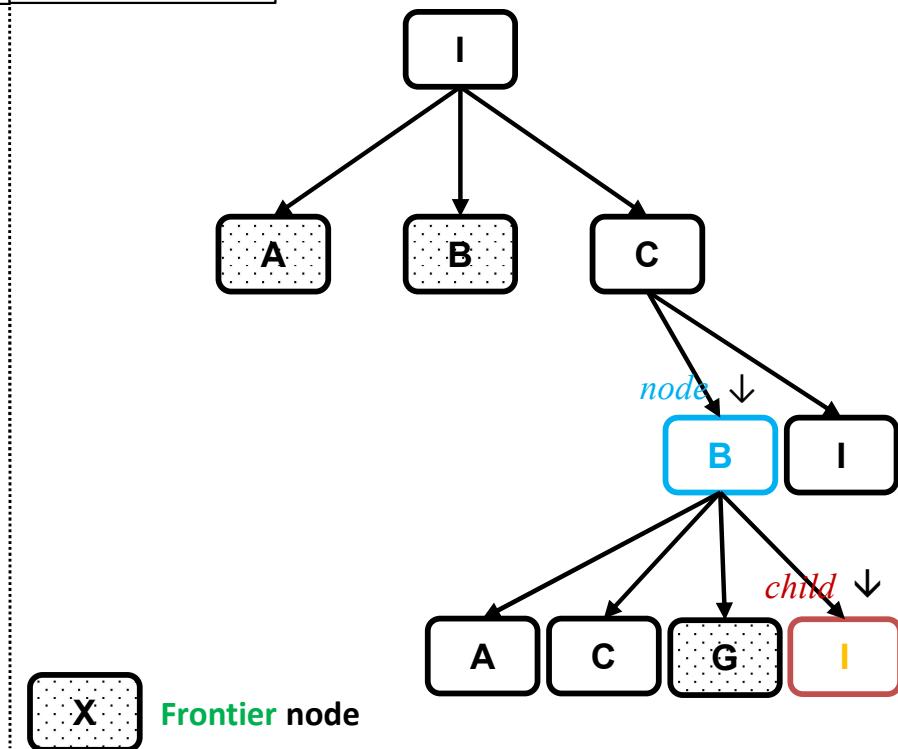
s \leftarrow *child.STATE*

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

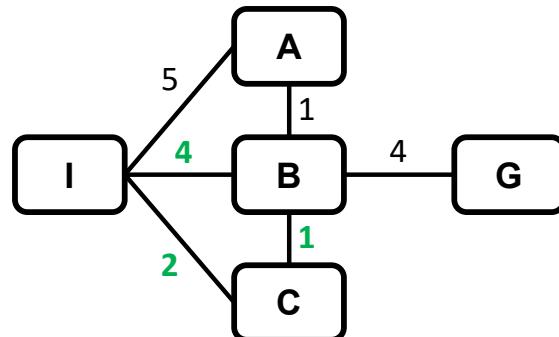
reached[*s*] \leftarrow *child*

 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
```

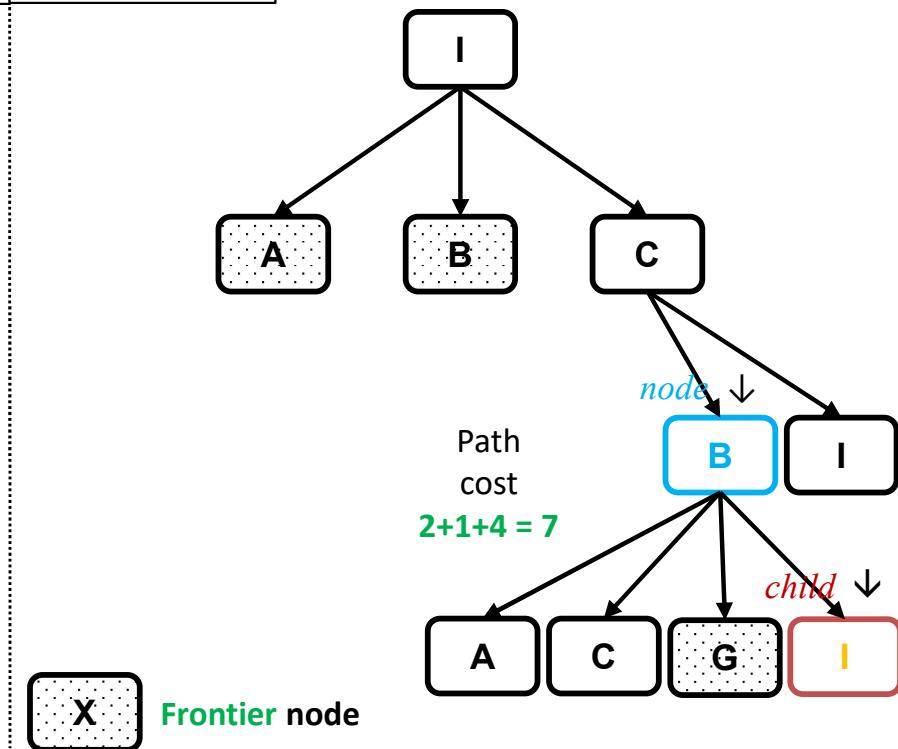
```
            s ← child.STATE
```

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

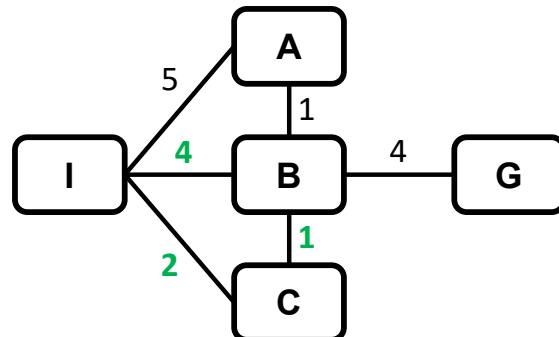
```
                reached[s] ← child
```

```
                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 \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

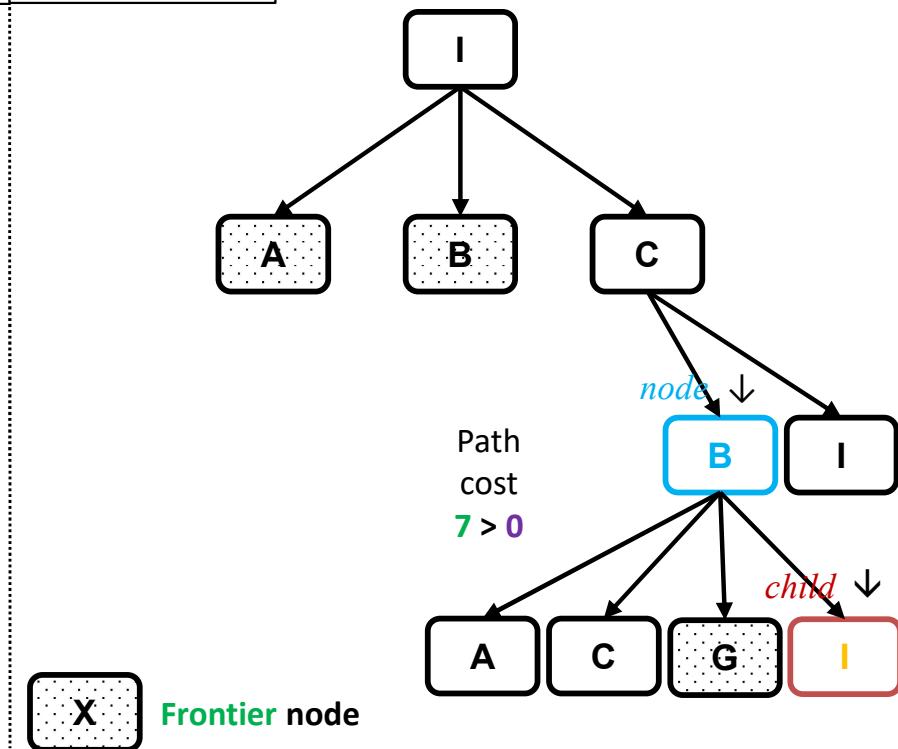
s \leftarrow *child.STATE*

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

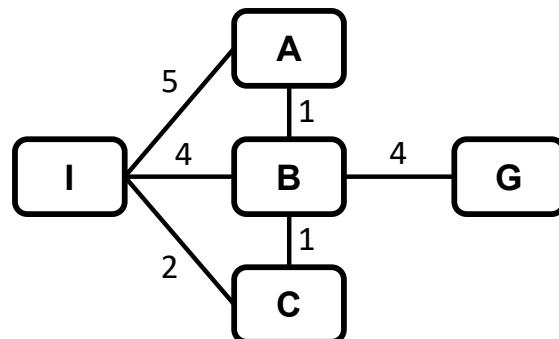
reached[s] <- child

 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**

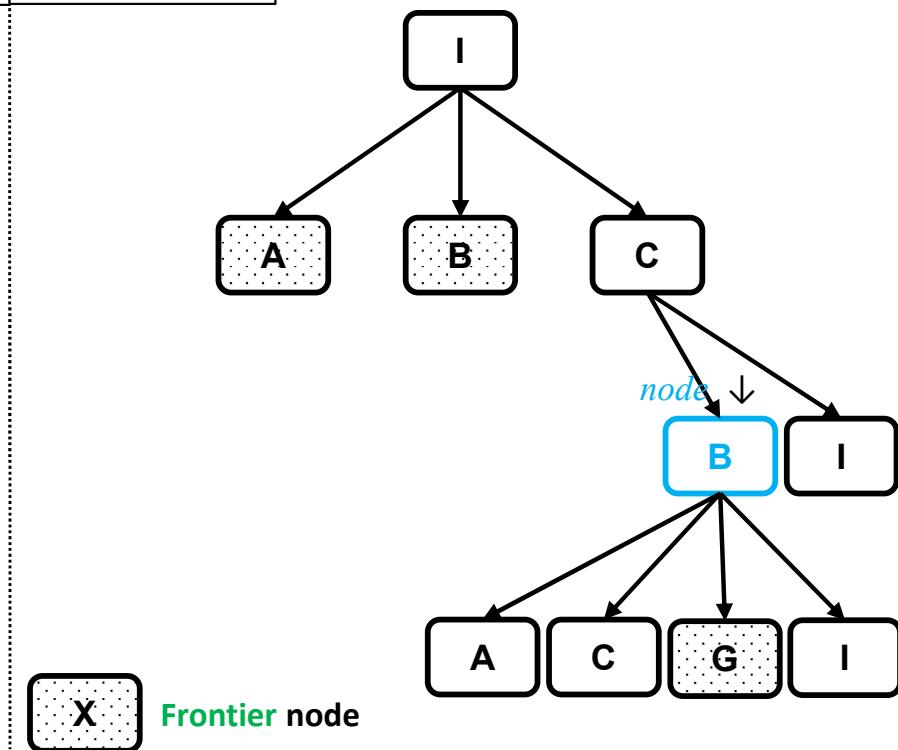
s \leftarrow *child.STATE*

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

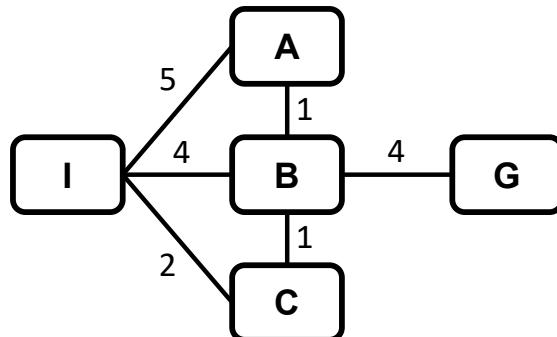
reached[s] \leftarrow *child*

 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				
$f(\text{Node})$	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**

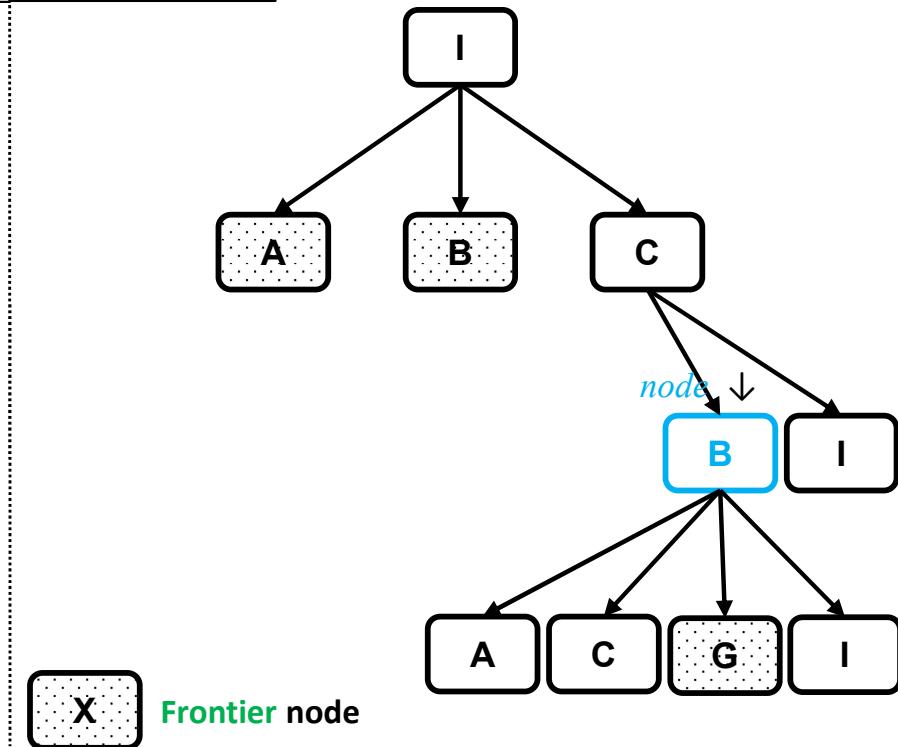
s \leftarrow *child.STATE*

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

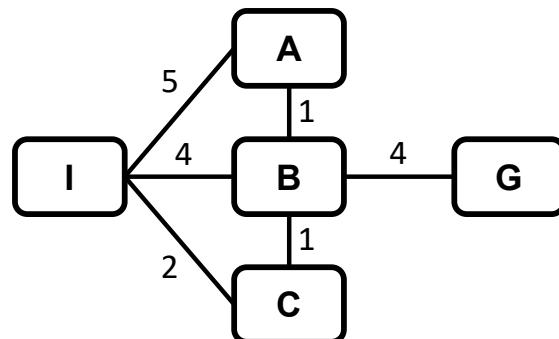
reached[*s*] \leftarrow *child*

 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*

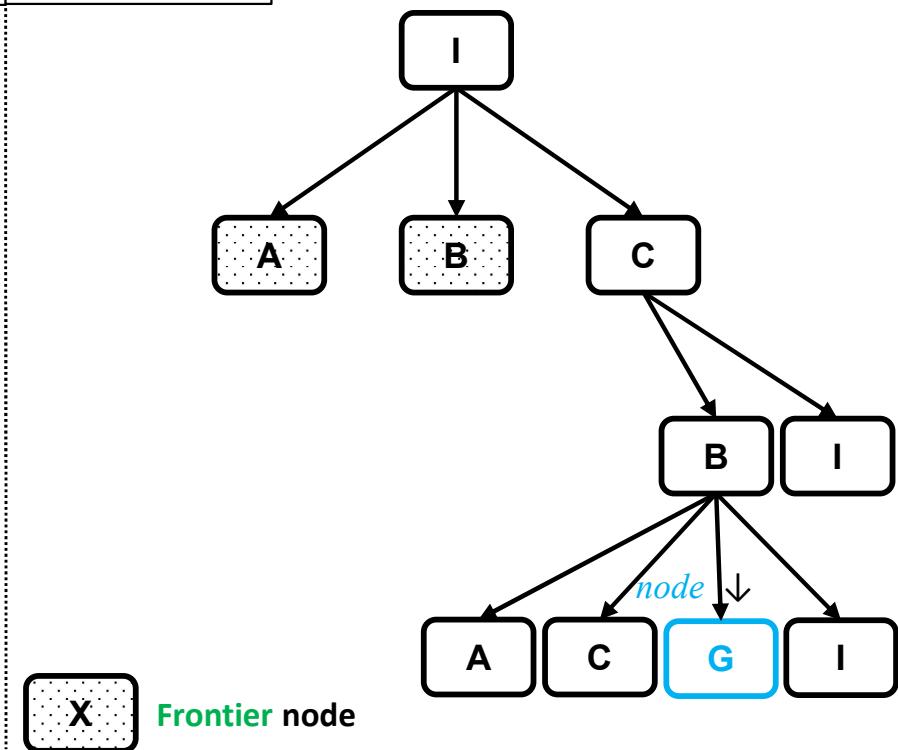
 add *child* to *frontier*

return failure

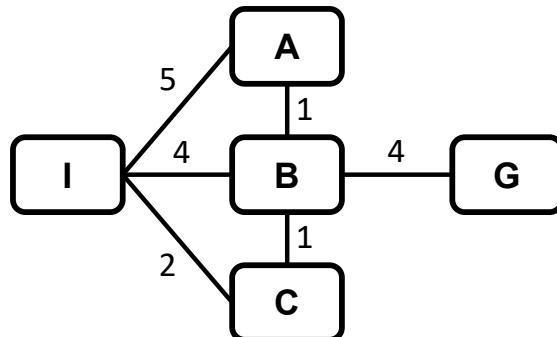
Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

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*

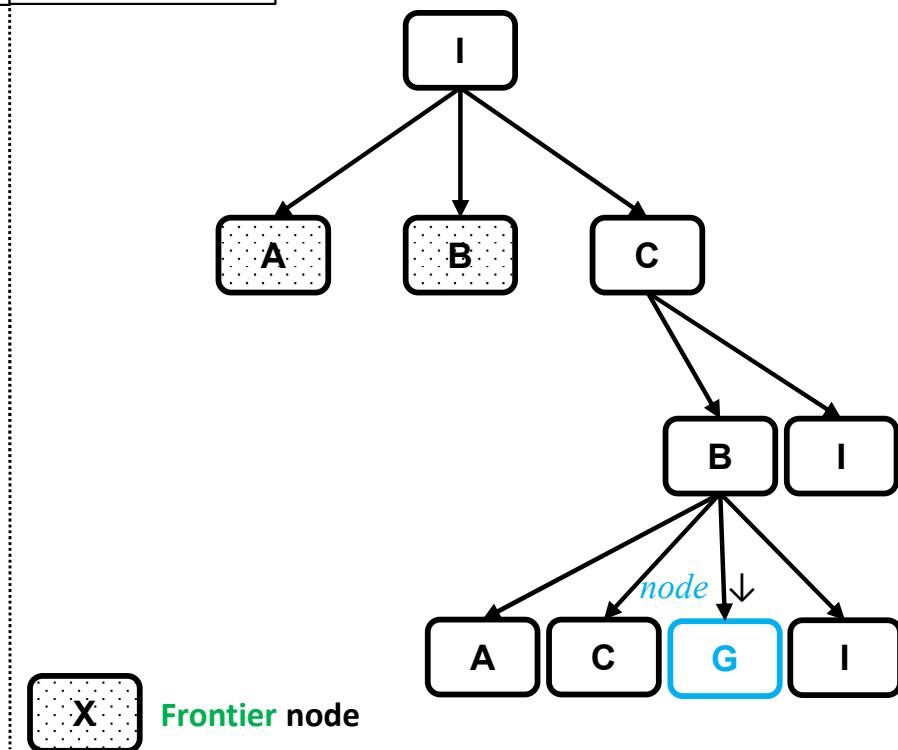
 add *child* to *frontier*

return failure

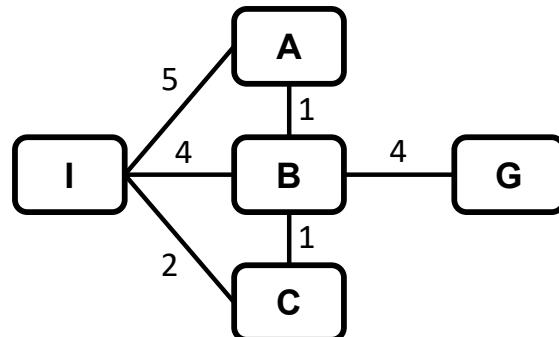
State Space Graph	Frontier / Reached
Algorithm	Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	<i>f</i> (Node)	7	7				

Reached	Parent	---	I	C	I	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

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*

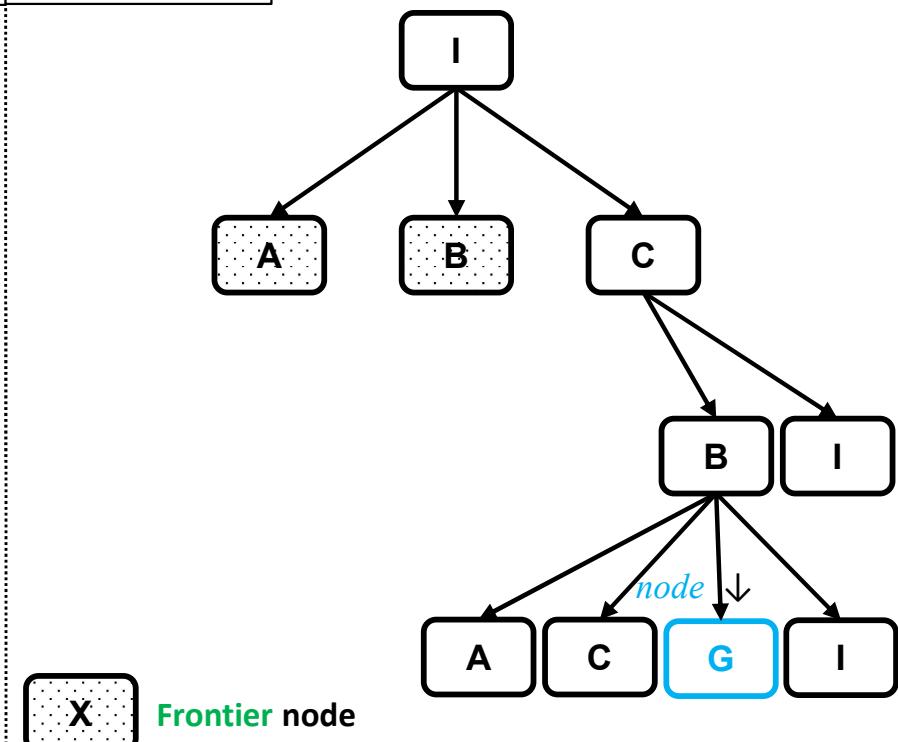
 add *child* to *frontier*

return failure

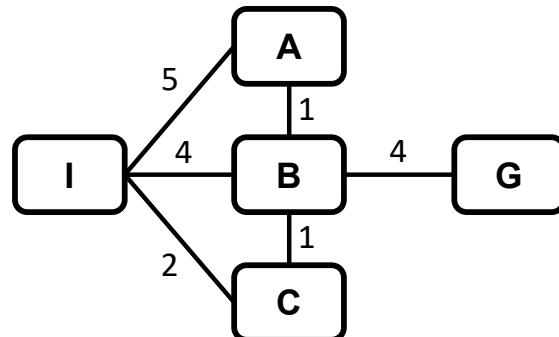
Frontier	Parent	I	I				
Node	B	A					
<i>f</i> (Node)	7	7					

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
$h(\text{State})$	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

State Space Graph		Frontier / Reached
Algorithm		Search Tree

Frontier	Parent	I	I				
	Node	B	A				
	f(Node)	7	7				

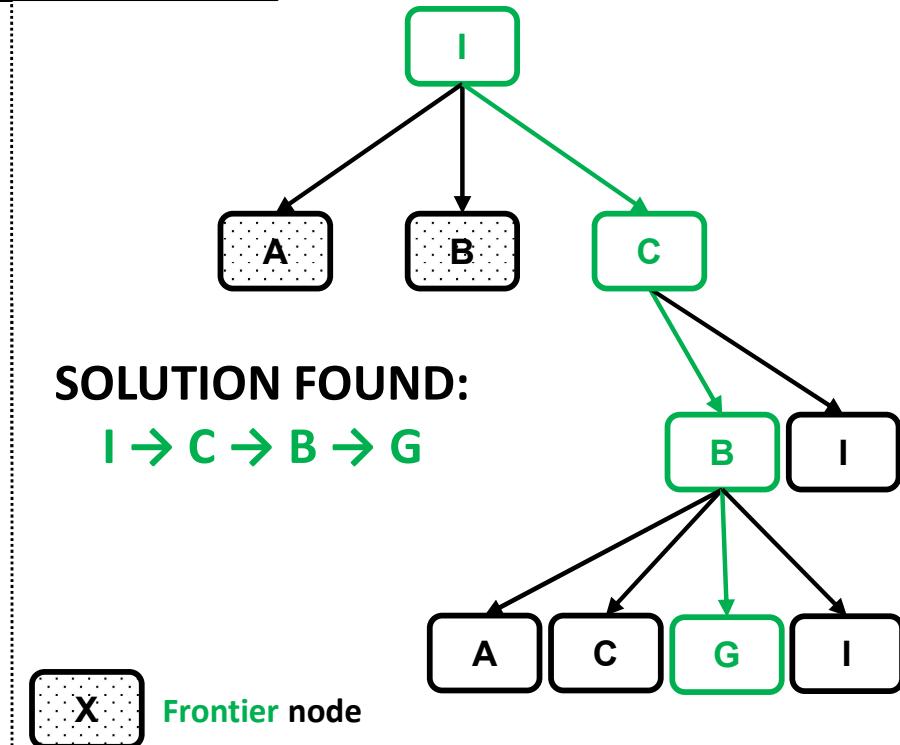
Reached	Parent	---	I	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	5	3	2	7		

Algorithm

Search Tree

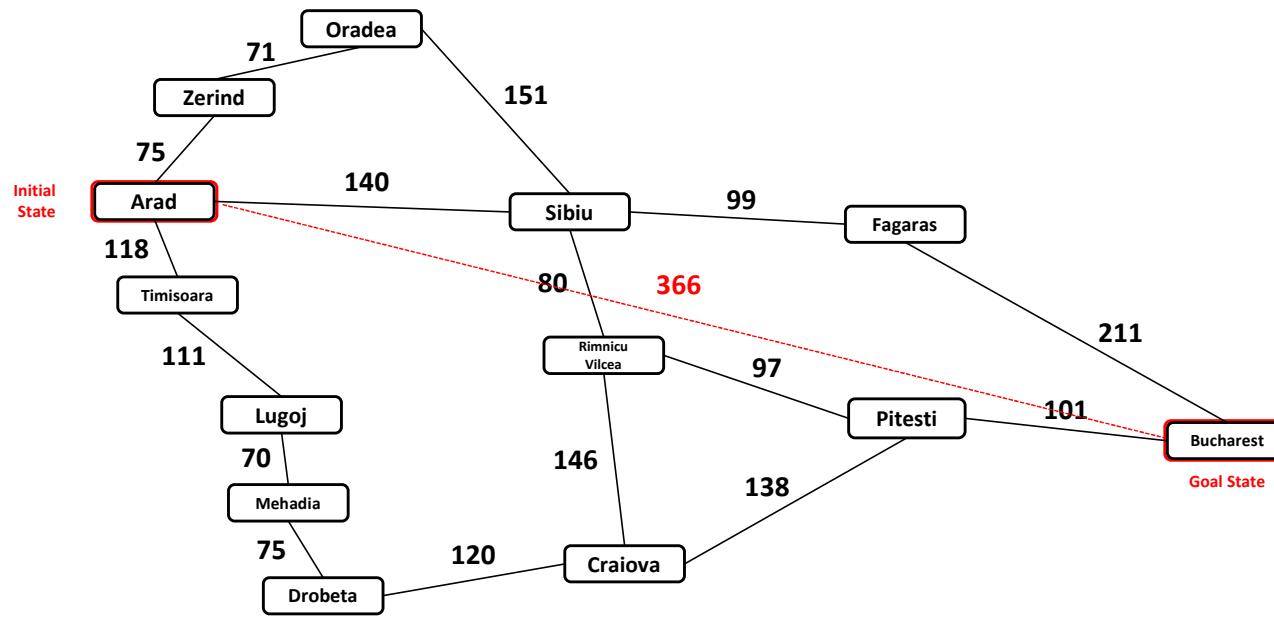
SOLUTION FOUND:

I \rightarrow **C** \rightarrow **B** \rightarrow **G**



What Made A* Work Well?

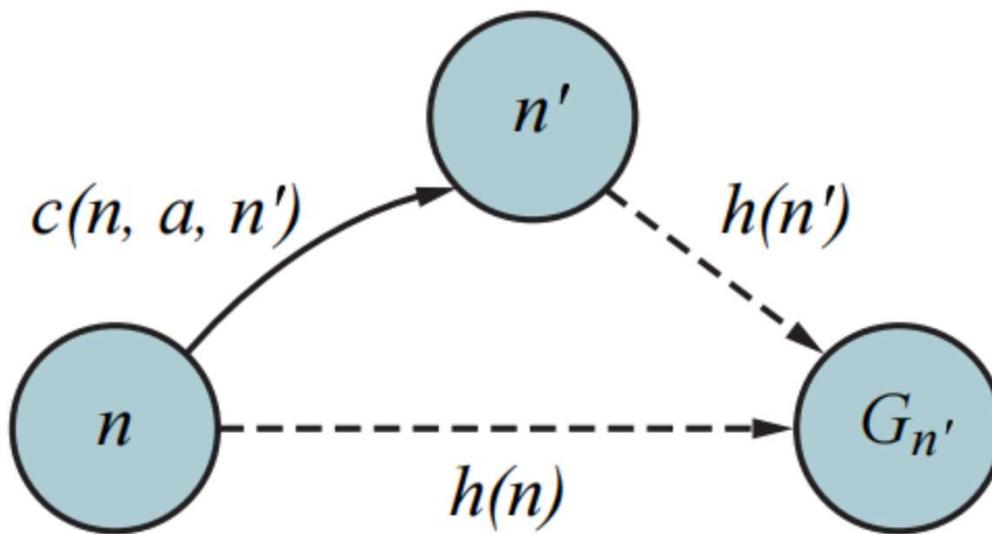
- Straight-line heuristics is **admissible**: it never overestimates the cost.



- An **admissible heuristics** is guaranteed to give you the optimal solution

What Made A* Work Well?

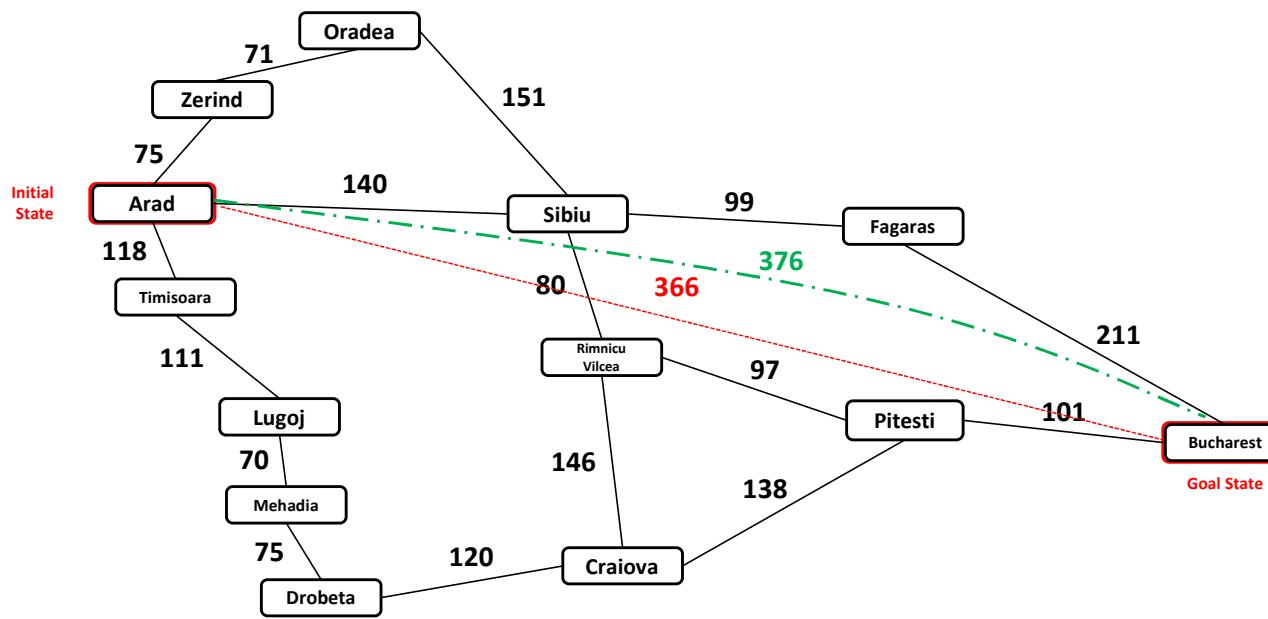
- Straight-line heuristics is **consistent**: its estimate is getting better and better as we get closer to the goal



- Every **consistent** heuristics is **admissible** heuristics, but not the other way around

Dominating Heuristics

- We can have more than one available heuristics.
For example $h_1(n)$ and $h_2(n)$.

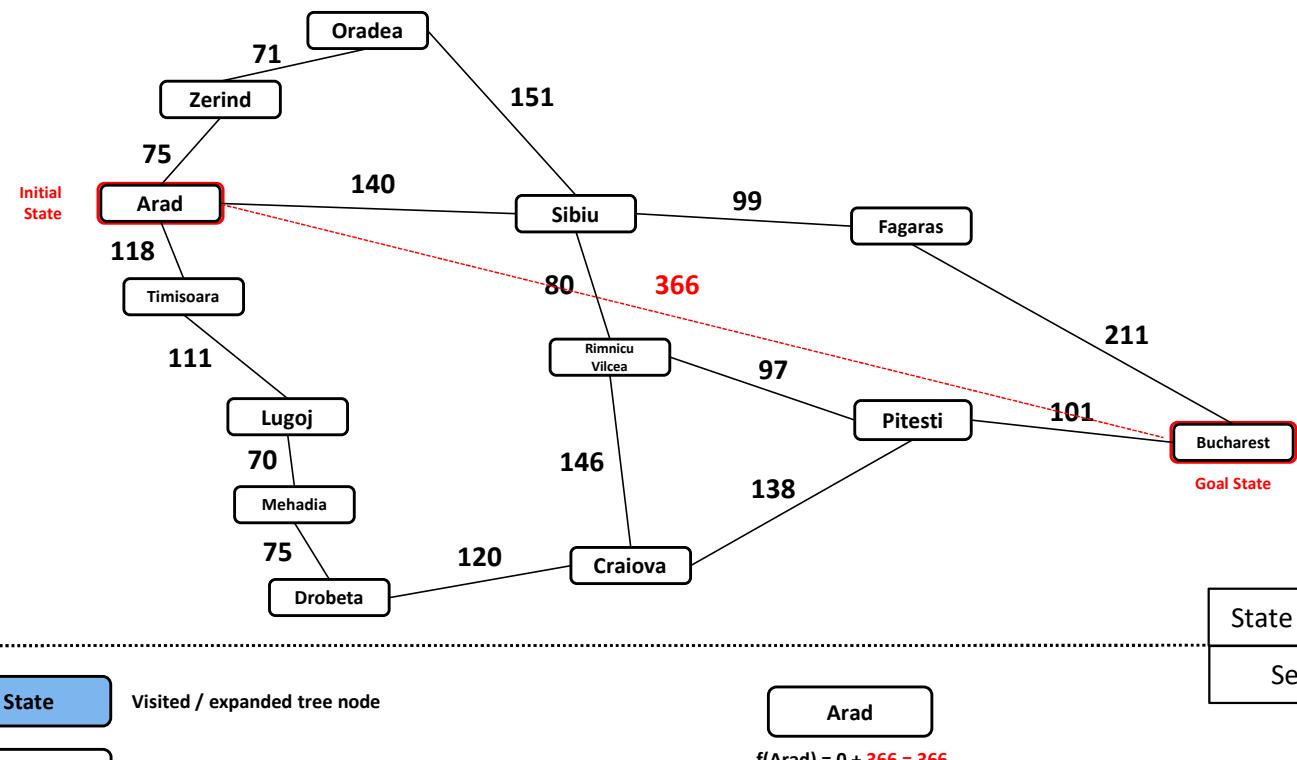


- Heuristics $h_2(n)$ estimate is closer to actual cost than $h_1(n)$. $h_2(n)$ dominates $h_1(n)$. Use $h_2(n)$.

A* Algorithm

Another Example

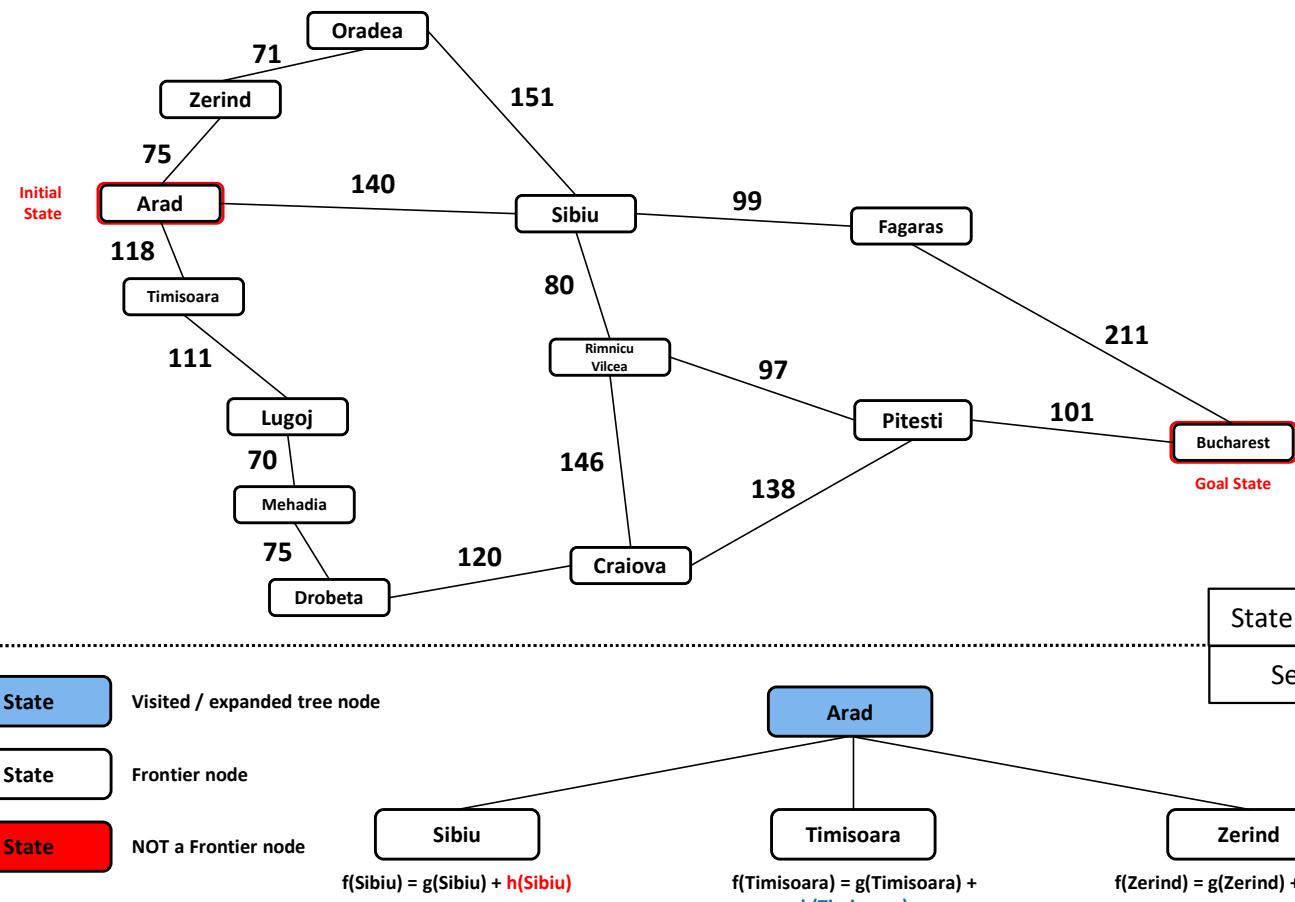
Dracula's Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):

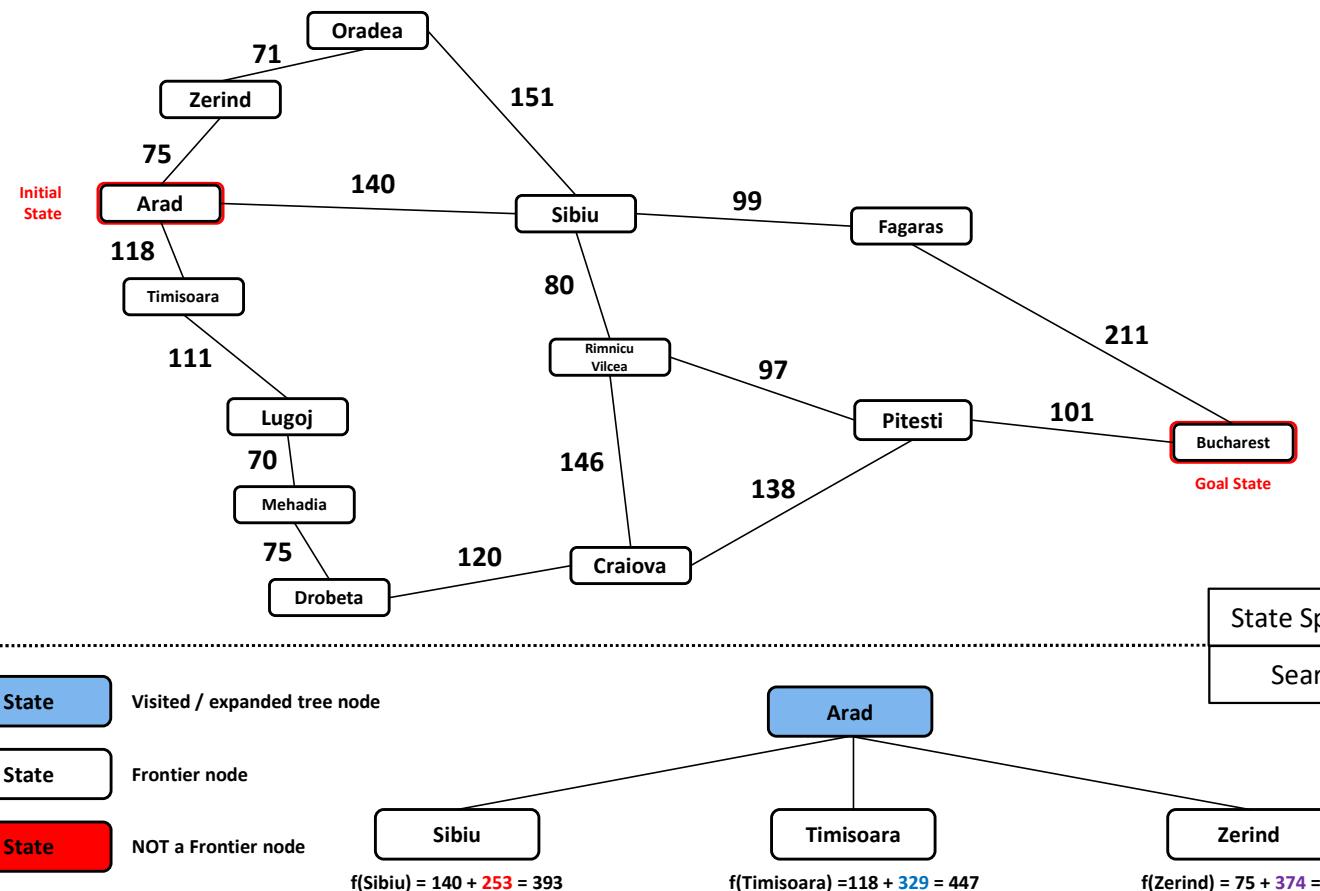
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



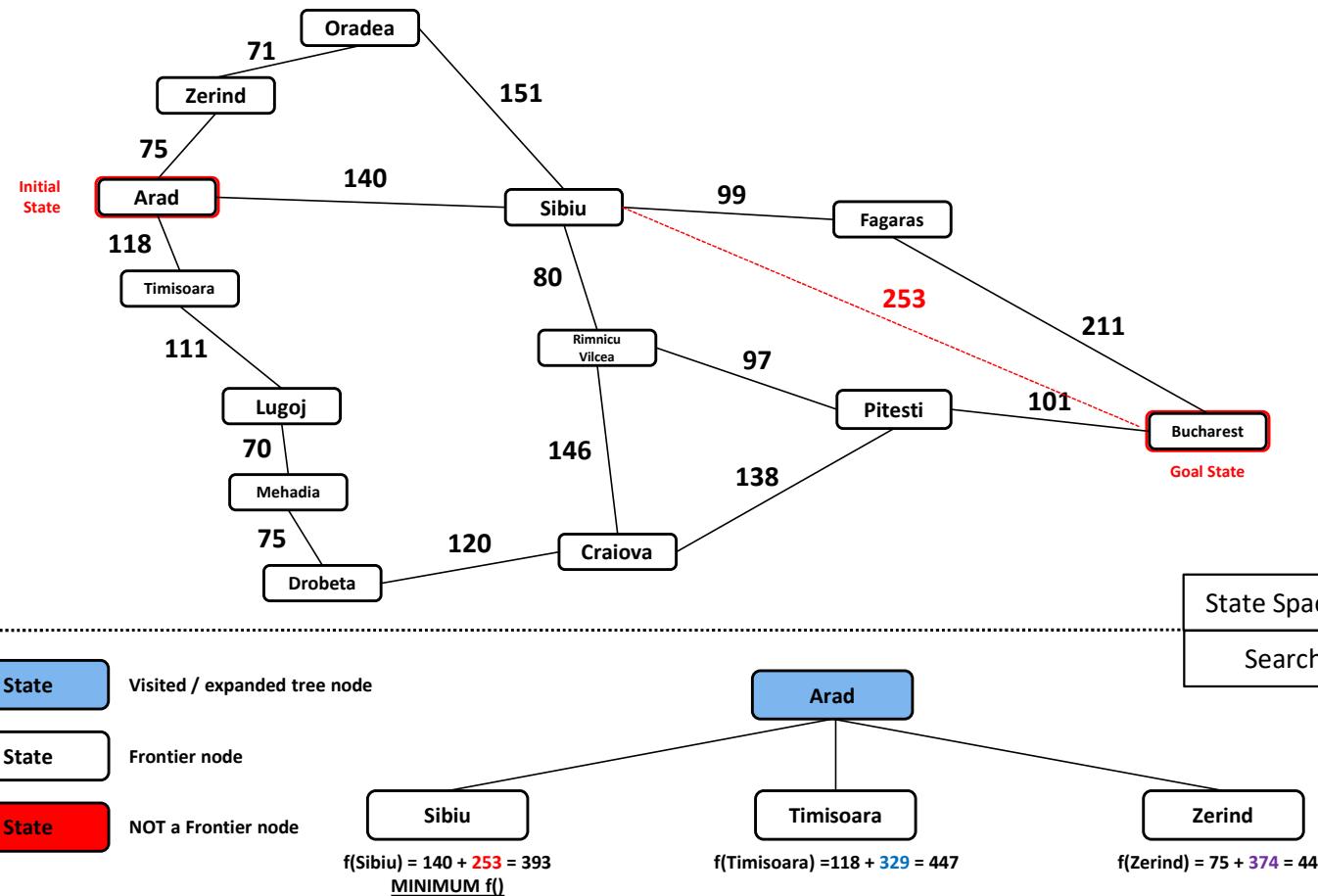
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



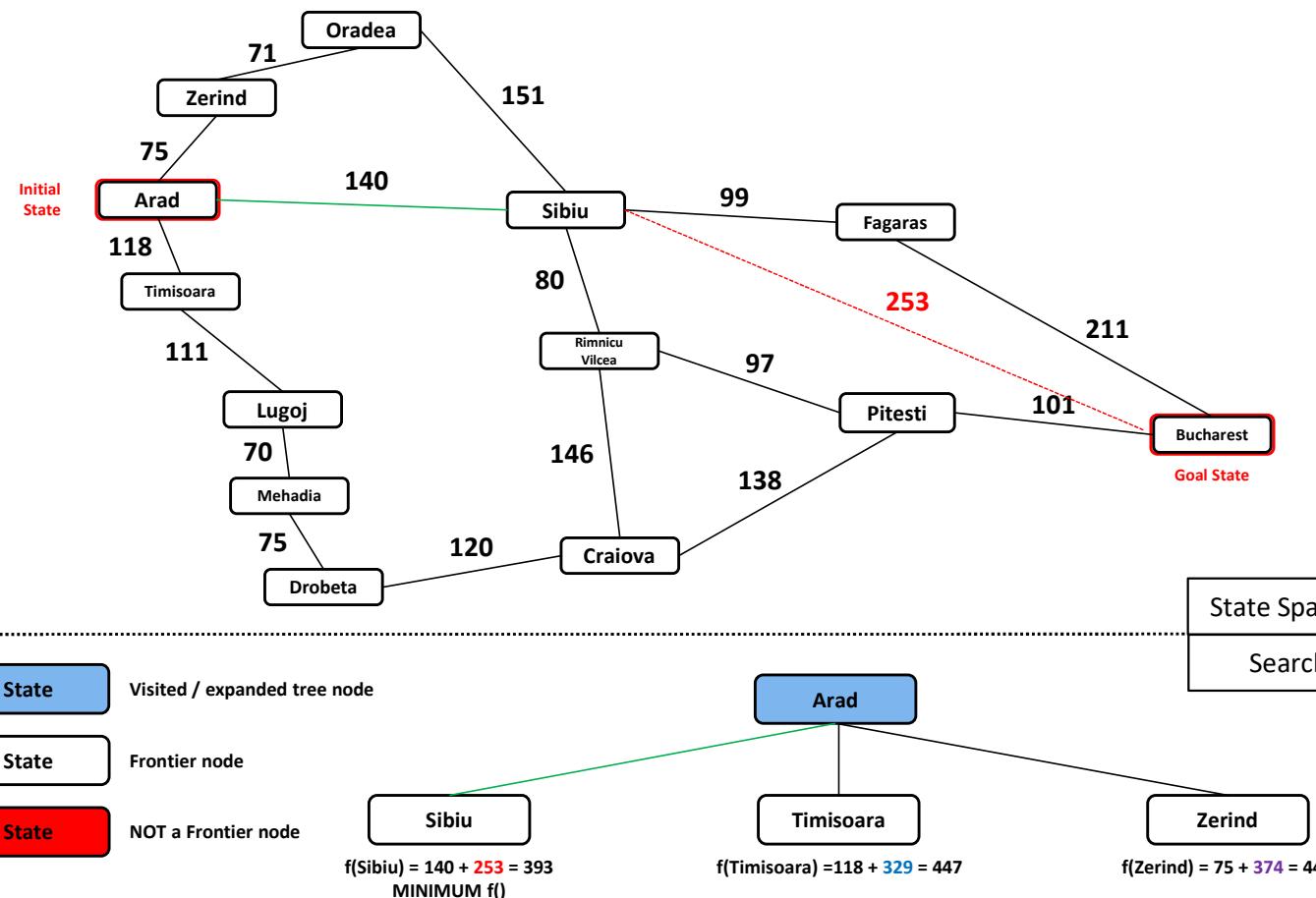
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



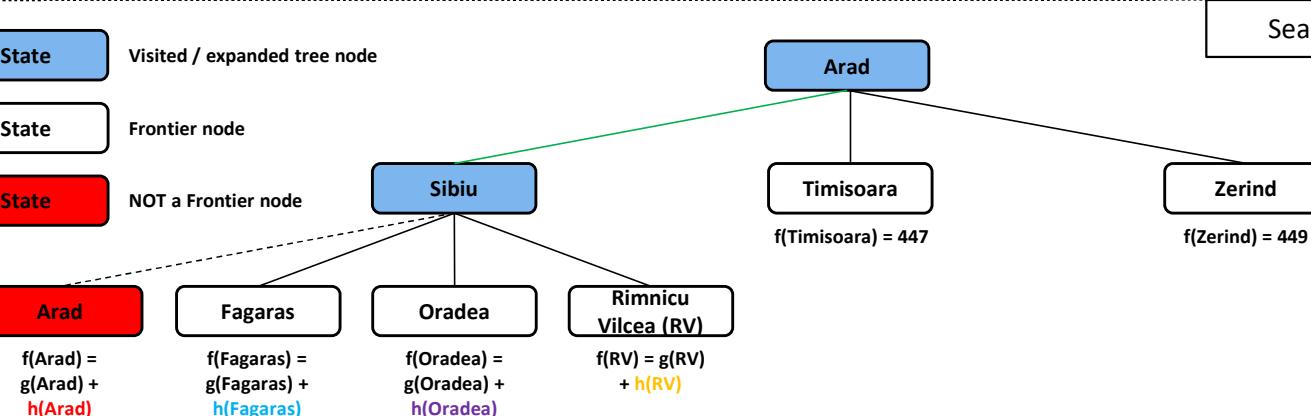
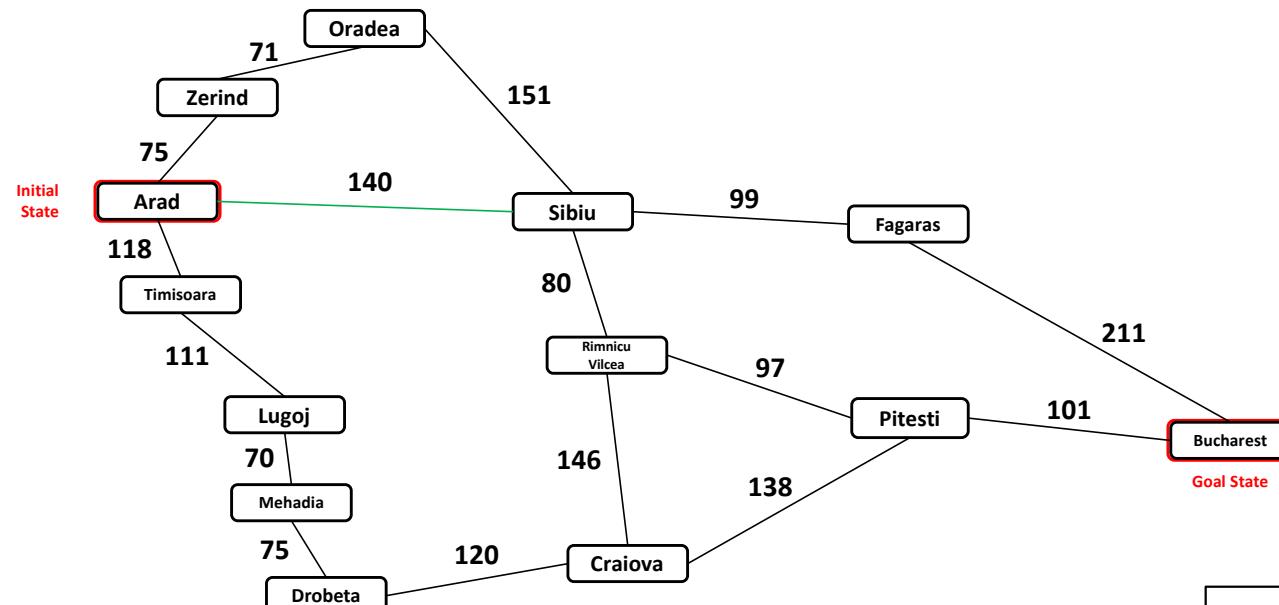
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



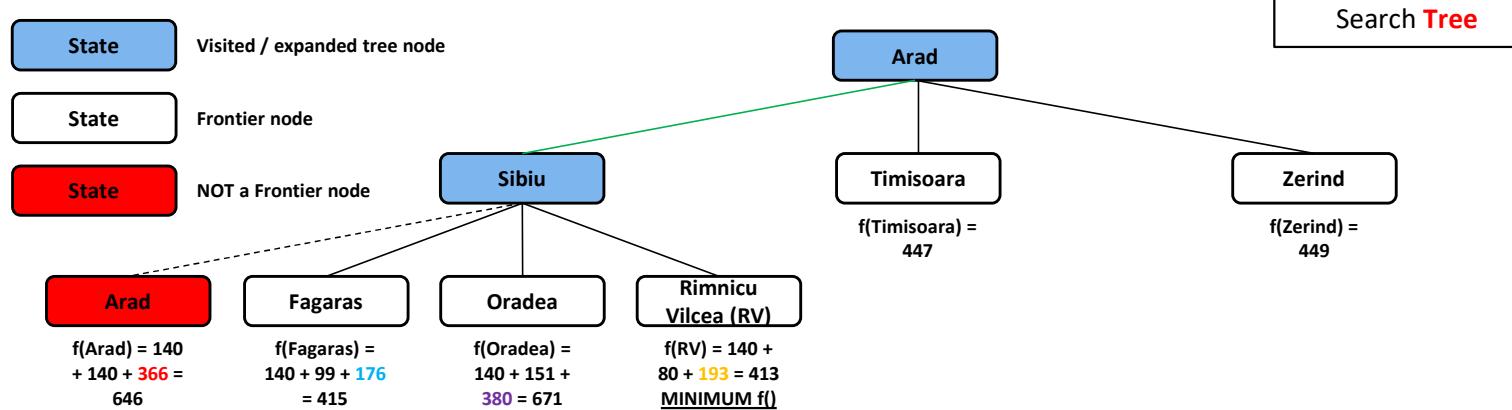
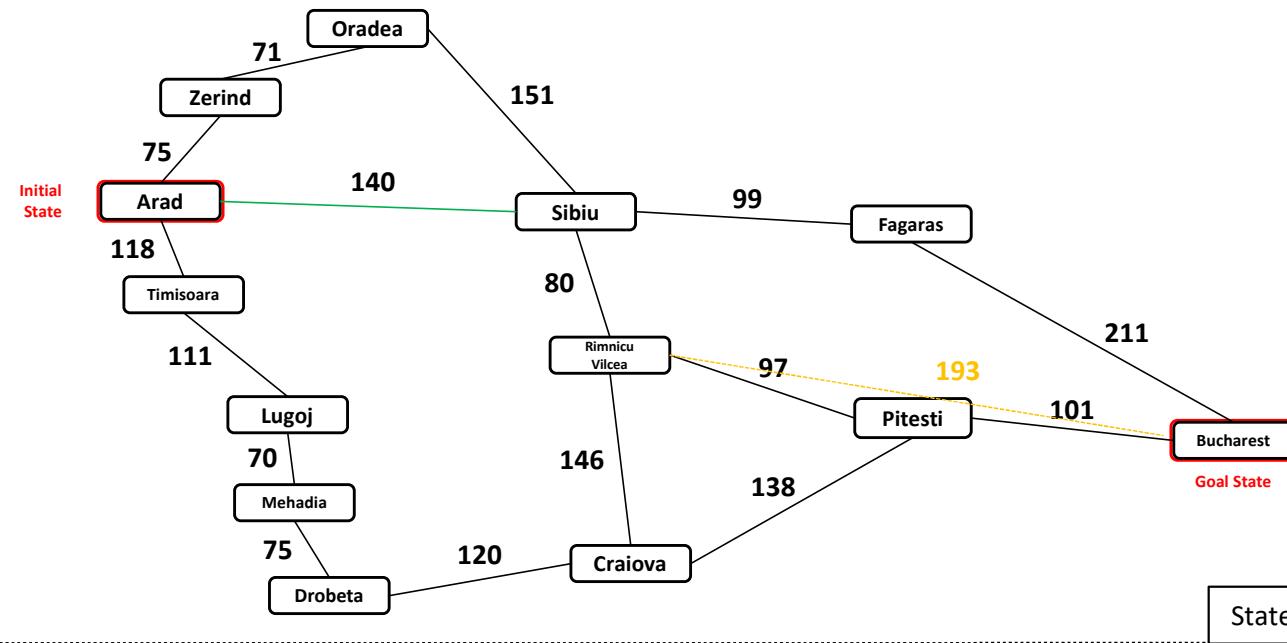
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



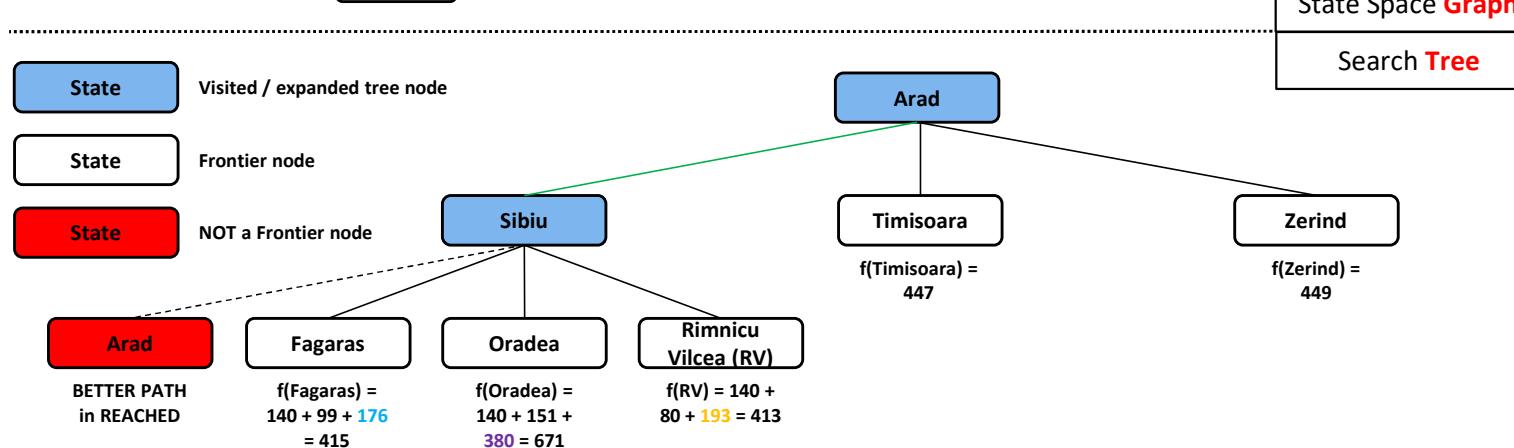
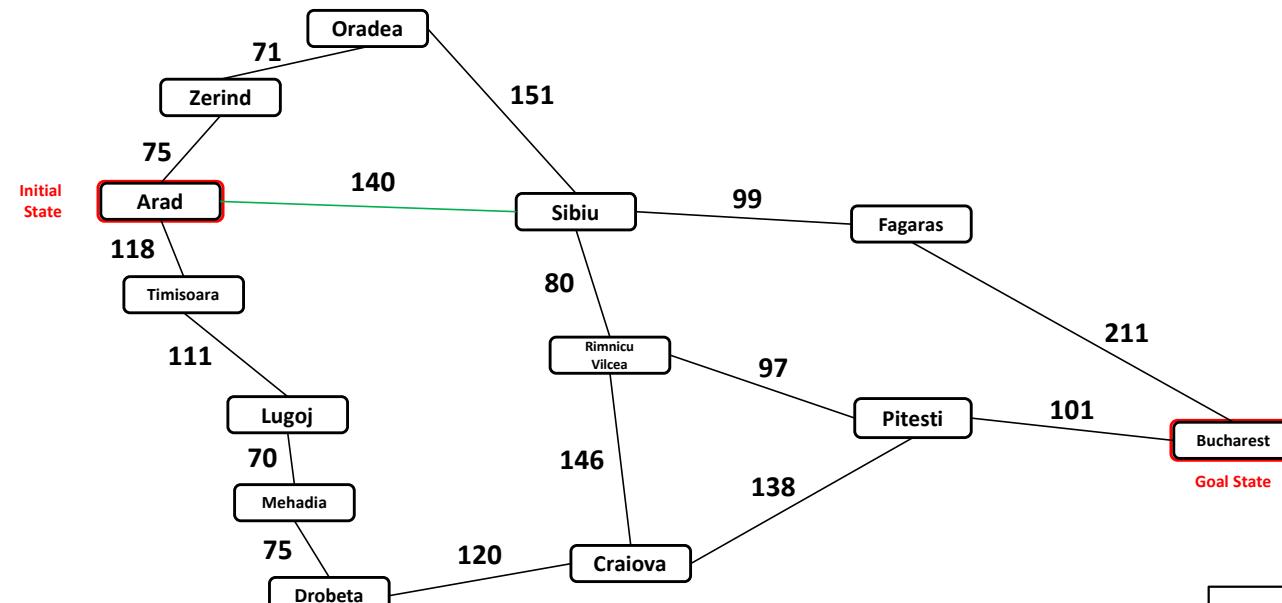
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



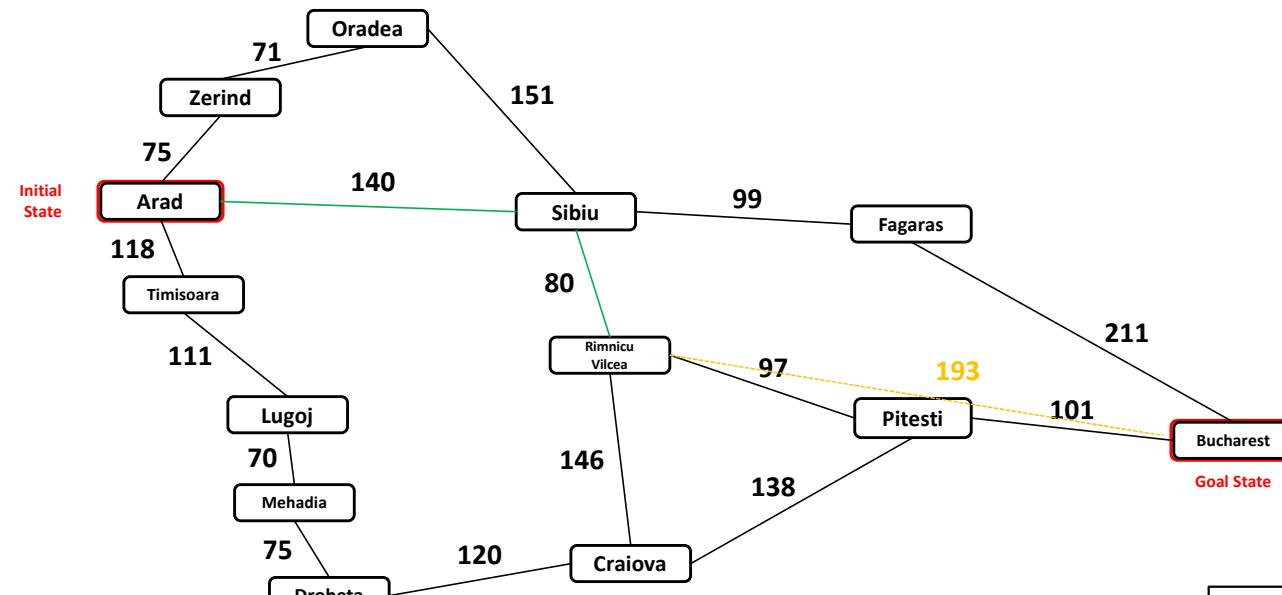
Straight-line distance to Bucharest (h(State)):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search

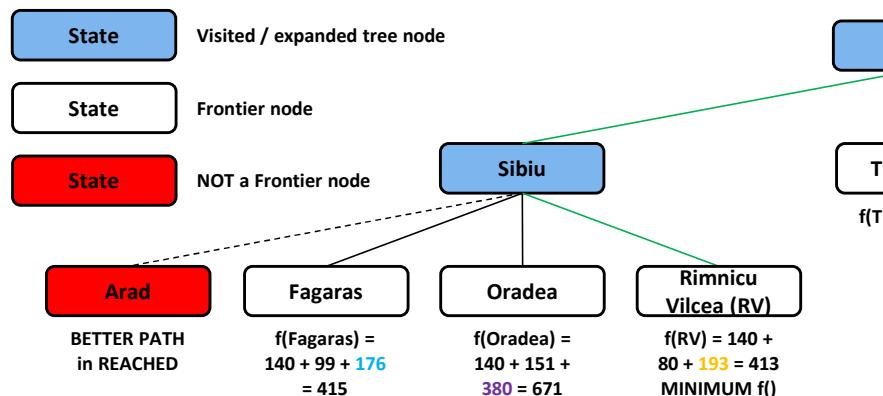


Straight-line distance to Bucharest (h(State)):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search

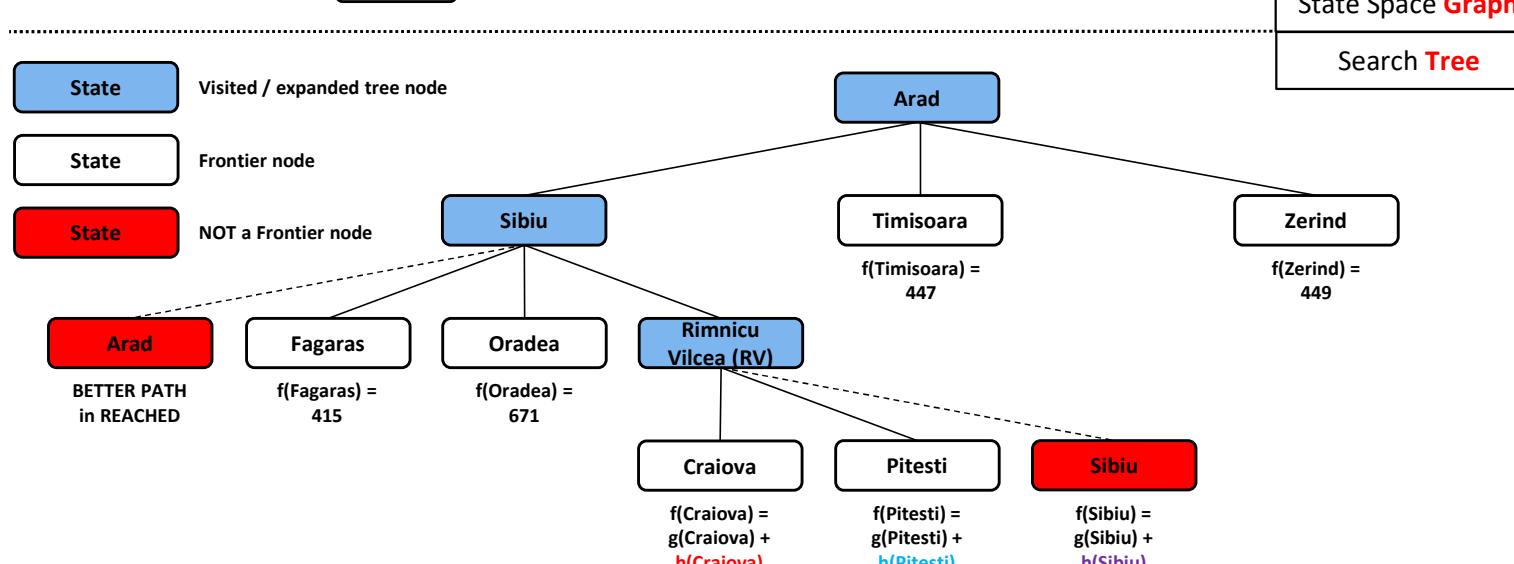
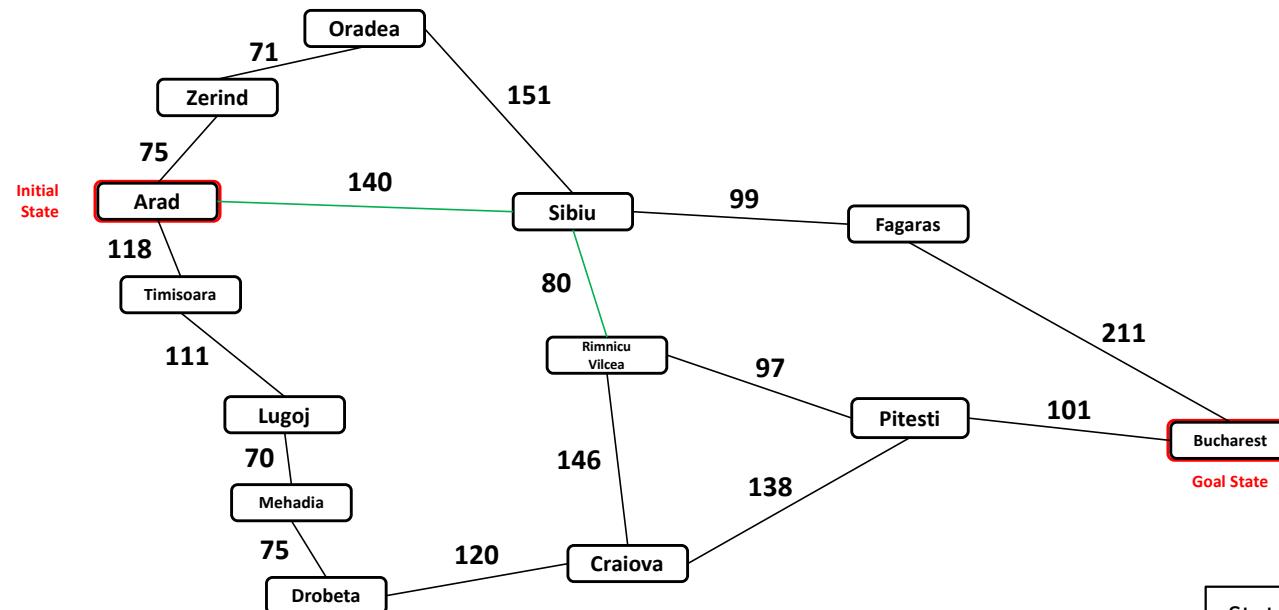


State Space Graph
Search Tree



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

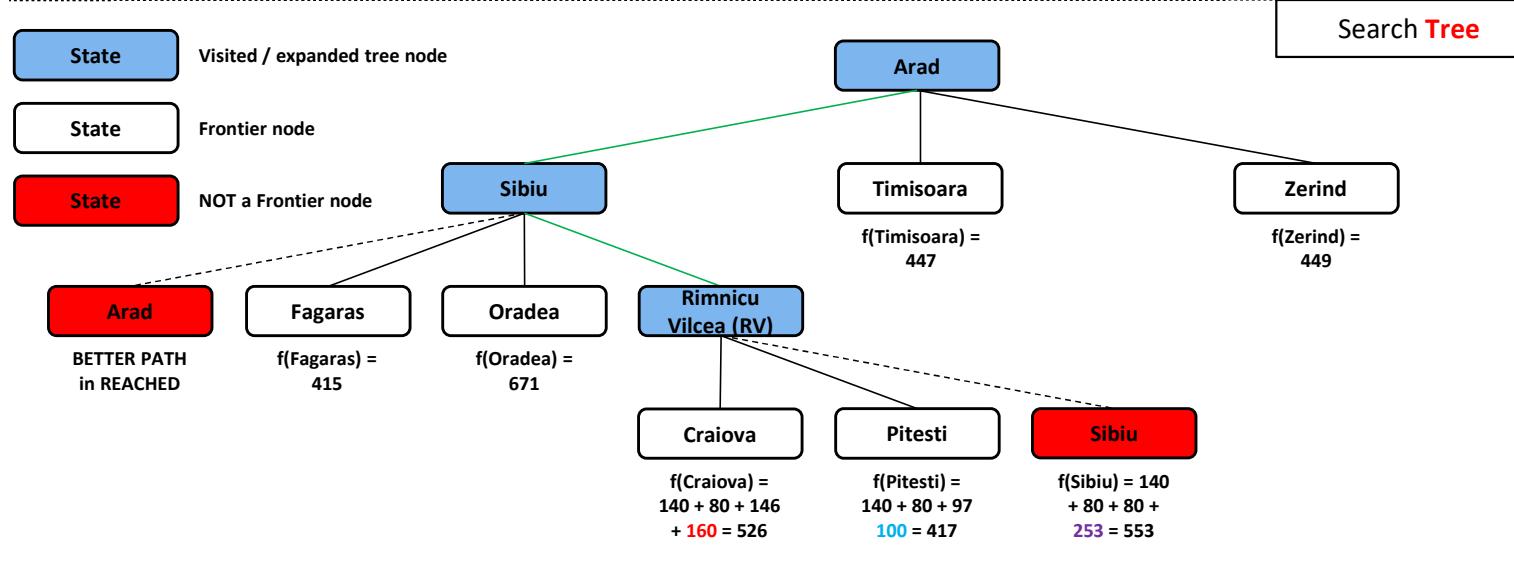
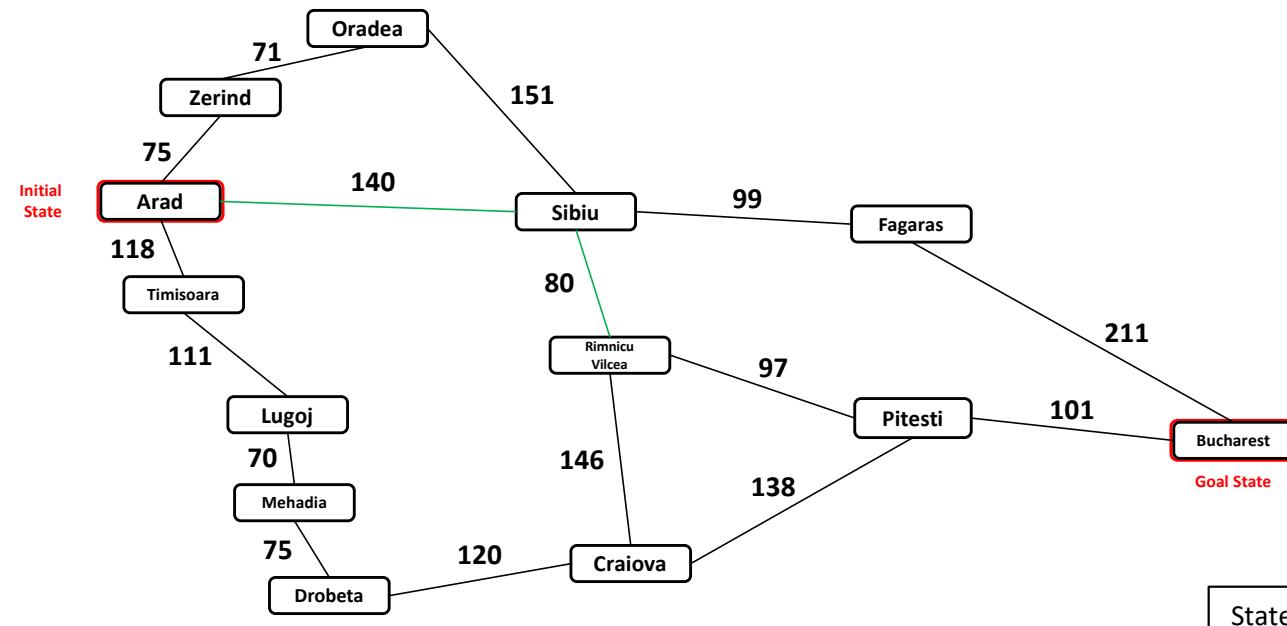
Dracula's Roadtrip: A* Search



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

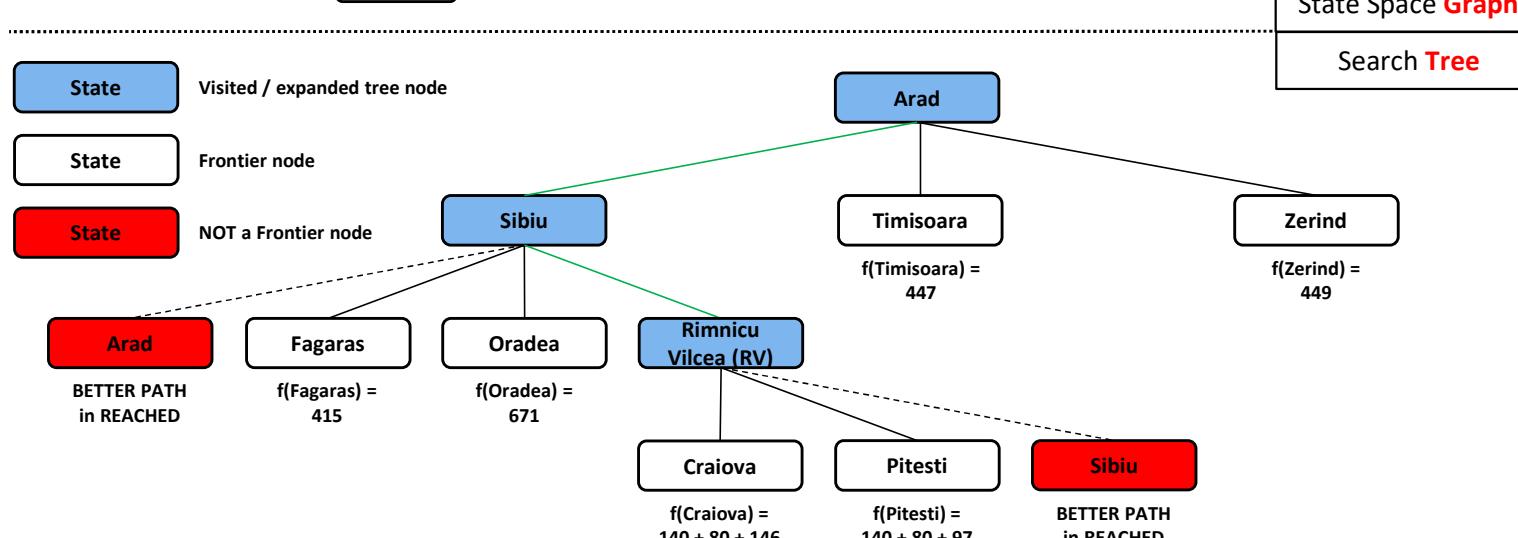
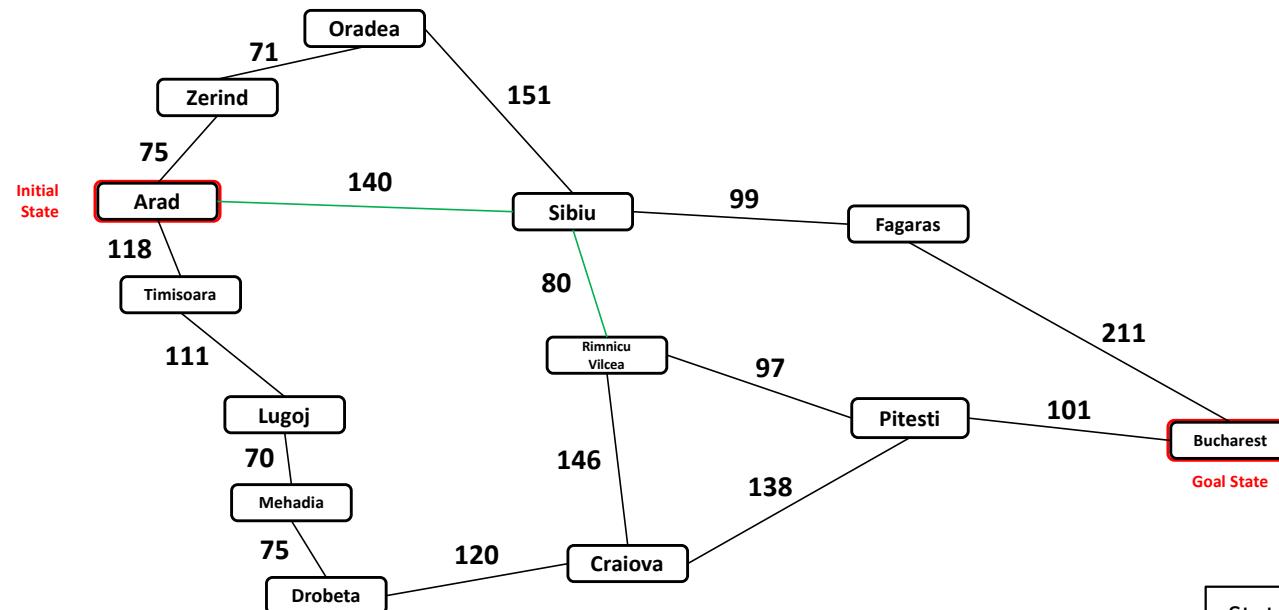
Dracula's Roadtrip: A* Search



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

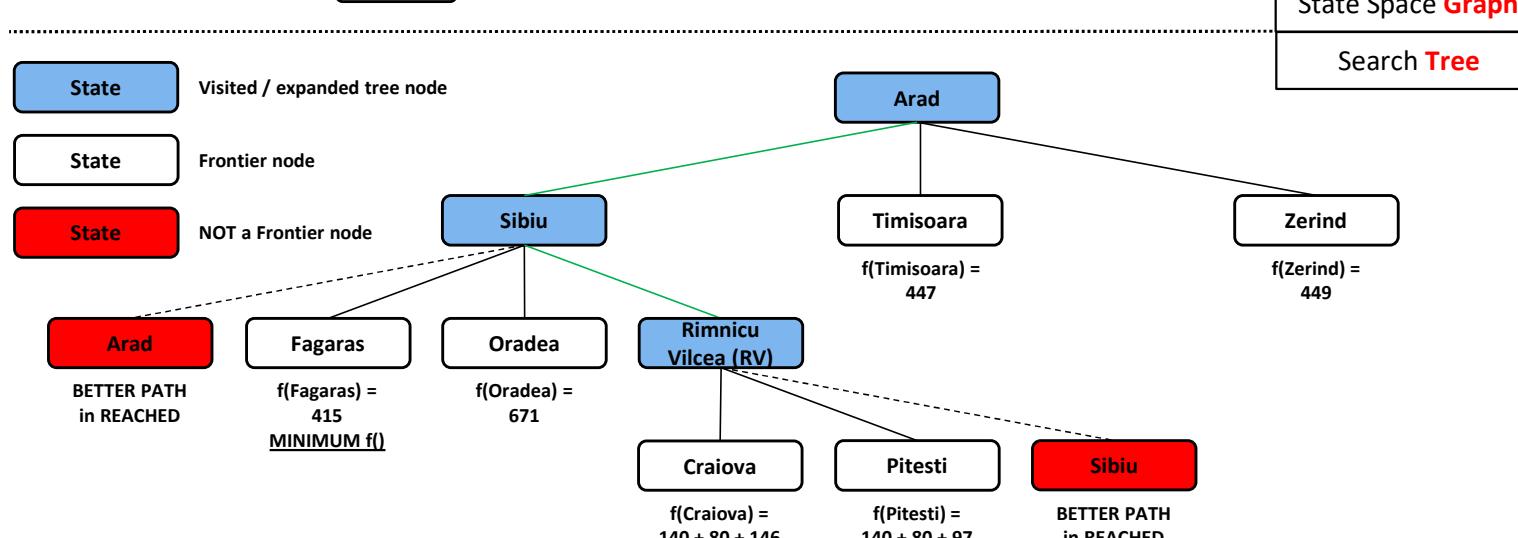
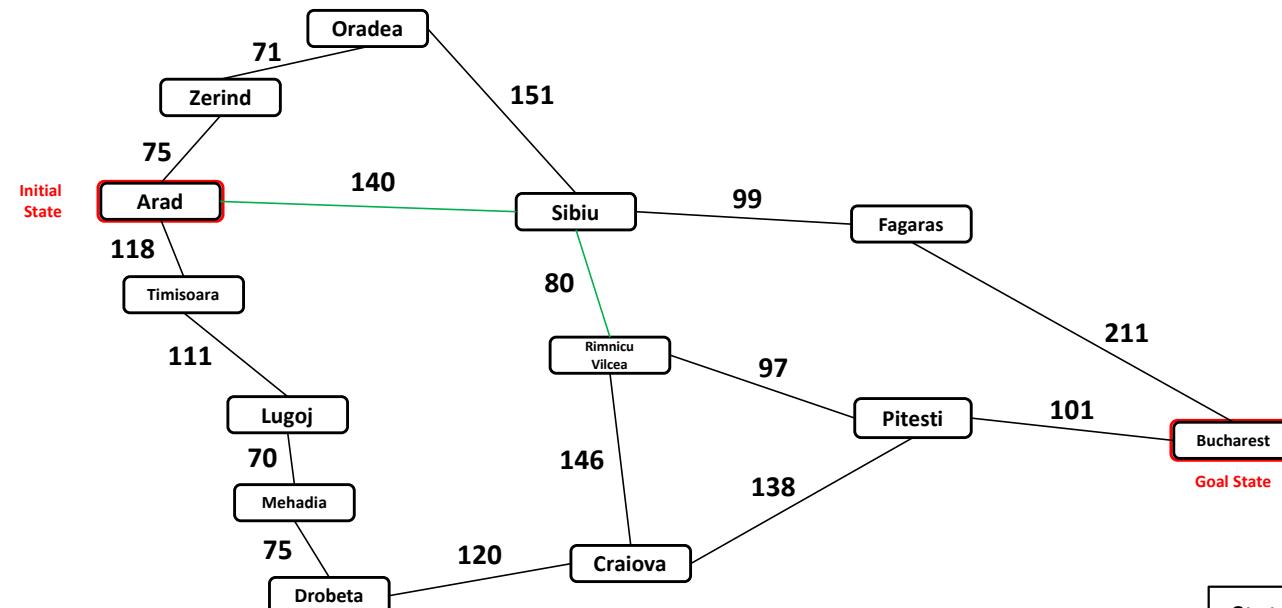
Dracula's Roadtrip: A* Search



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

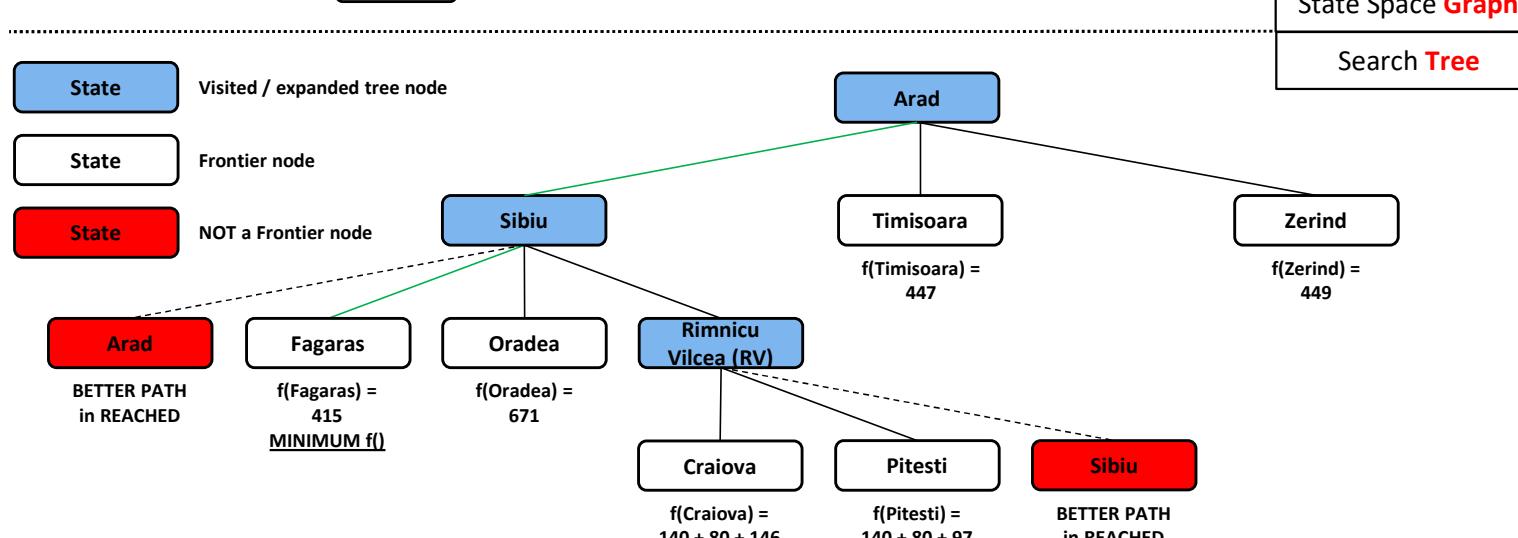
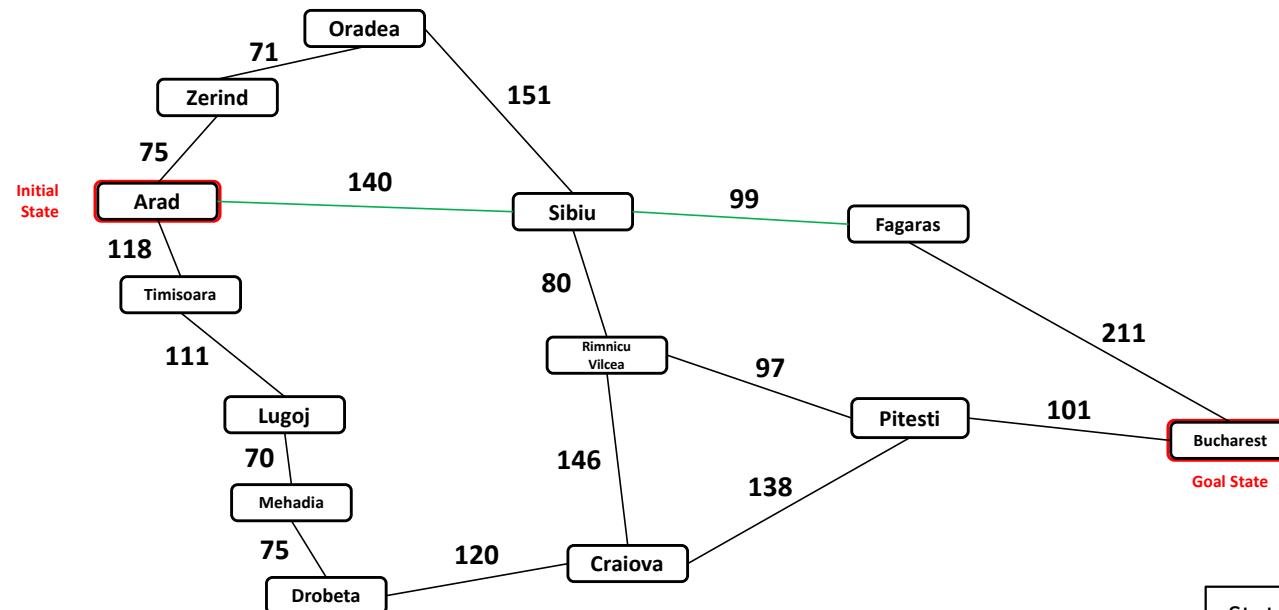
Dracula's Roadtrip: A* Search



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

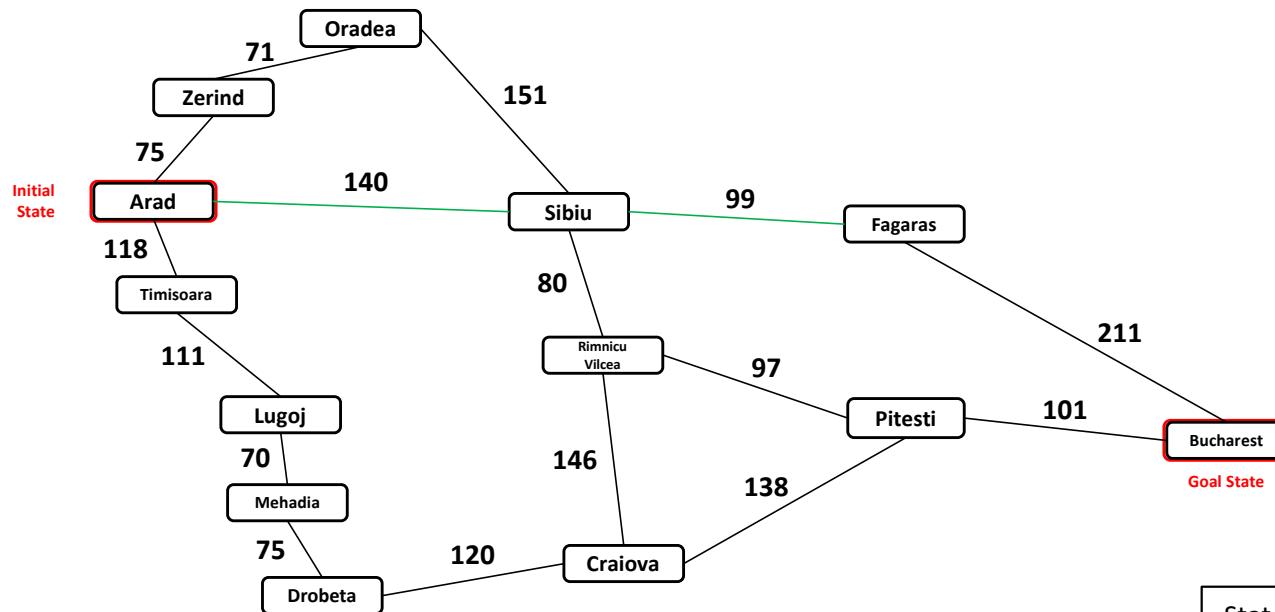
Dracula's Roadtrip: A* Search



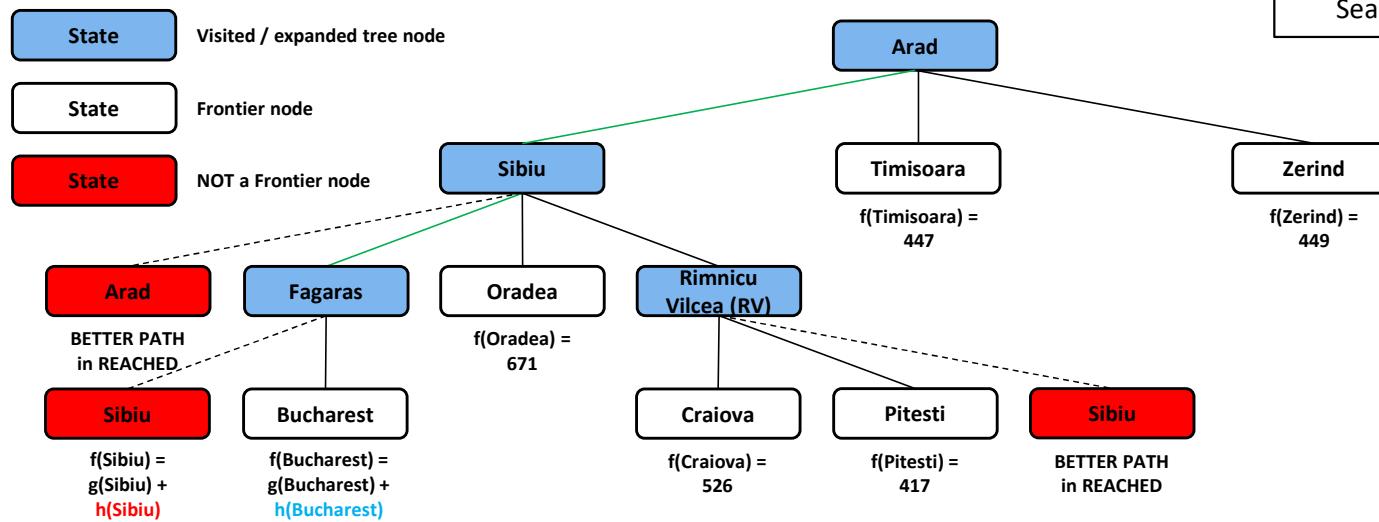
Straight-line distance to Bucharest (h(State)):

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

Dracula's Roadtrip: A* Search



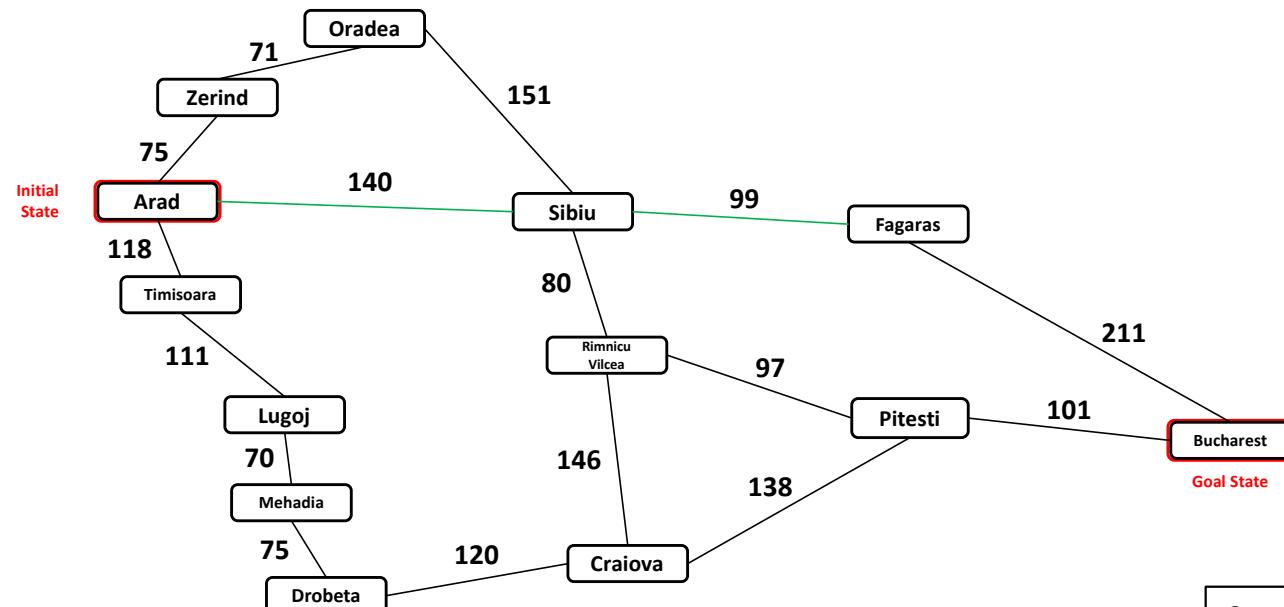
State Space Graph
Search Tree



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

Dracula's Roadtrip: A* Search



Visited / expanded tree node

Frontier node

NOT a Frontier node

Arad

BETTER PATH in REACHED

$f(\text{Sibiu}) = 140 + 99 + 99 + 211 = 591$

Sibiu

Fagaras

$f(\text{Bucharest}) = 140 + 99 + 211 + 0 = 450$

Oradea

$f(\text{Oradea}) = 671$

$f(\text{Craiova}) = 526$

Timisoara

$f(\text{Timisoara}) = 447$

Rimnicu Vilcea (RV)

$f(\text{Pitesti}) = 417$

Zerind

$f(\text{Zerind}) = 449$

Sibiu

BETTER PATH in REACHED

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

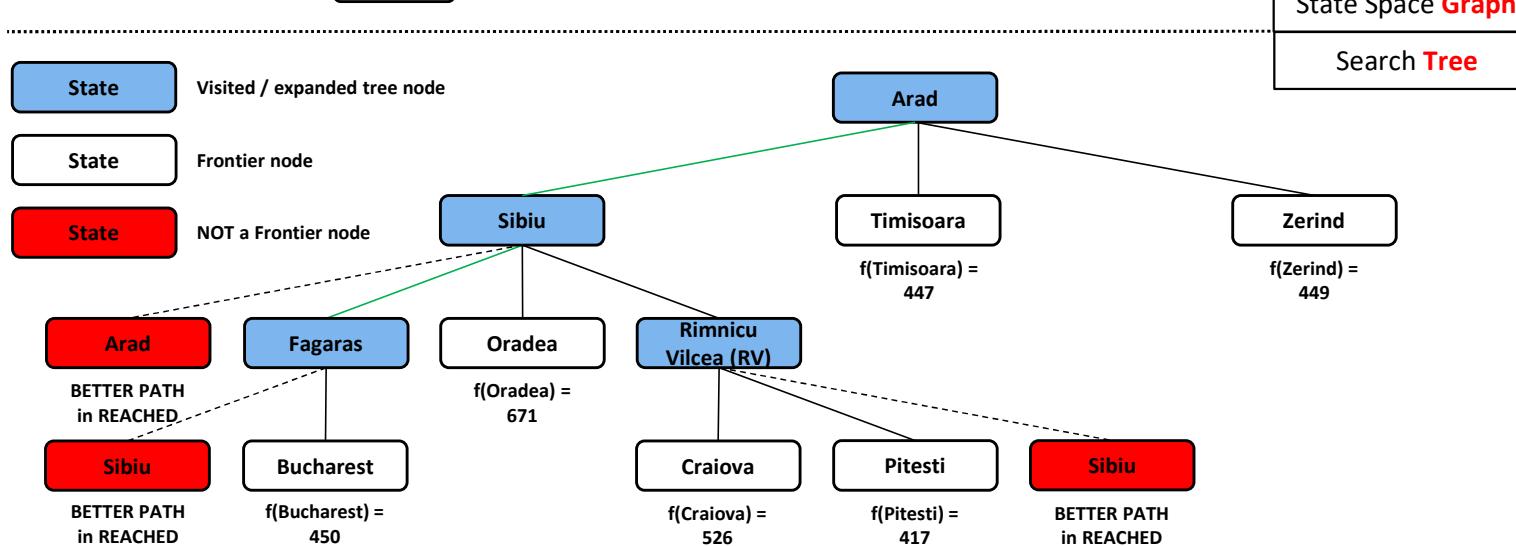
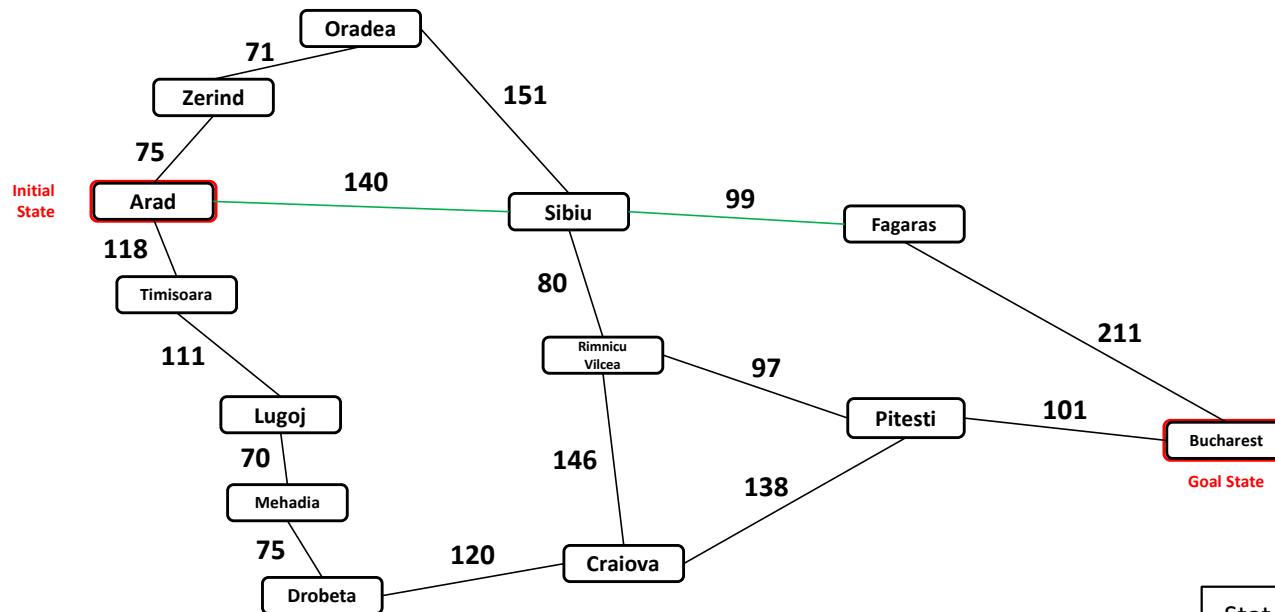
Timisoara 329

Urziceni 80

Vaslui 199

Zerind 374

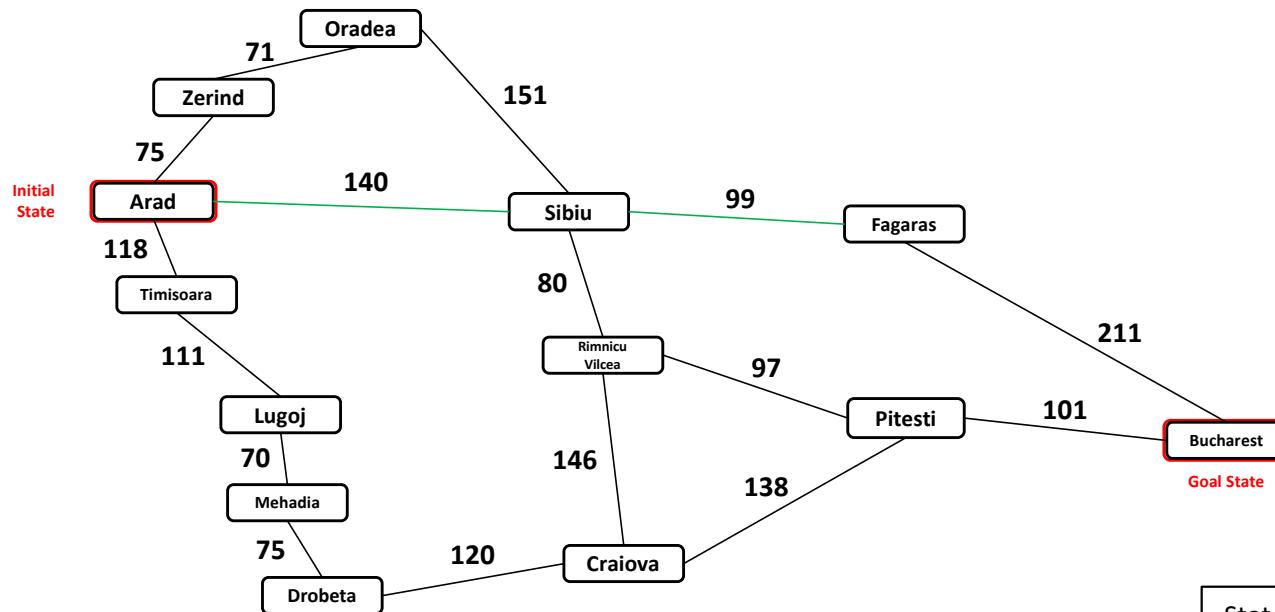
Dracula's Roadtrip: A* Search



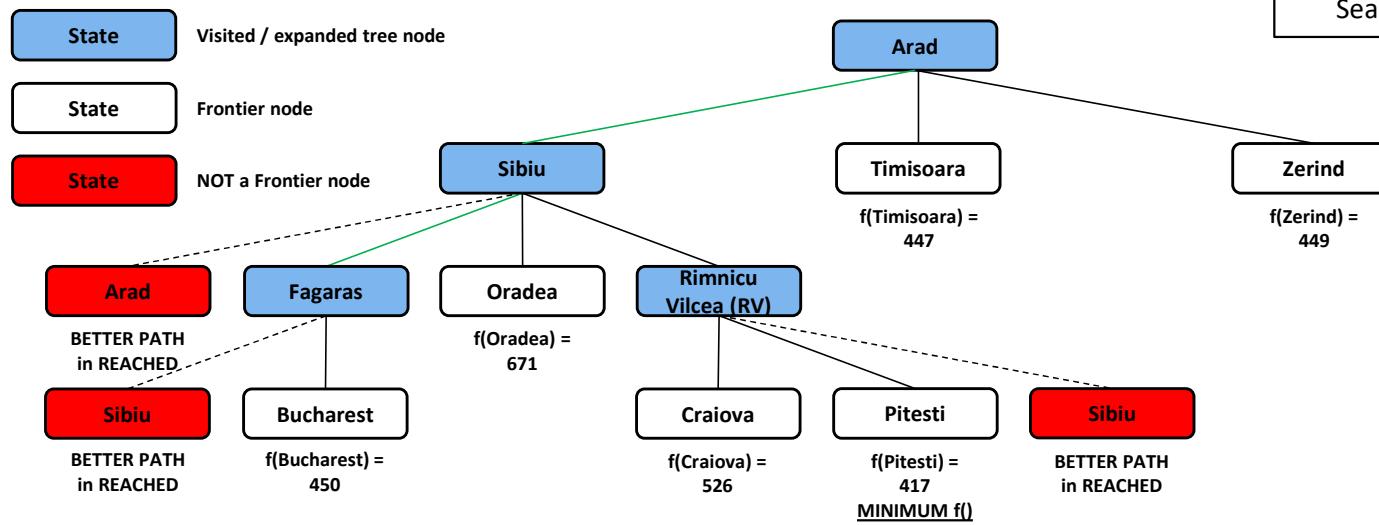
Straight-line distance to Bucharest ($h(\text{State})$):

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

Dracula's Roadtrip: A* Search



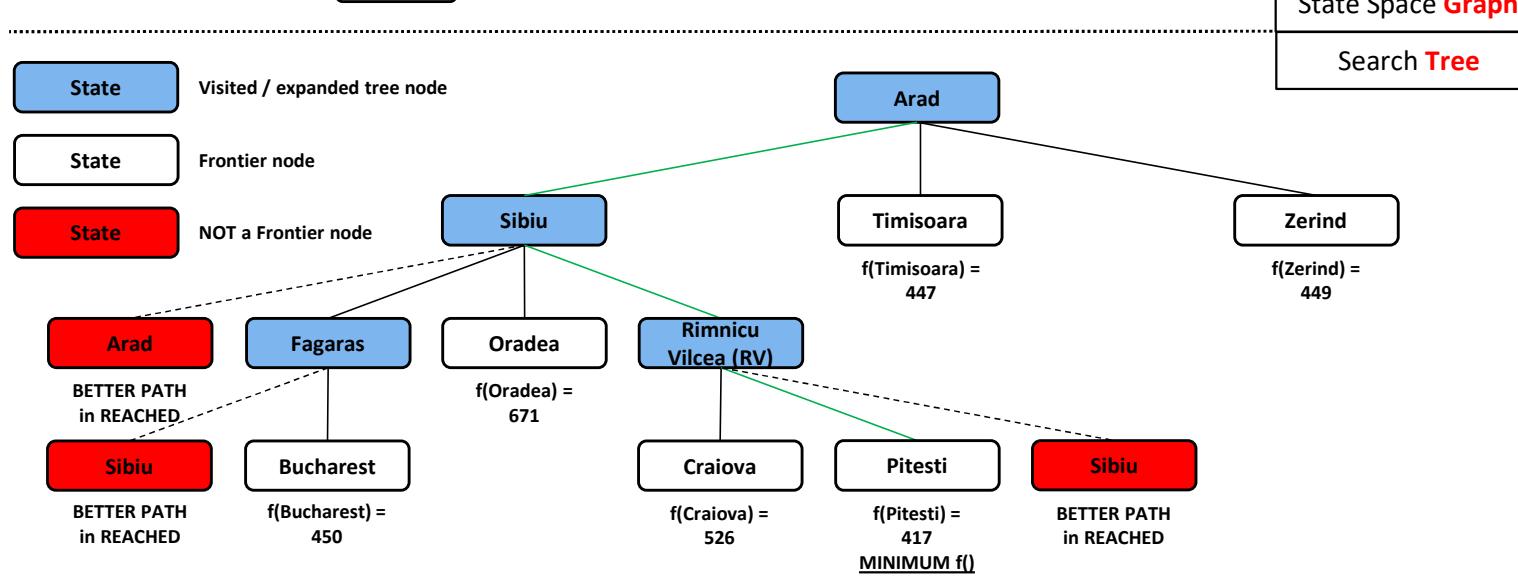
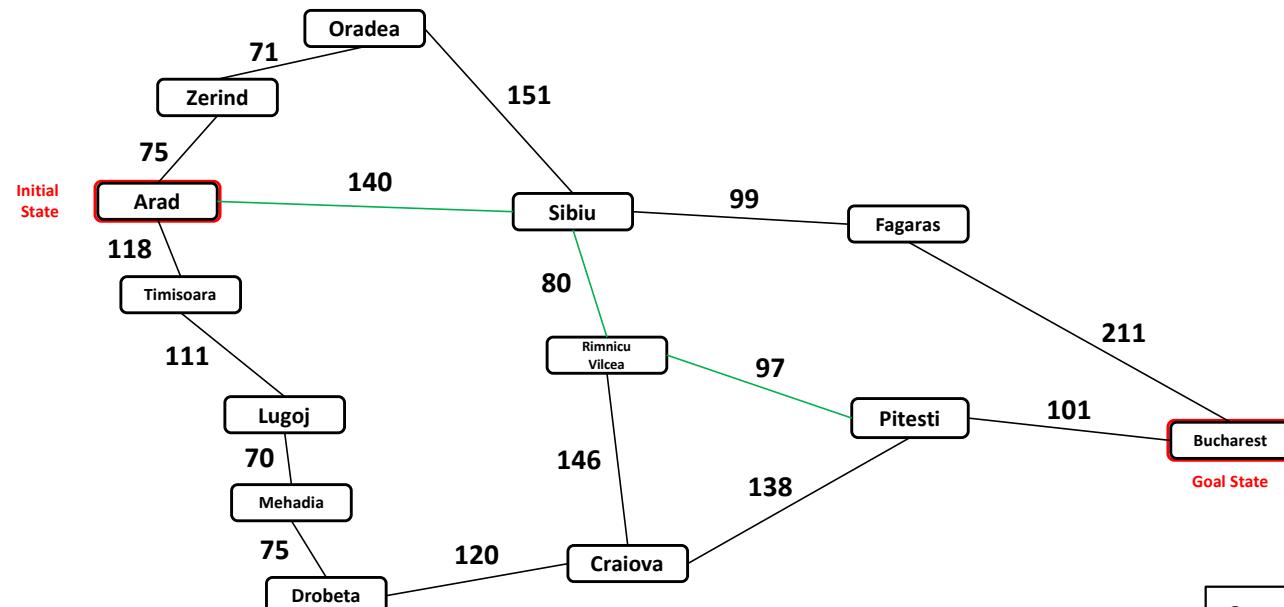
State Space Graph
Search Tree



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

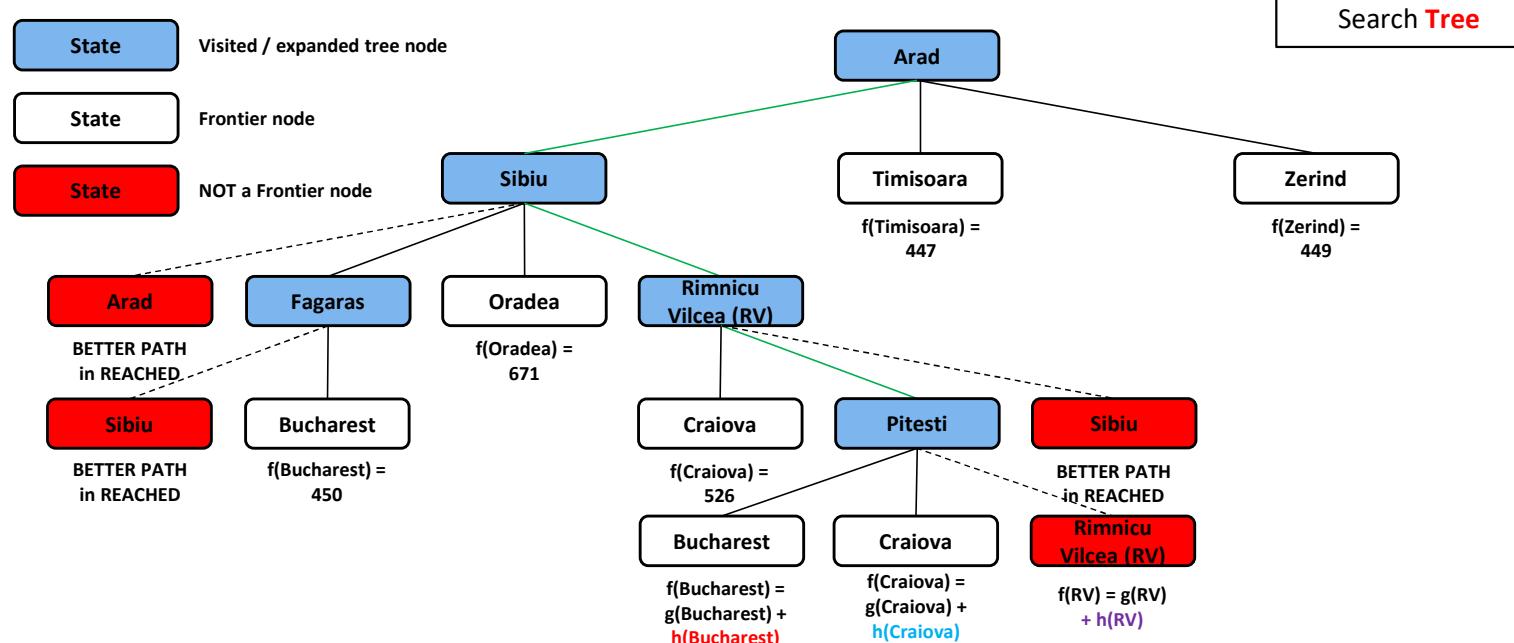
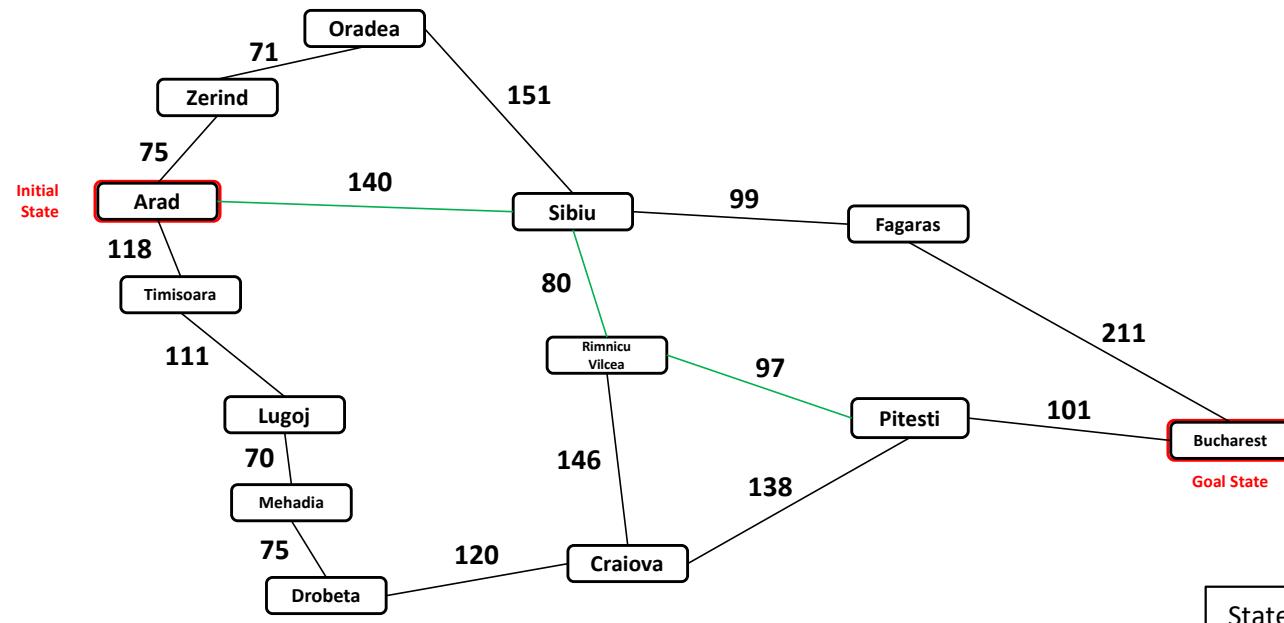
Dracula's Roadtrip: A* Search



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

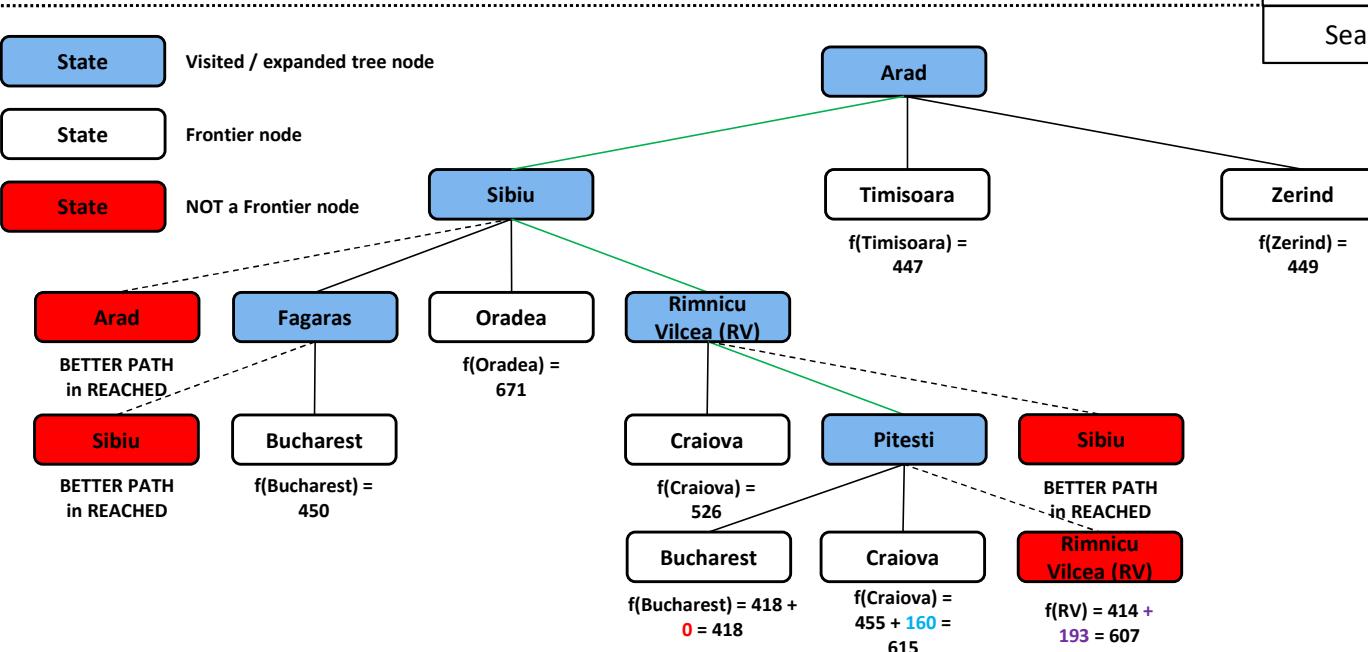
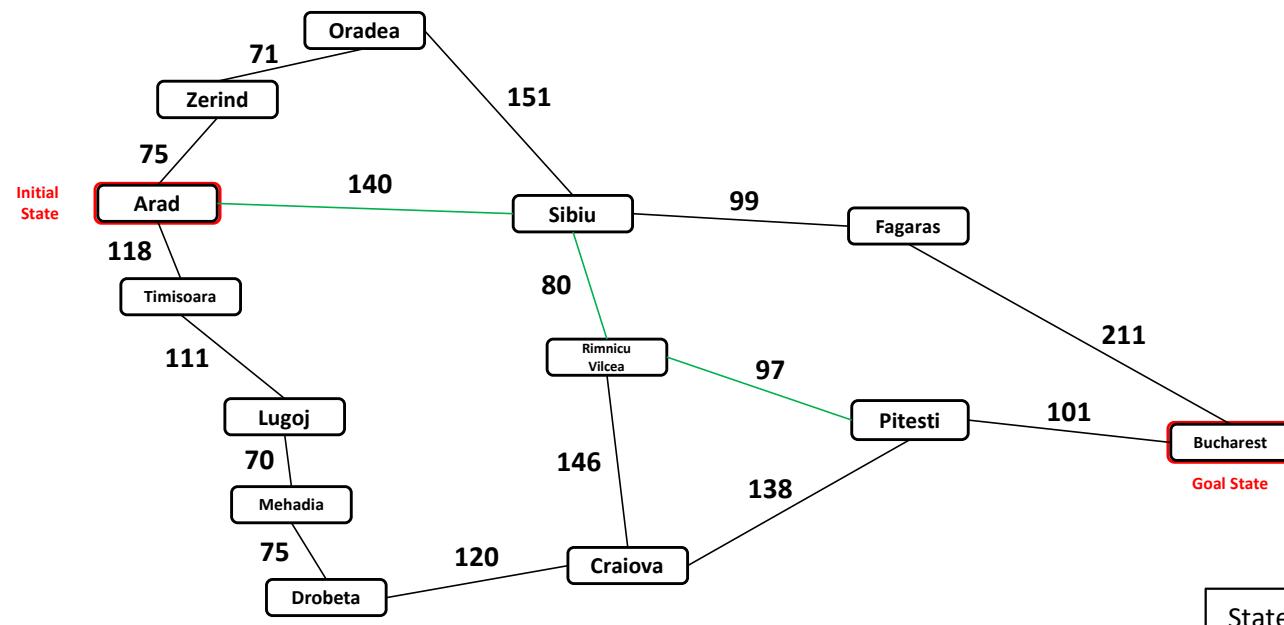
Dracula's Roadtrip: A* Search



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

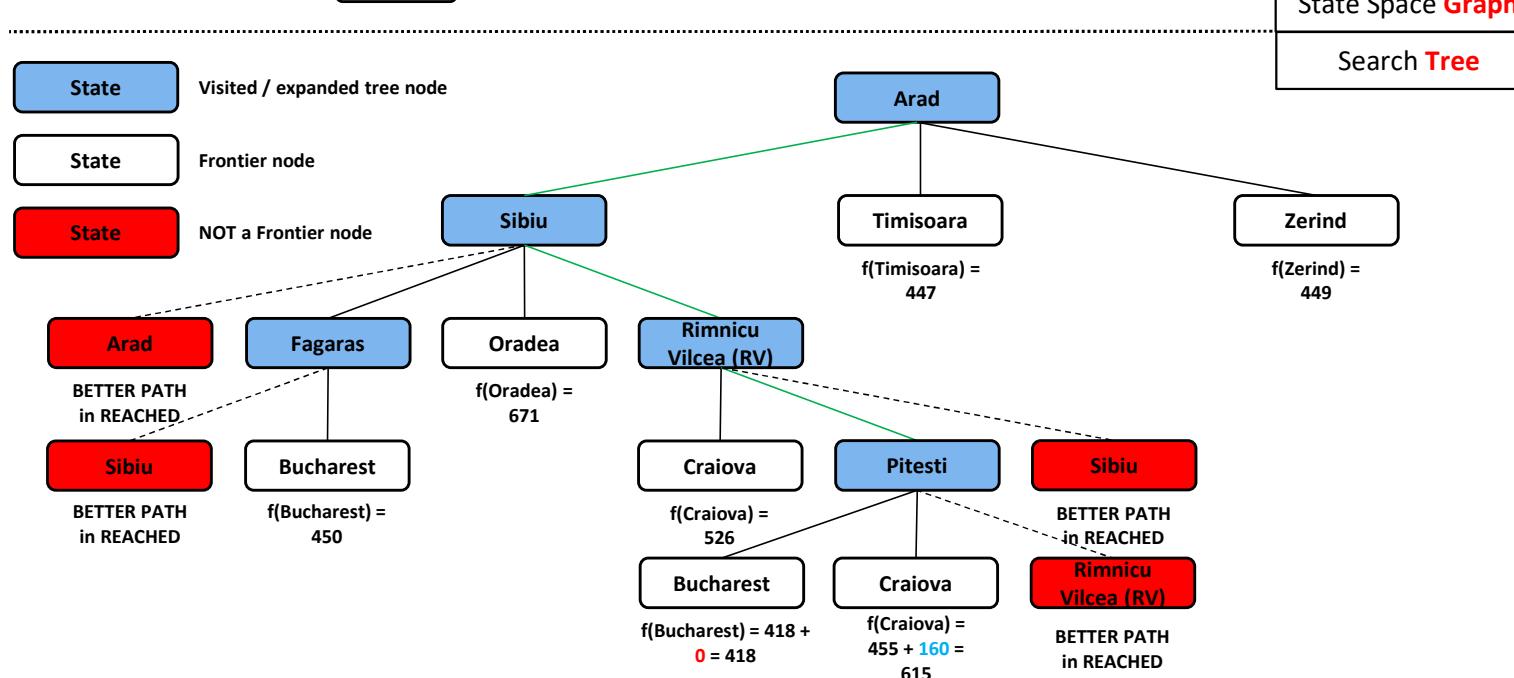
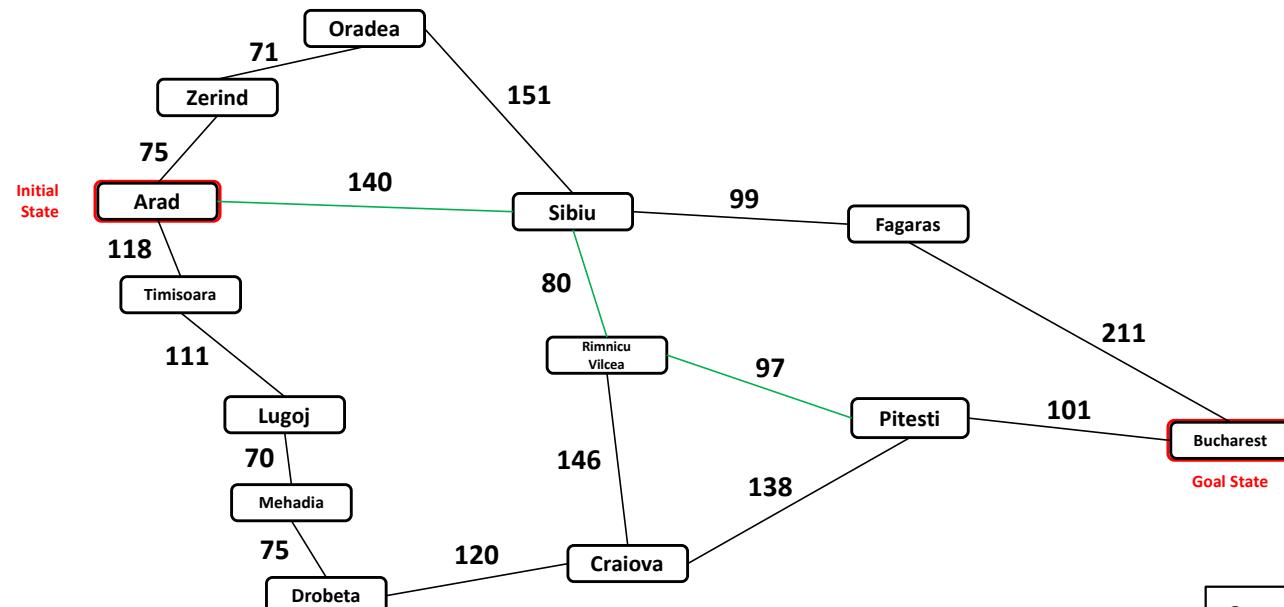
Dracula's Roadtrip: A* Search



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

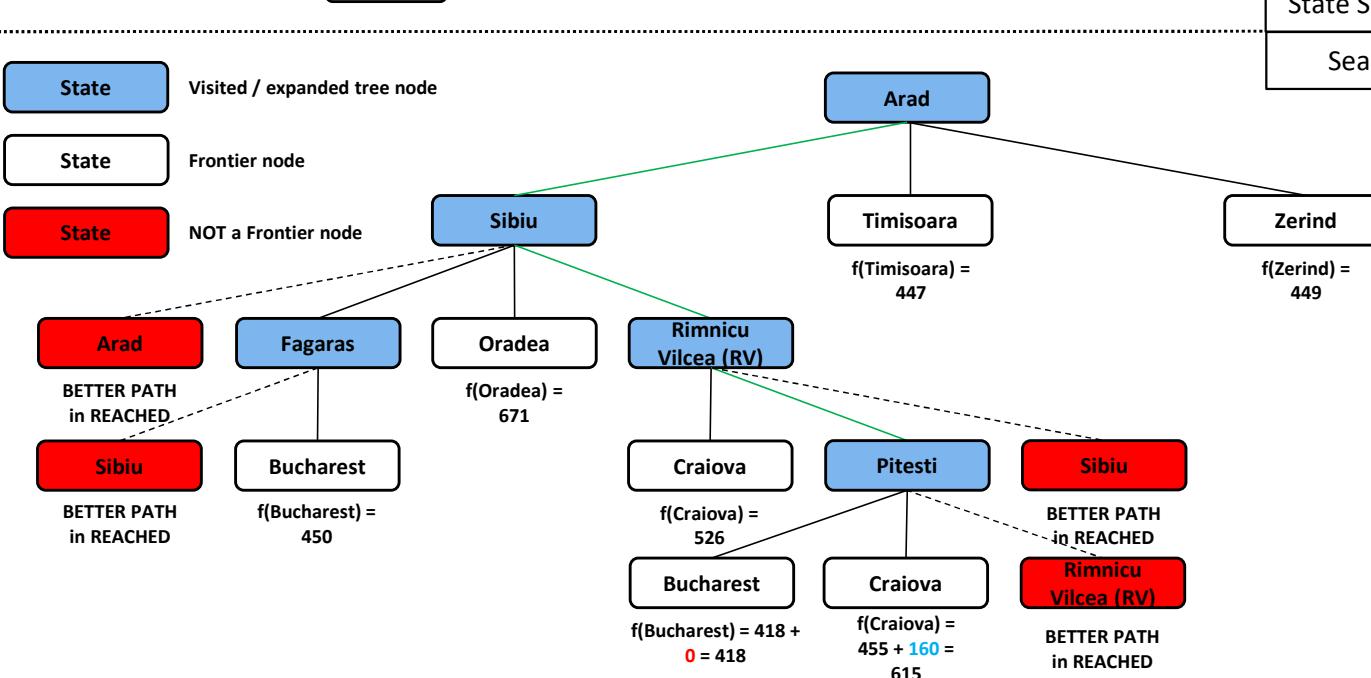
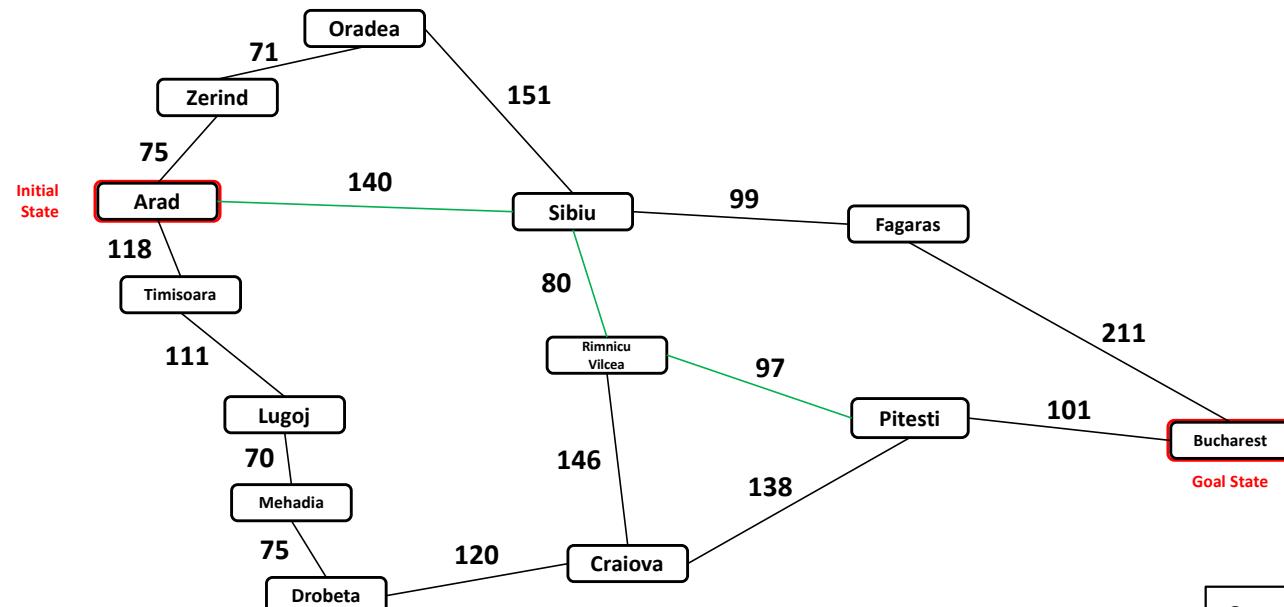
Dracula's Roadtrip: A* Search



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

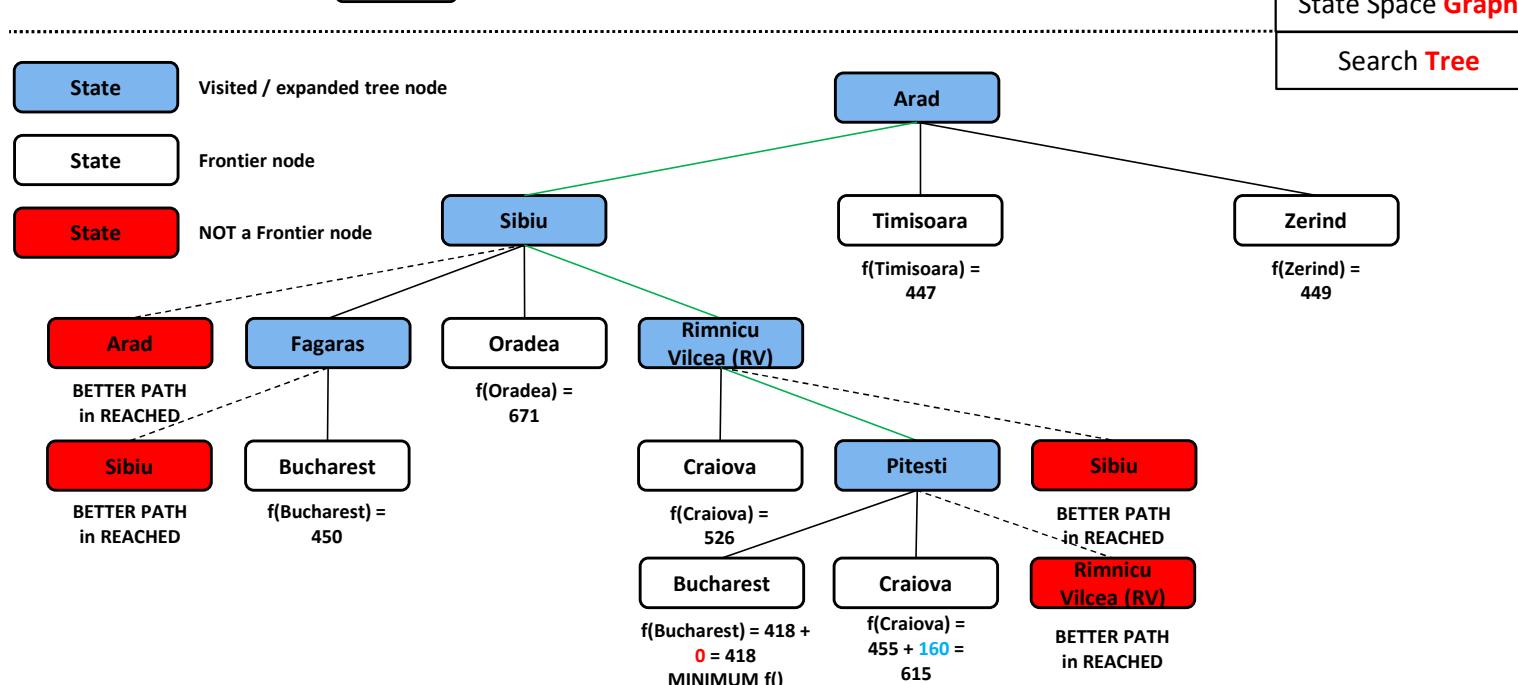
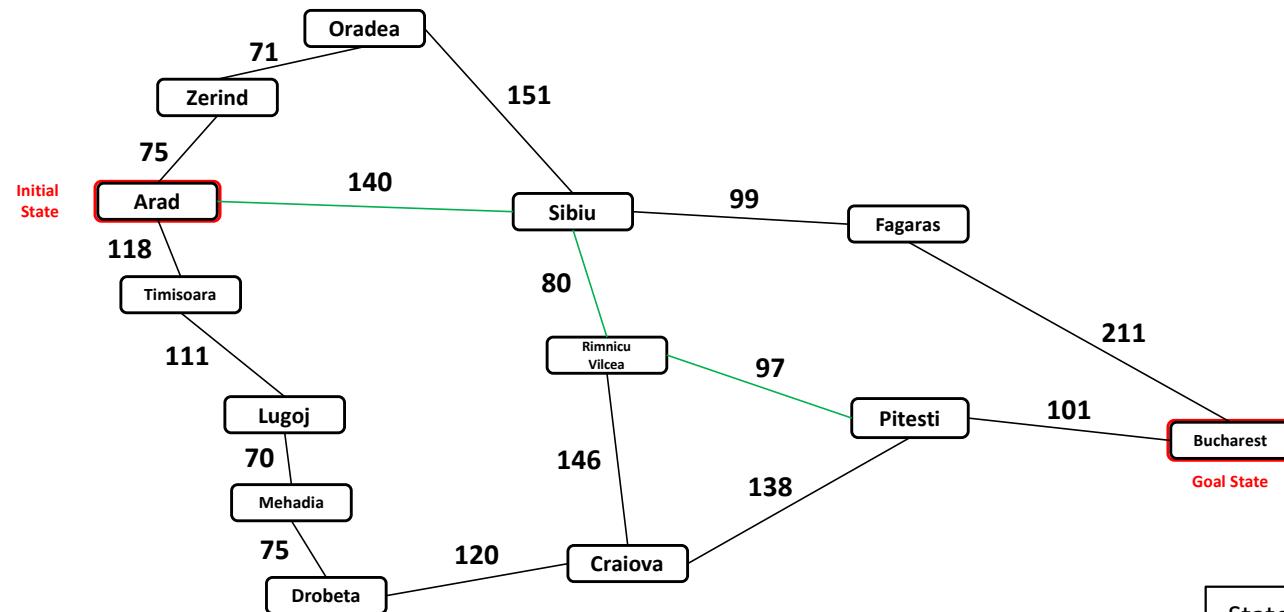
Dracula's Roadtrip: A* Search



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

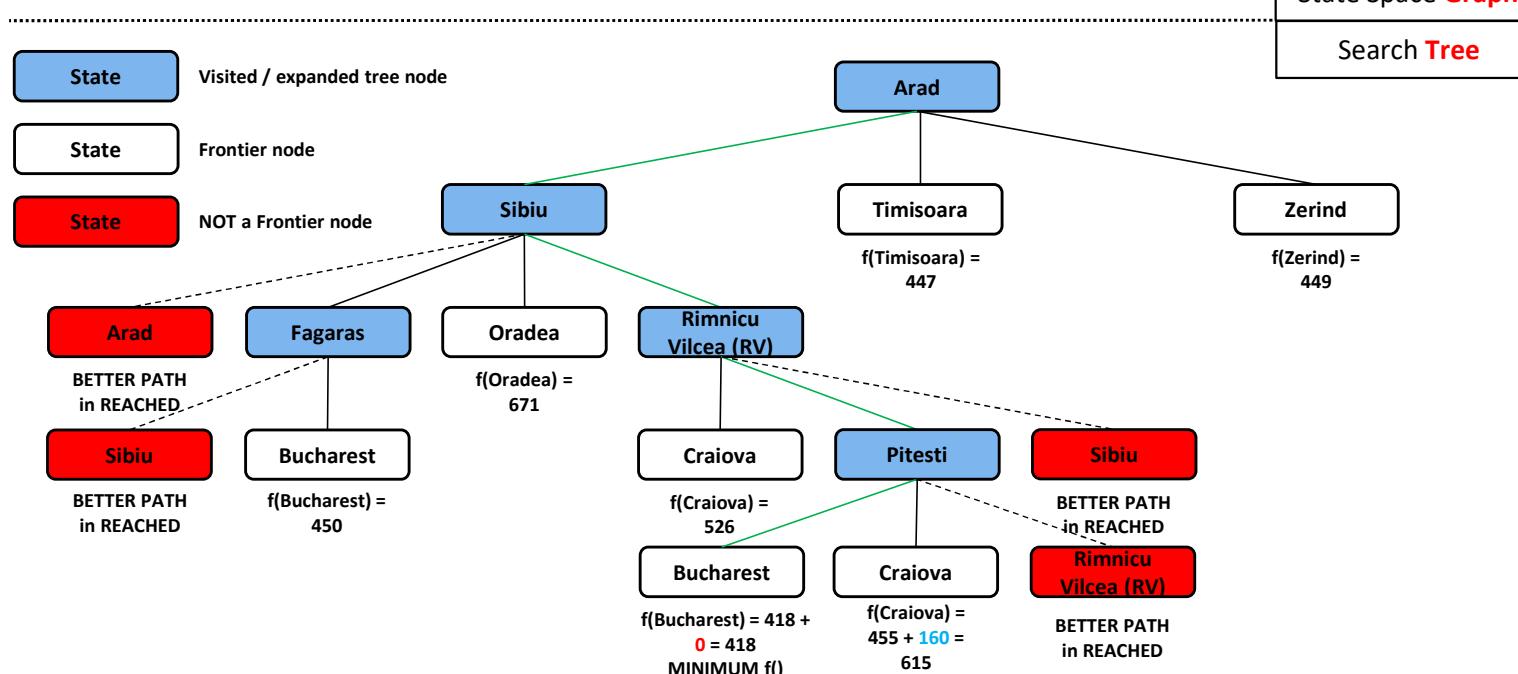
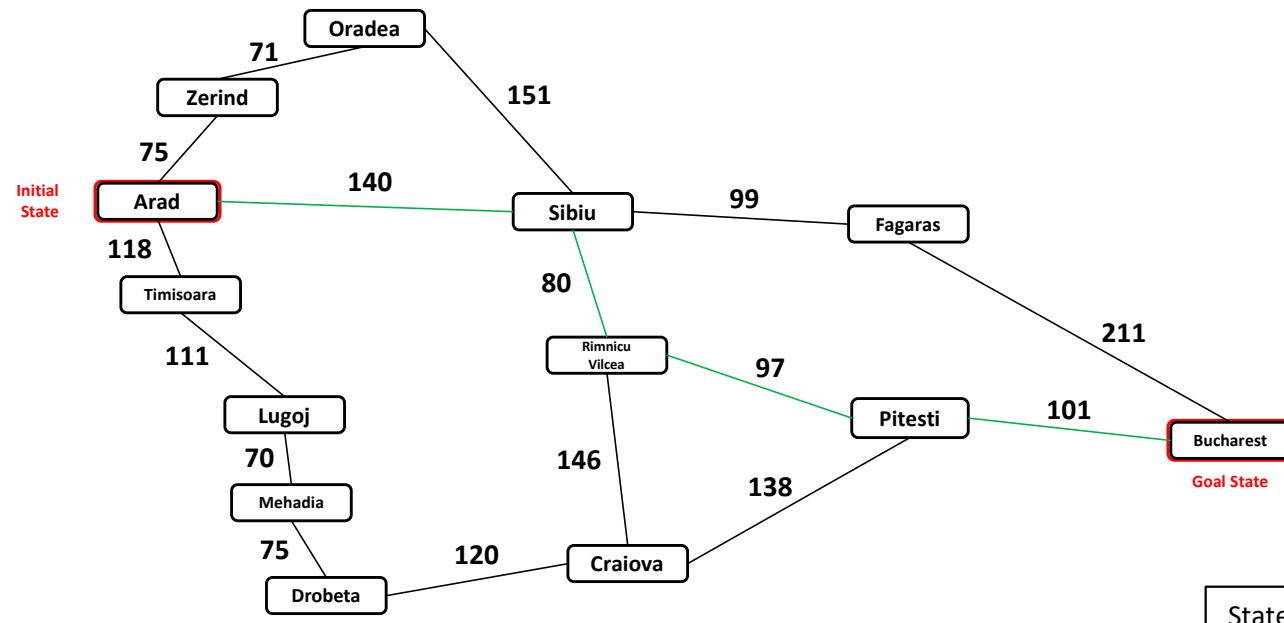
Dracula's Roadtrip: A* Search



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

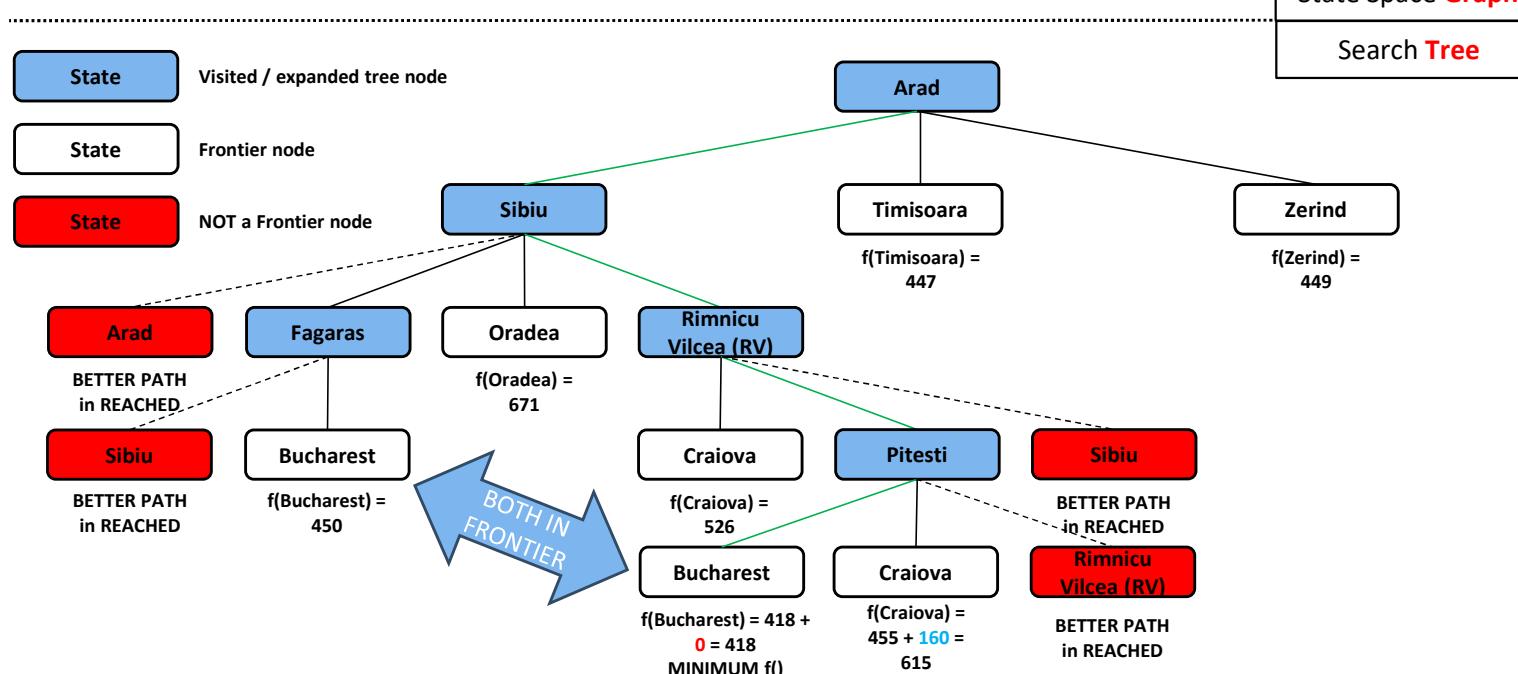
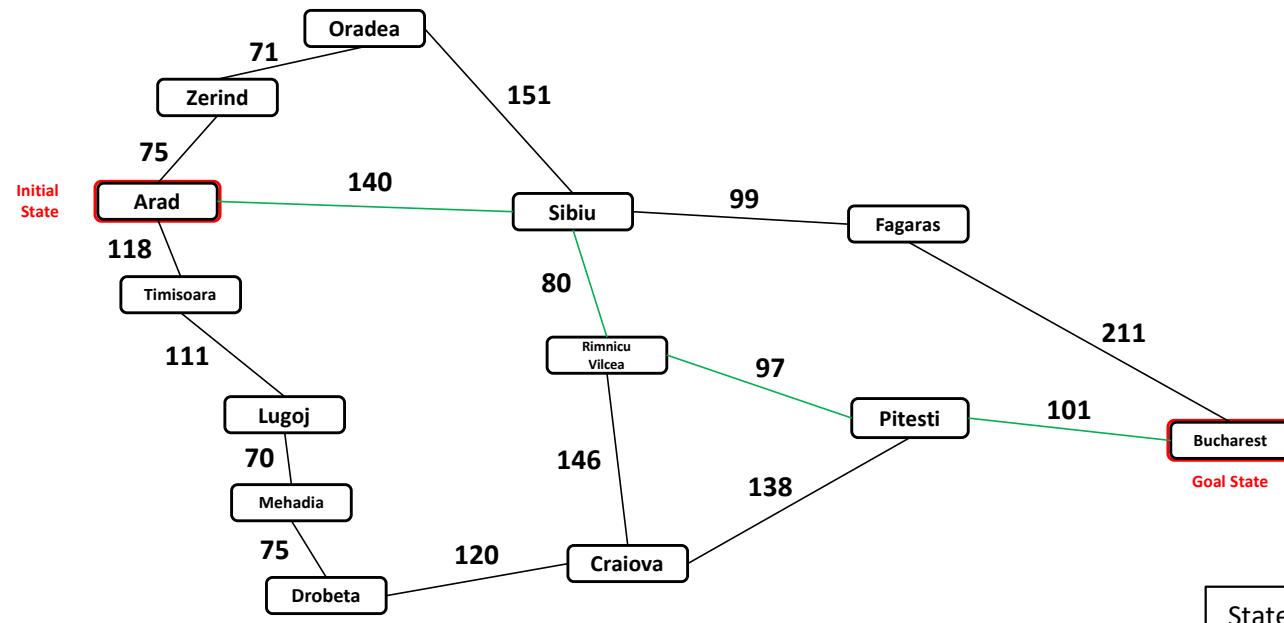
Dracula's Roadtrip: A* Search



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

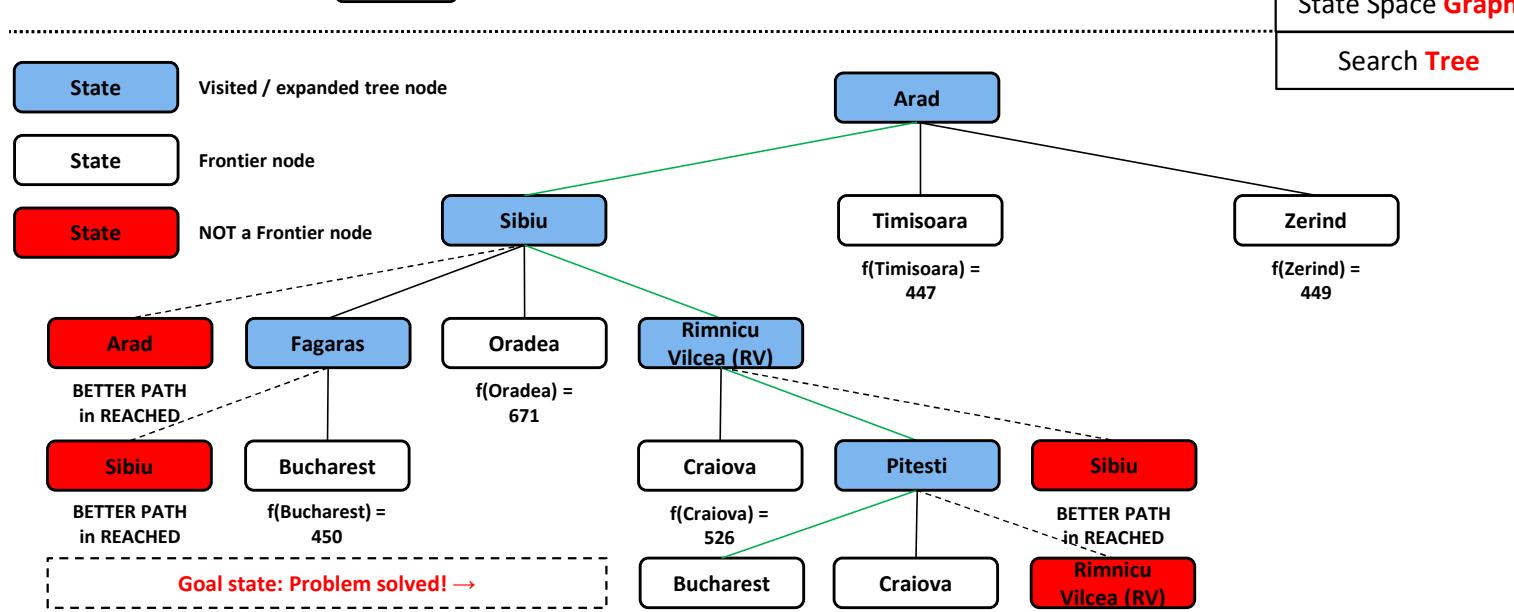
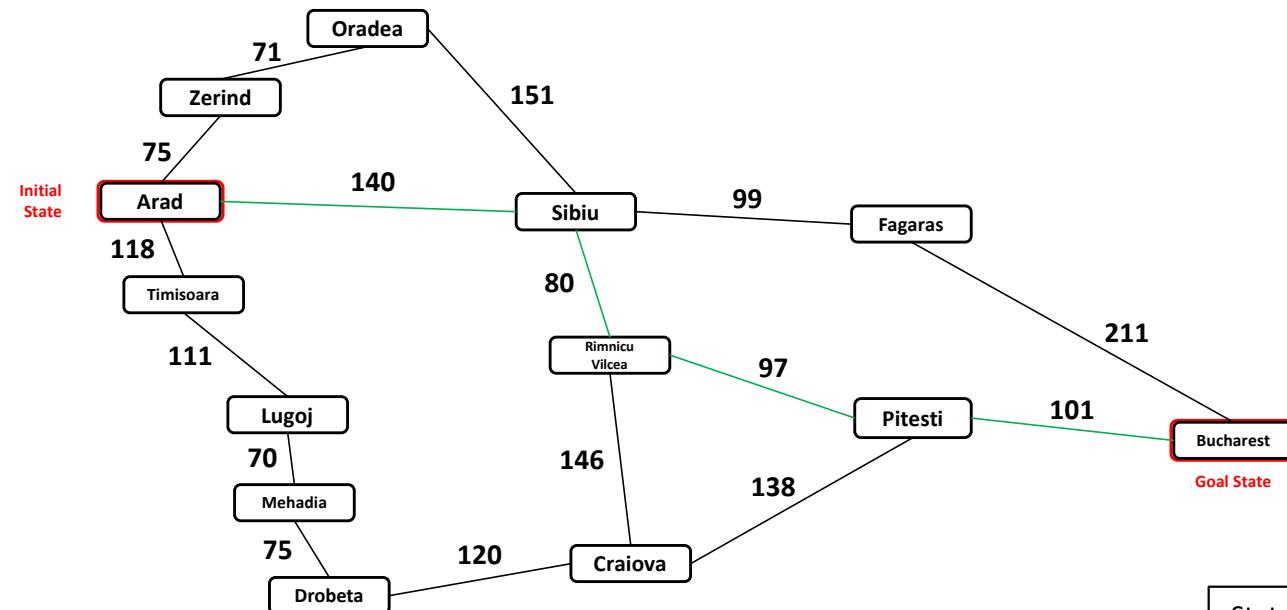
Dracula's Roadtrip: A* Search



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

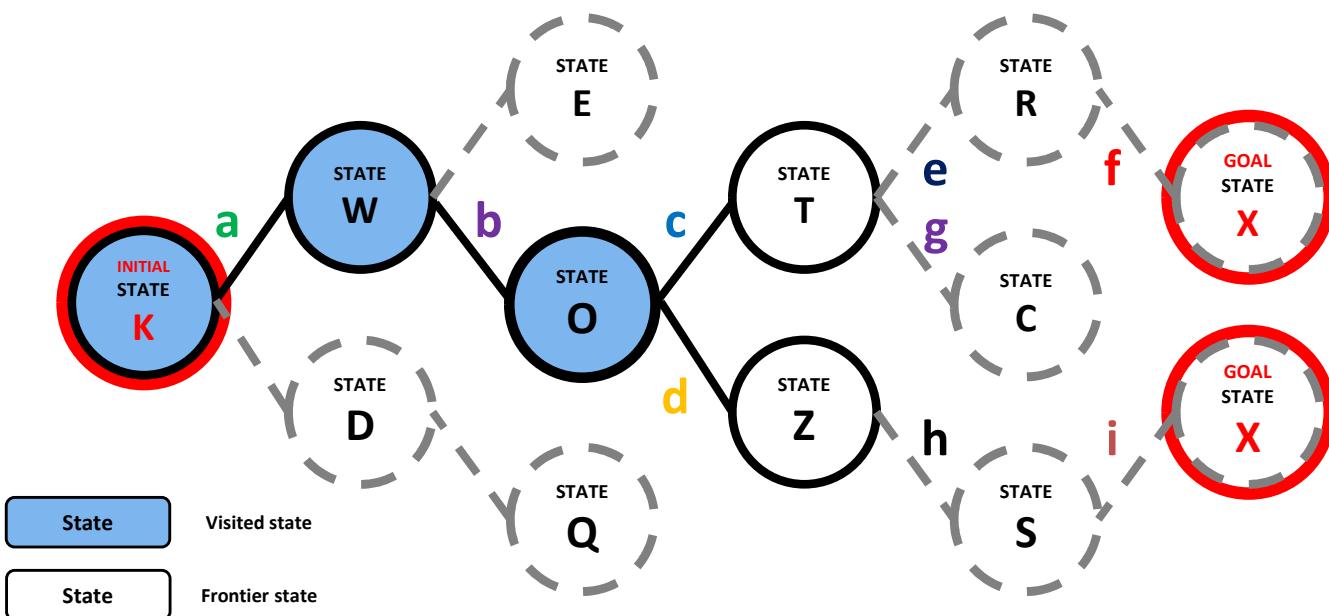
Dracula's Roadtrip: A* Search



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

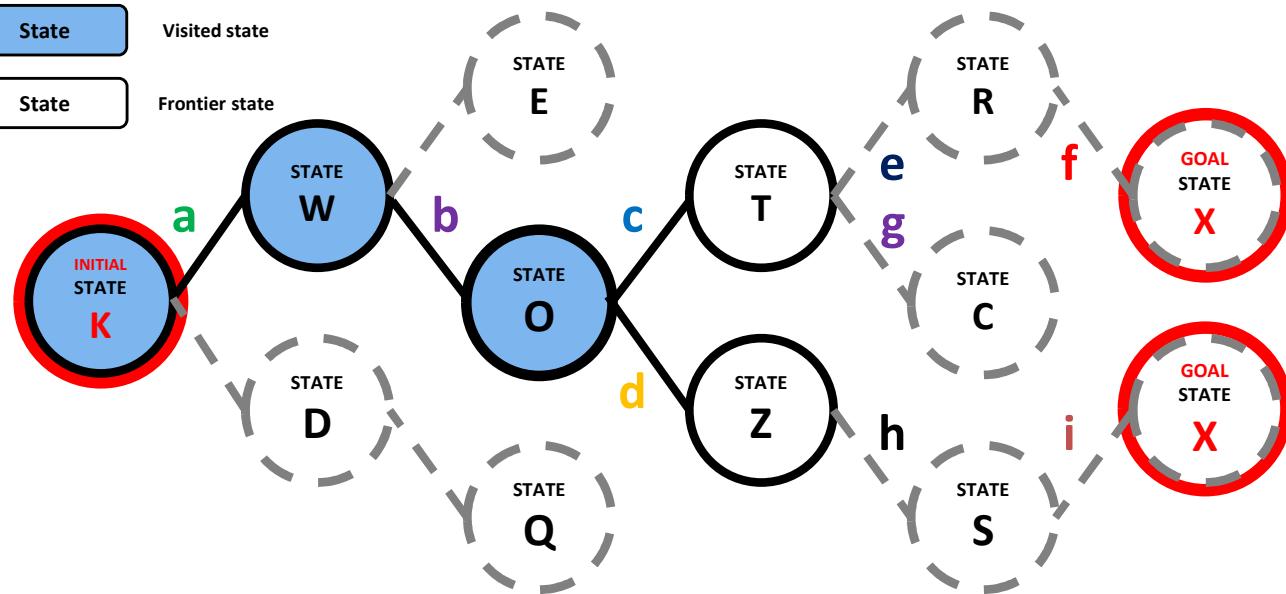
Greedy Best First Search vs. A* Search



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?
 $f(T) = g(T) + h(T)$
 $f(T) = a + b + c + h(T)$

$$f(Z) = g(Z) + h(Z)$$

$$f(T) = a + b + d + h(Z)$$

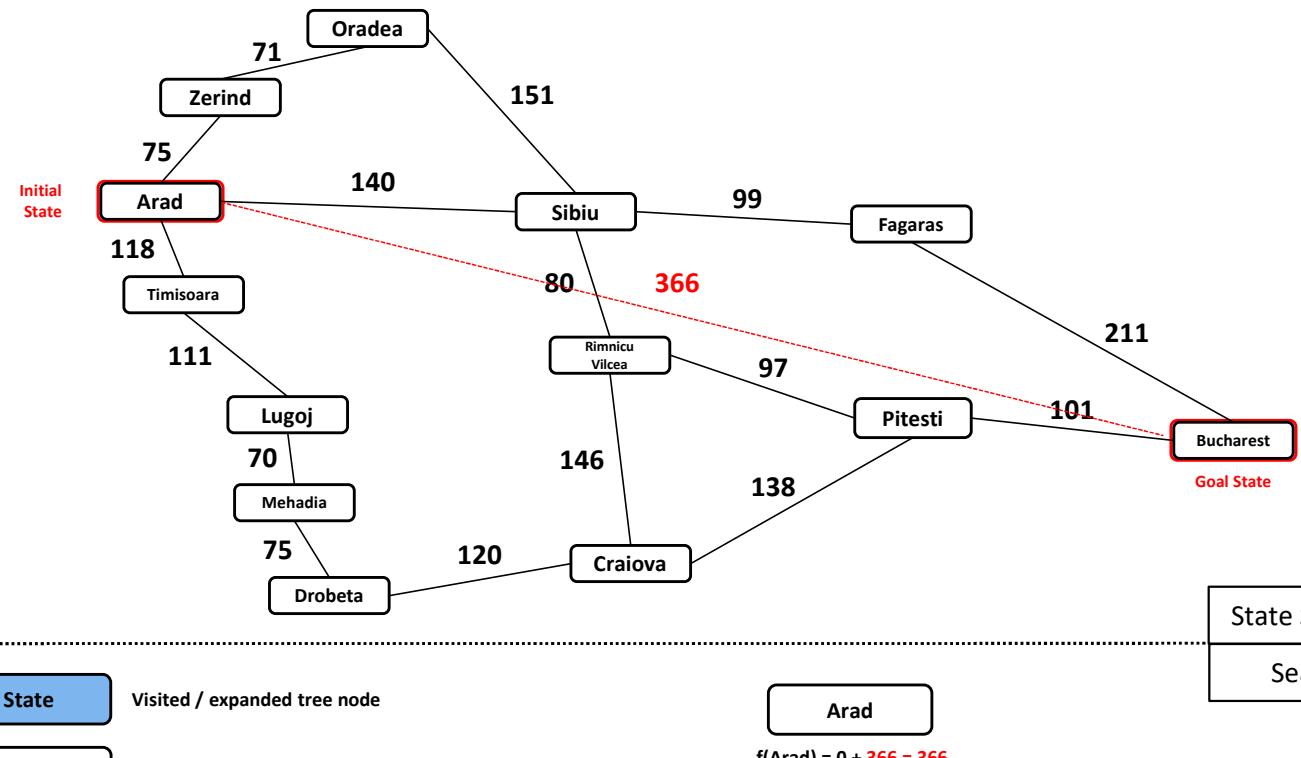
Pick state with min $f()$

Greedy Best First Search

Algorithm

[BONUS Material]

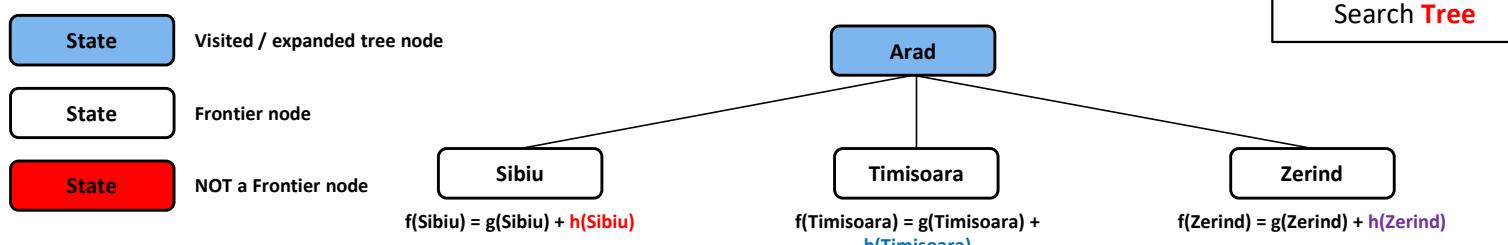
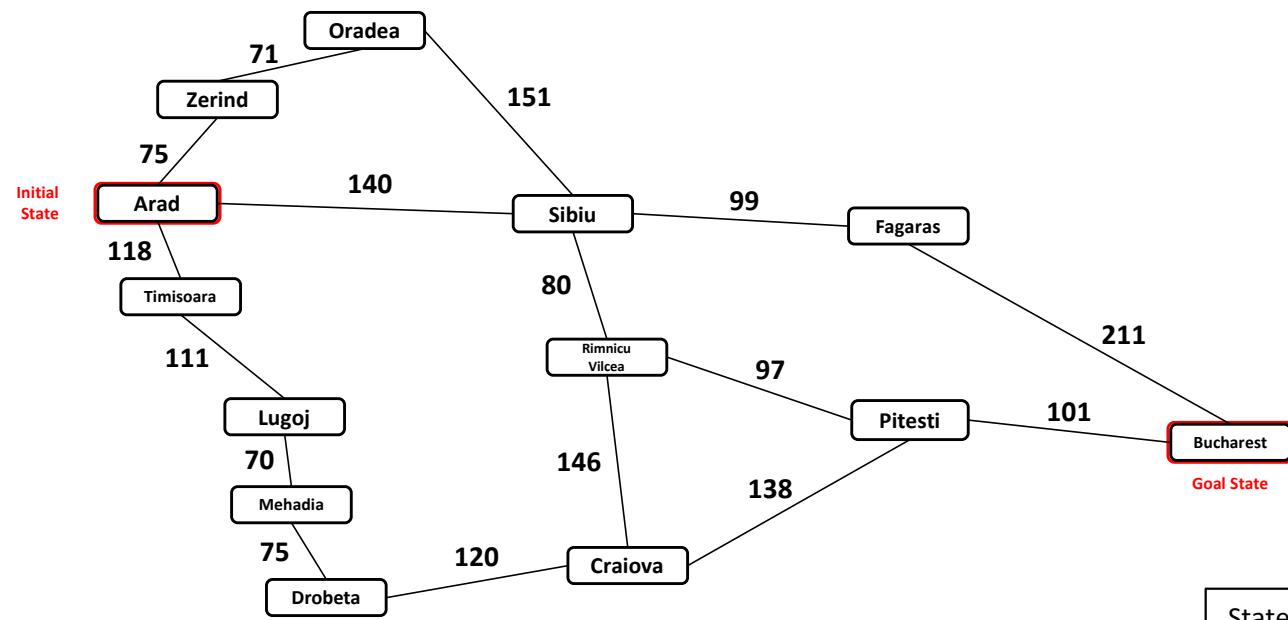
Dracula's Roadtrip: A* Search



Straight-line distance to Bucharest ($h(\text{State})$):

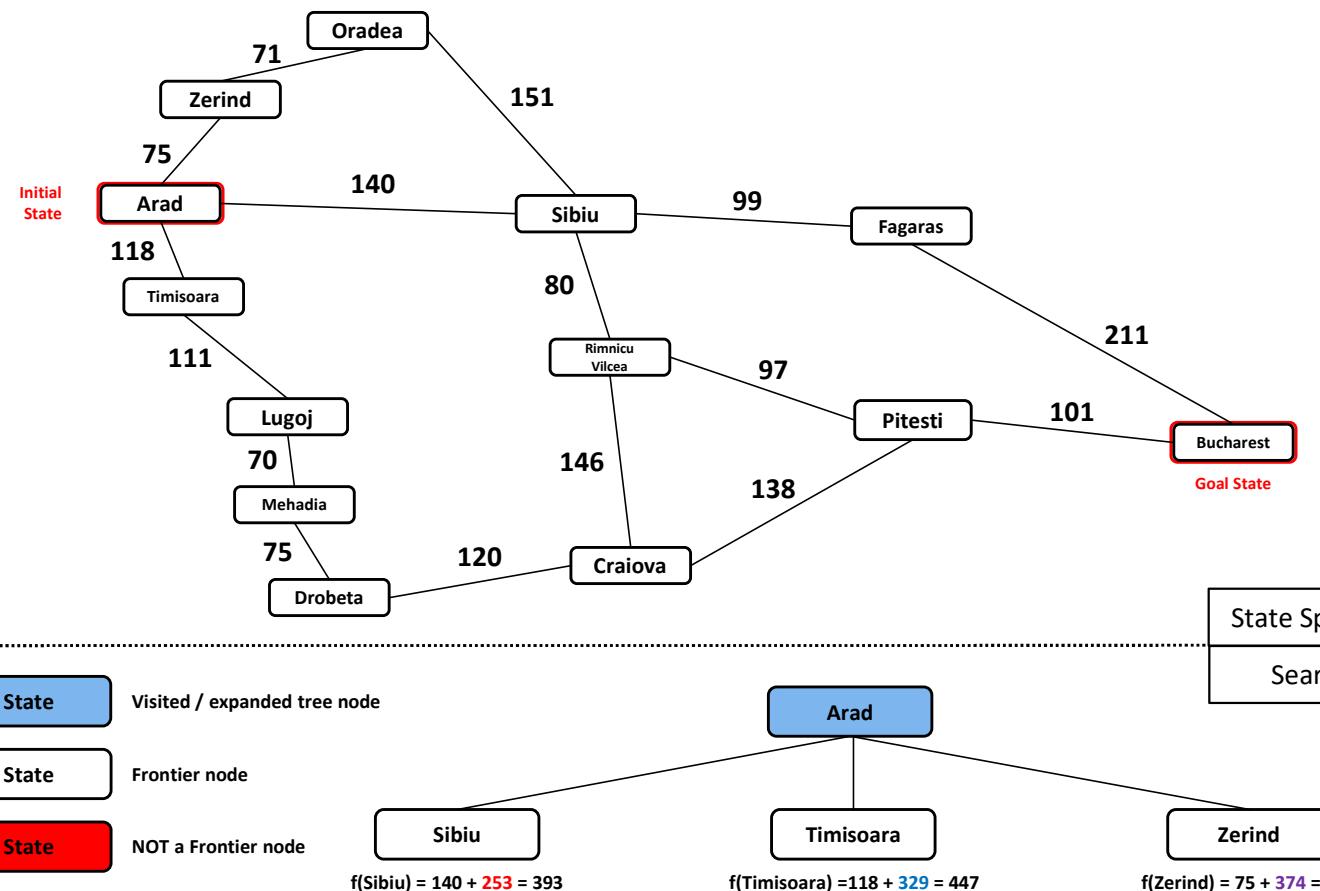
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



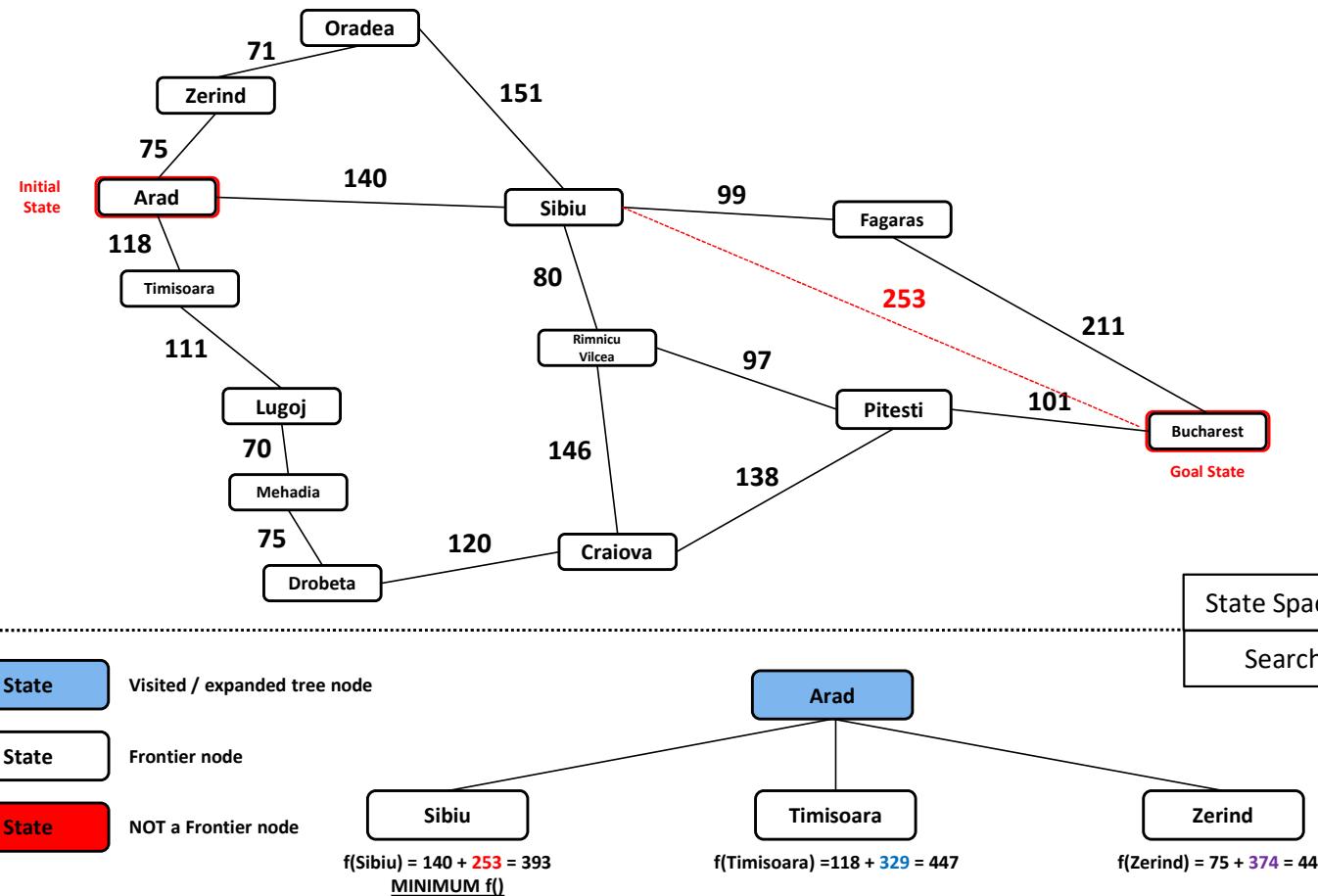
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



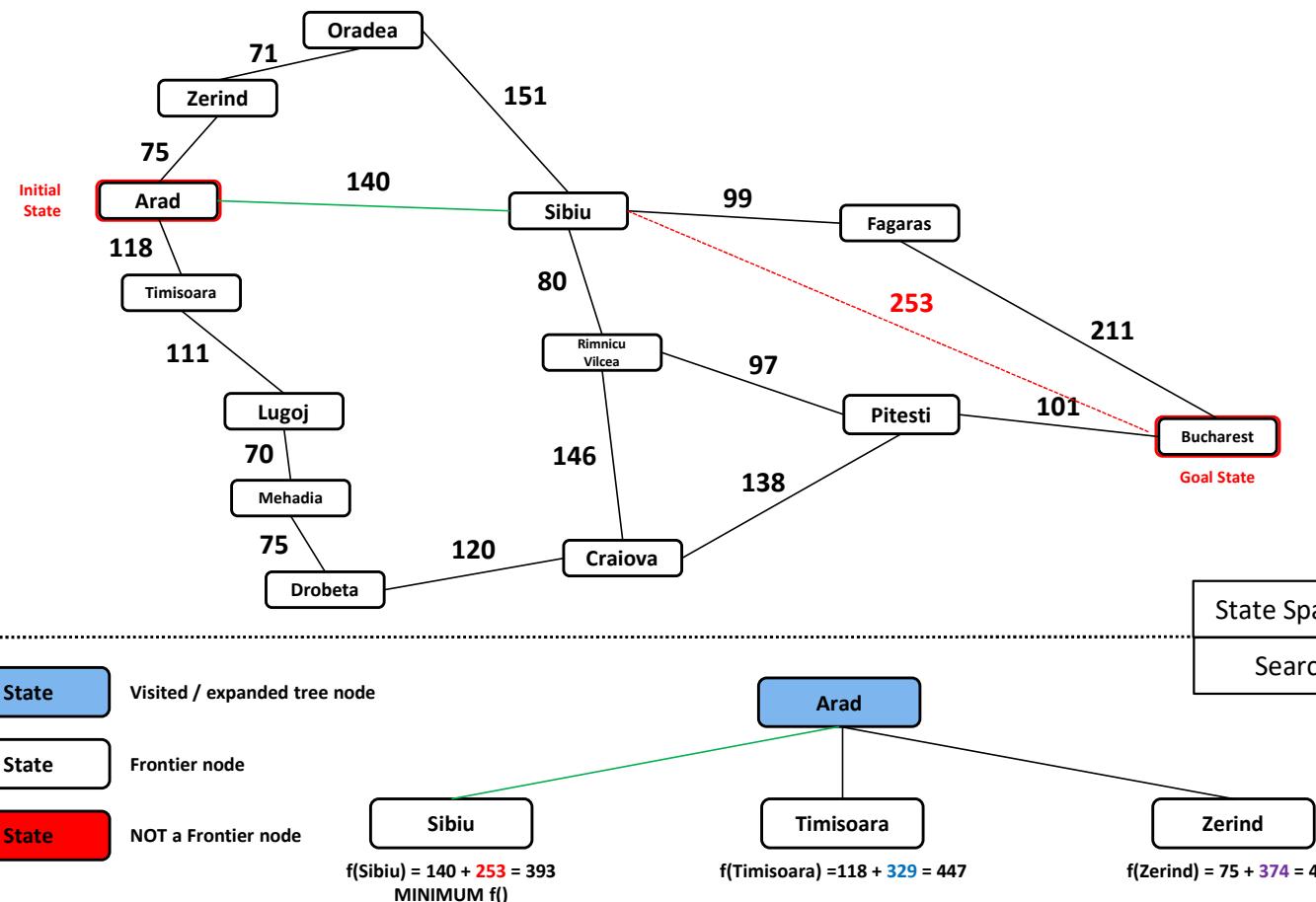
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



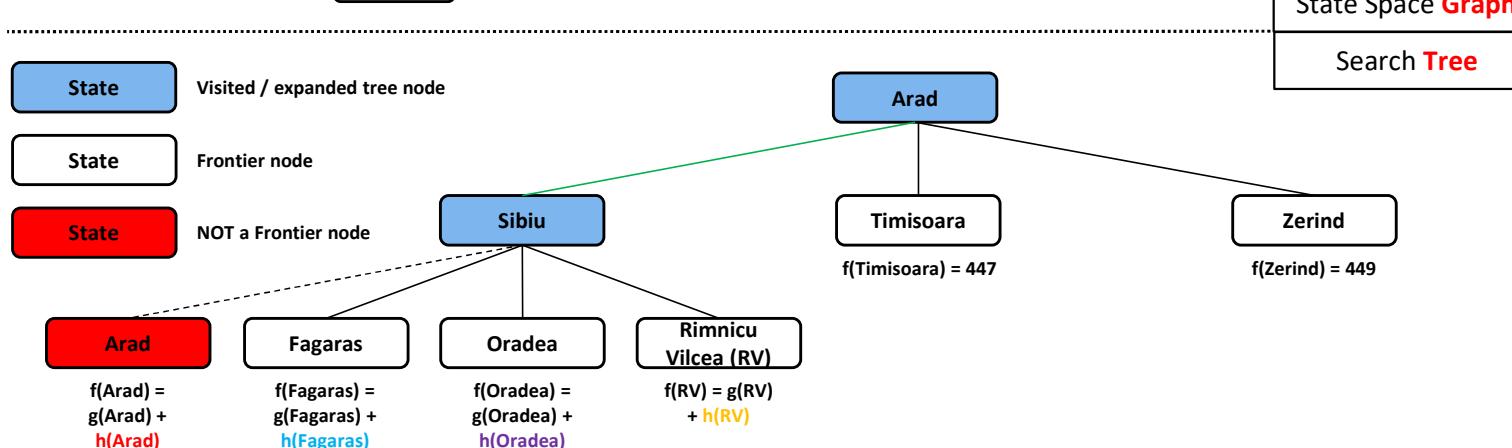
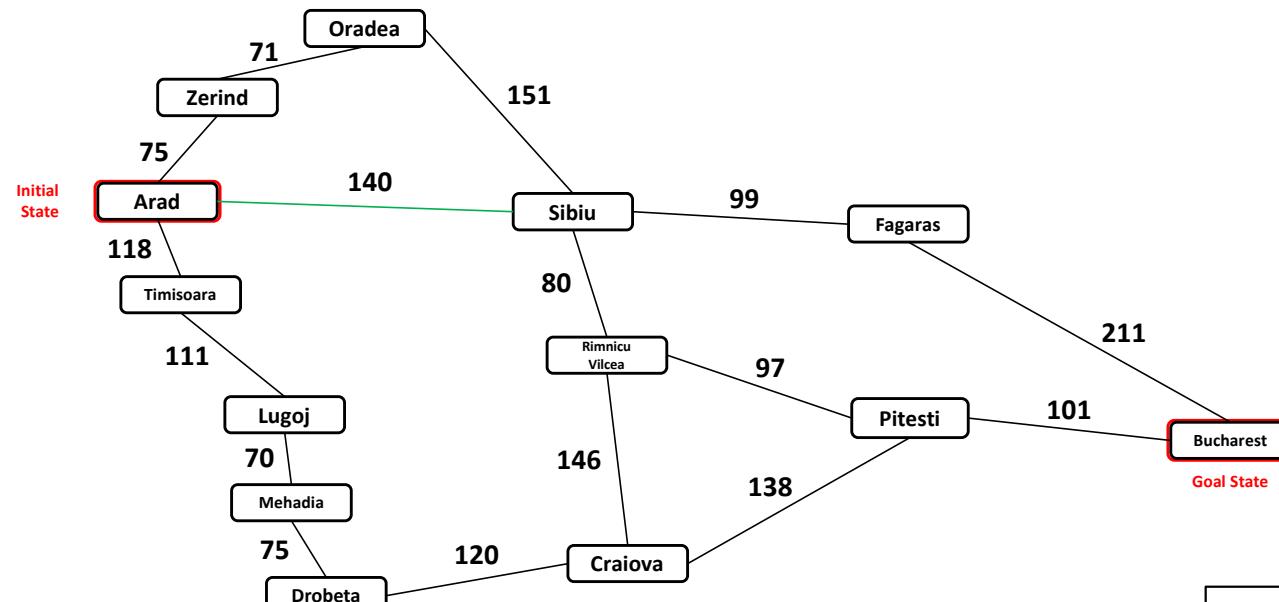
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



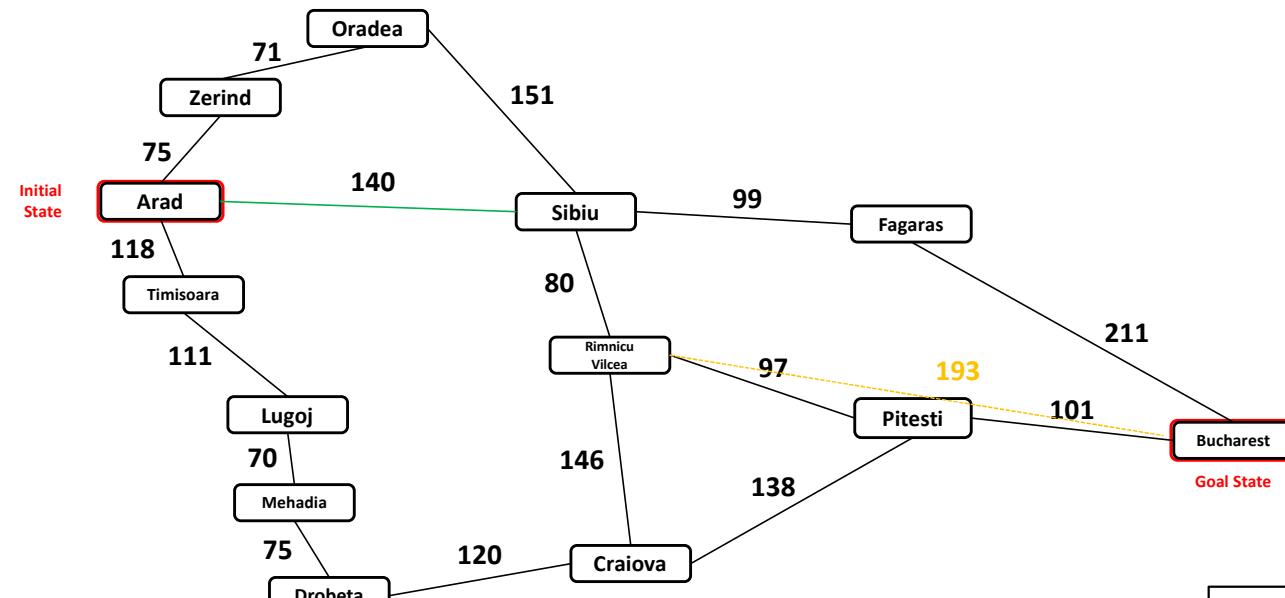
Straight-line distance to Bucharest ($h(\text{State})$):	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	
Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search

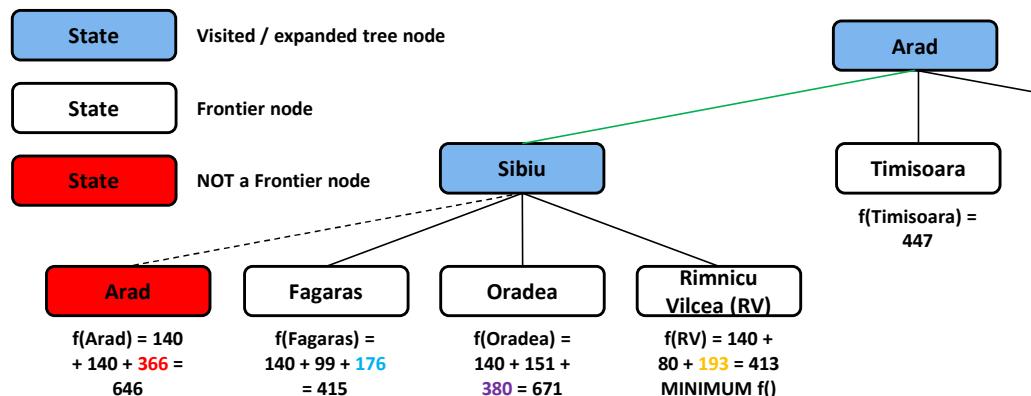


<u>Straight-line distance to Bucharest ($h(\text{State})$):</u>	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadi	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dracula's Roadtrip: A* Search



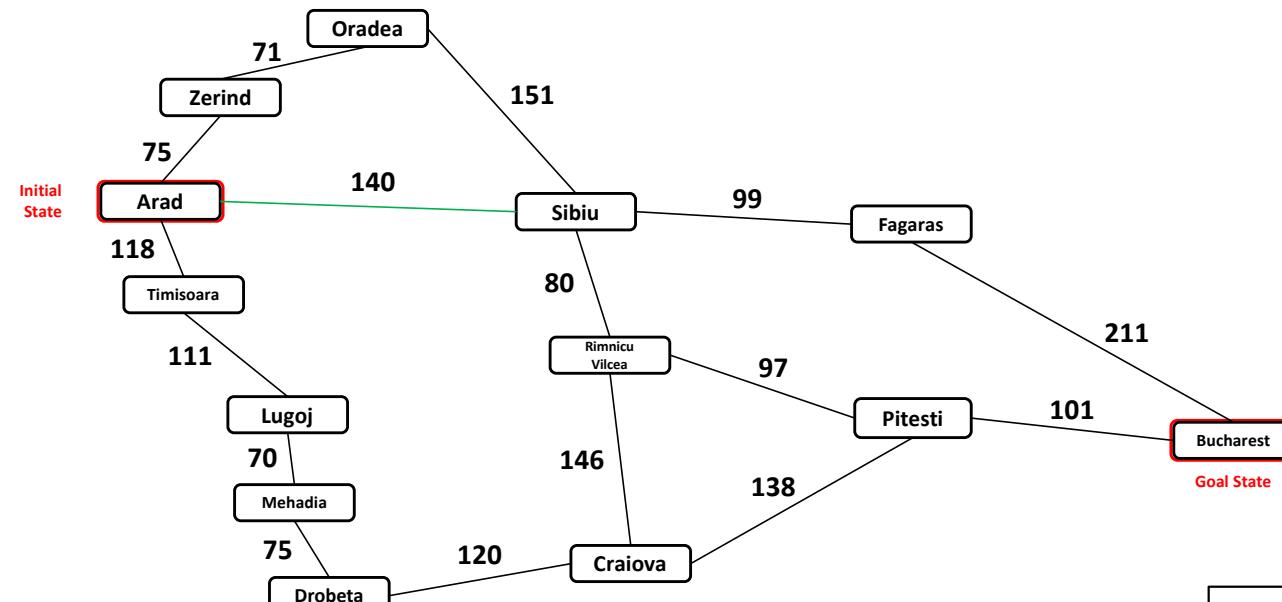
State Space Graph
Search Tree



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

Dracula's Roadtrip: A* Search



State Space Graph
Search Tree

State

Visited / expanded tree node

State

Frontier node

State

NOT a Frontier node

State

Arad

BETTER PATH
in REACHED

$$f(Fagaras) = 140 + 99 + \textcolor{blue}{176} = 415$$

f(Timisoara) = 447

Fagaras

Oradea

Rimnicu
Vilcea (RV)

Zerind

Sibiu

Timisoara

Arad

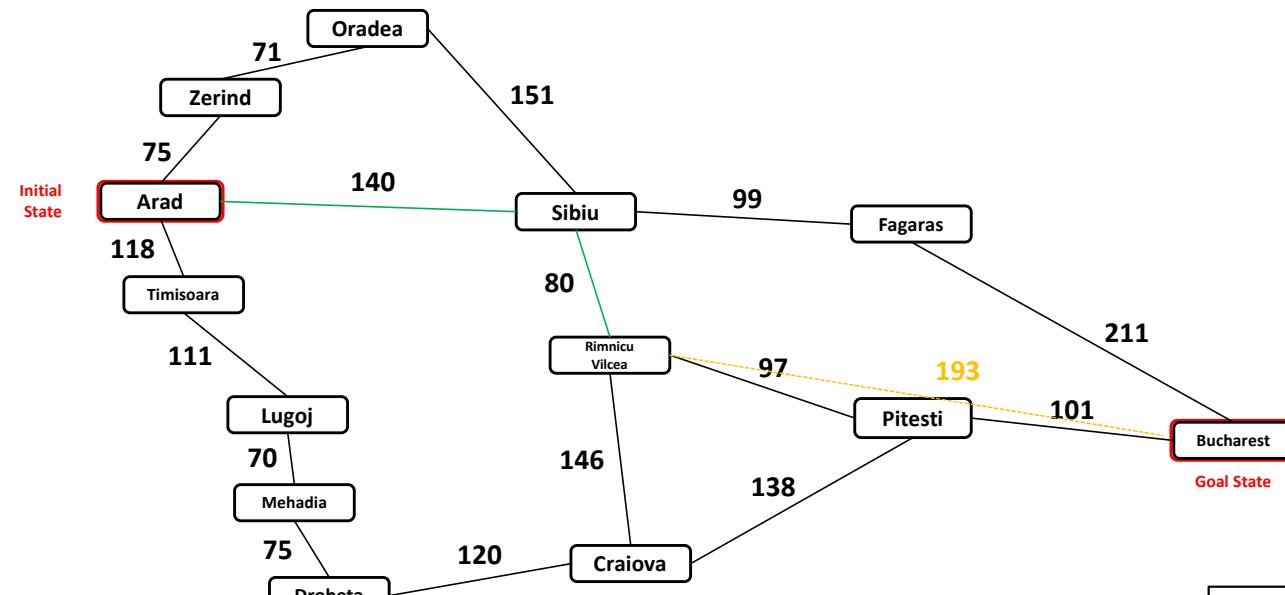
Zerind

Oradea

Rimnicu
Vilcea (RV)

Sibiu

Dracula's Roadtrip: A* Search



Visited / expanded tree node

Frontier node

NOT a Frontier node

Arad

BETTER PATH
in REACHED

$$f(\text{Fagaras}) = 140 + 99 + \textcolor{blue}{176} = 415$$

Fagaras

$$f(\text{Oradea}) = 140 + 151 + 380 = 671$$

Oradea

Rimnicu Vilcea (RV)

$$f(\text{RV}) = 140 + 80 + \textcolor{blue}{193} = 413$$

MINIMUM f()

Arad

Timisoara

Zerind

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

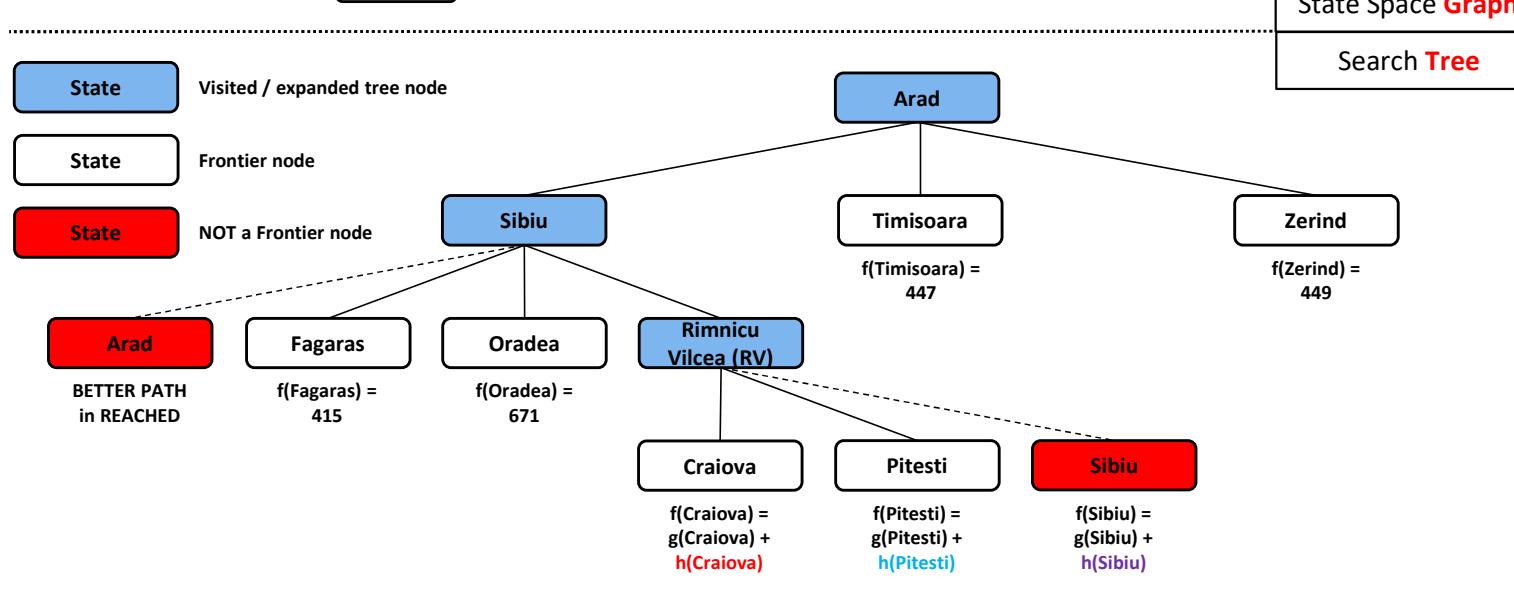
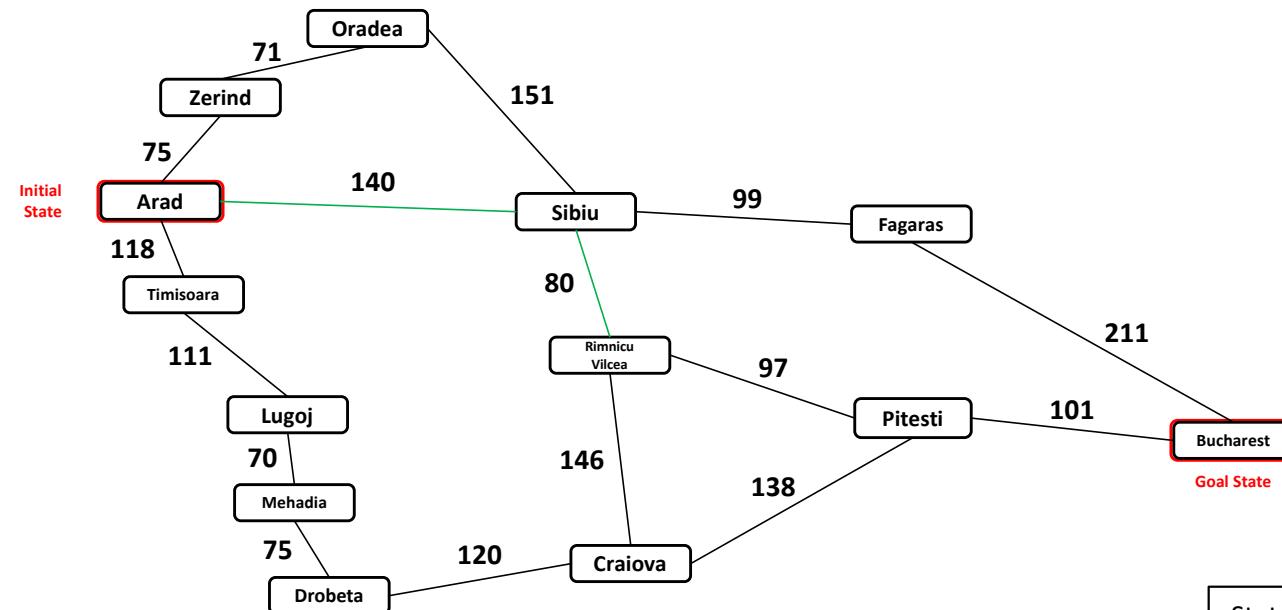
Timisoara 329

Urziceni 80

Vaslui 199

Zerind 374

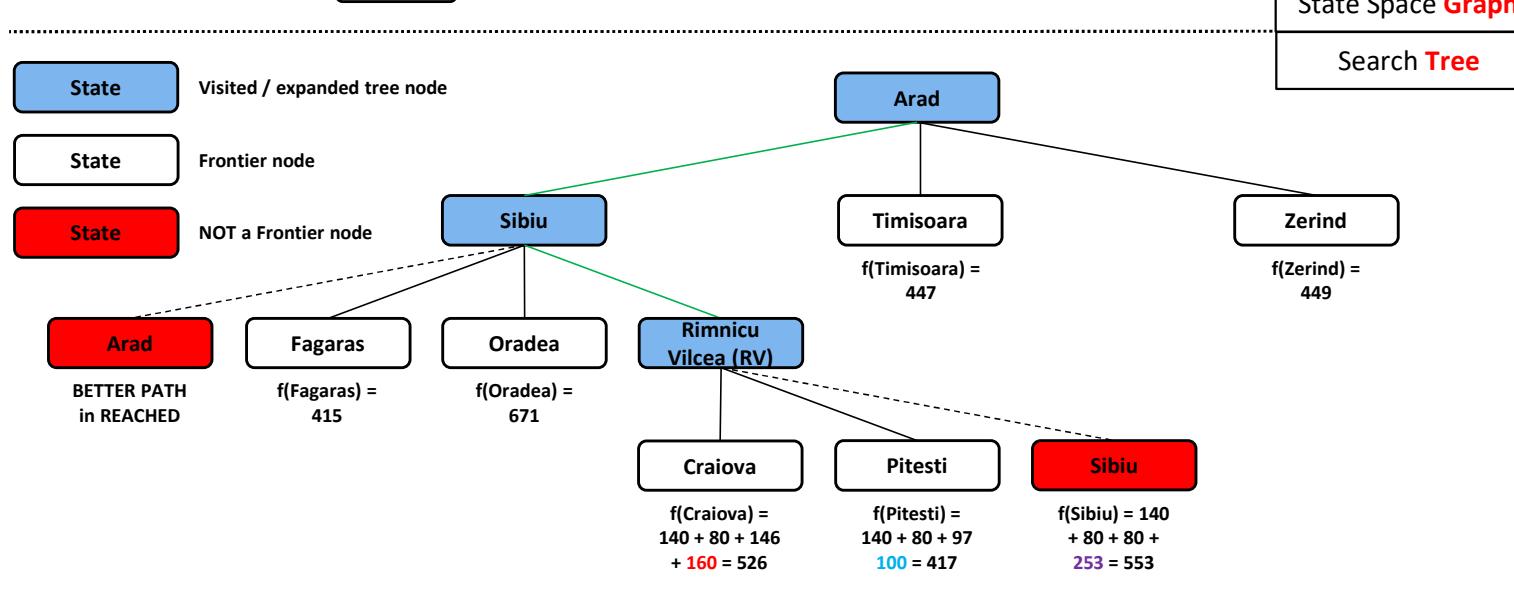
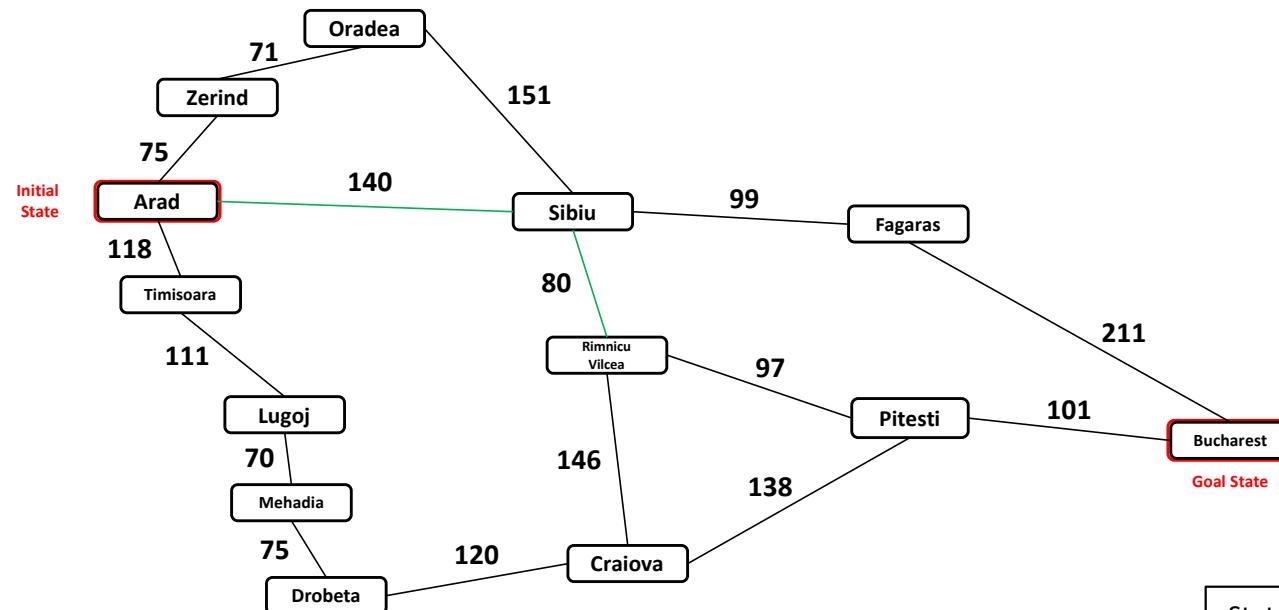
Dracula's Roadtrip: A* Search



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

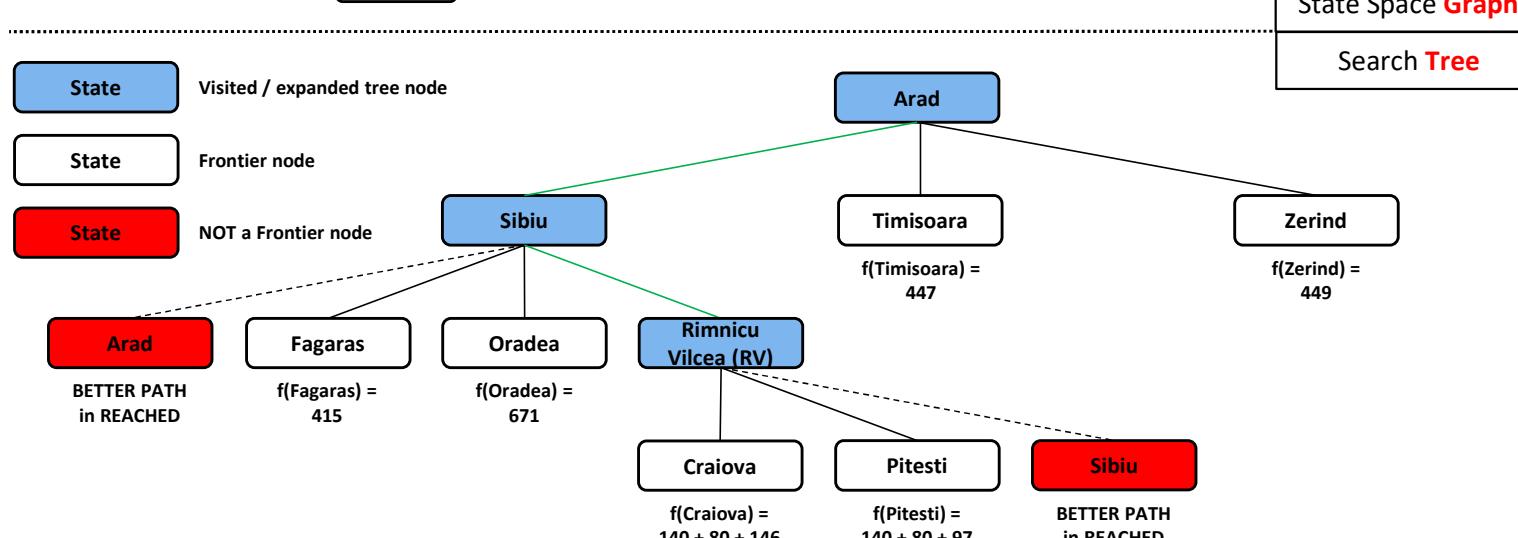
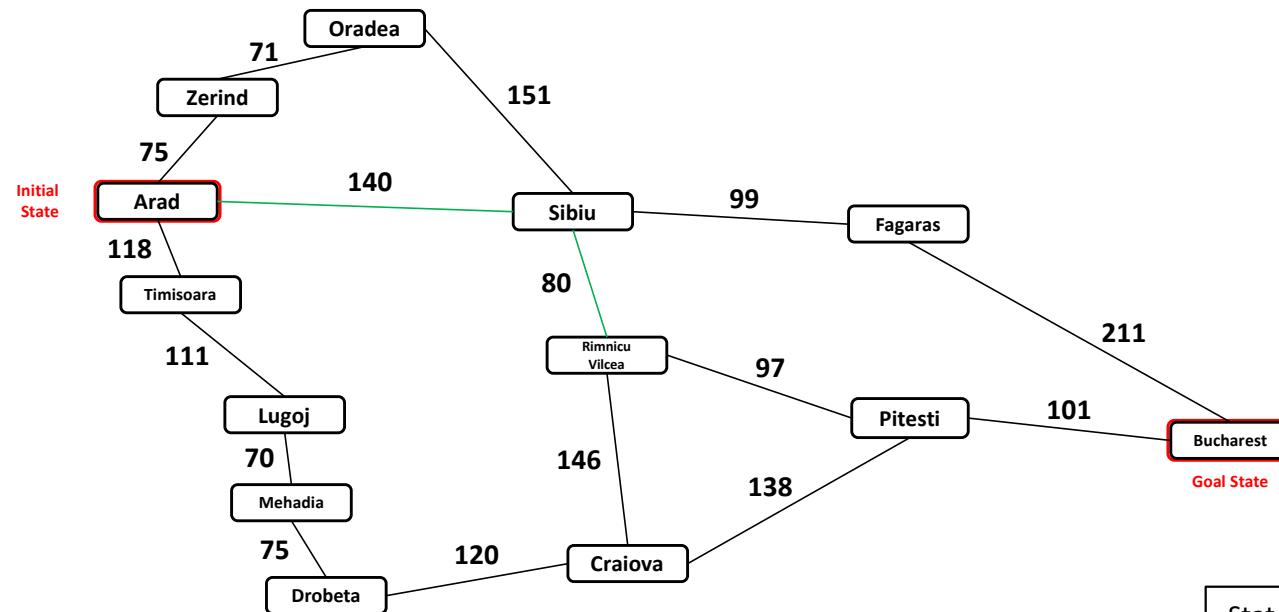
Dracula's Roadtrip: A* Search



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

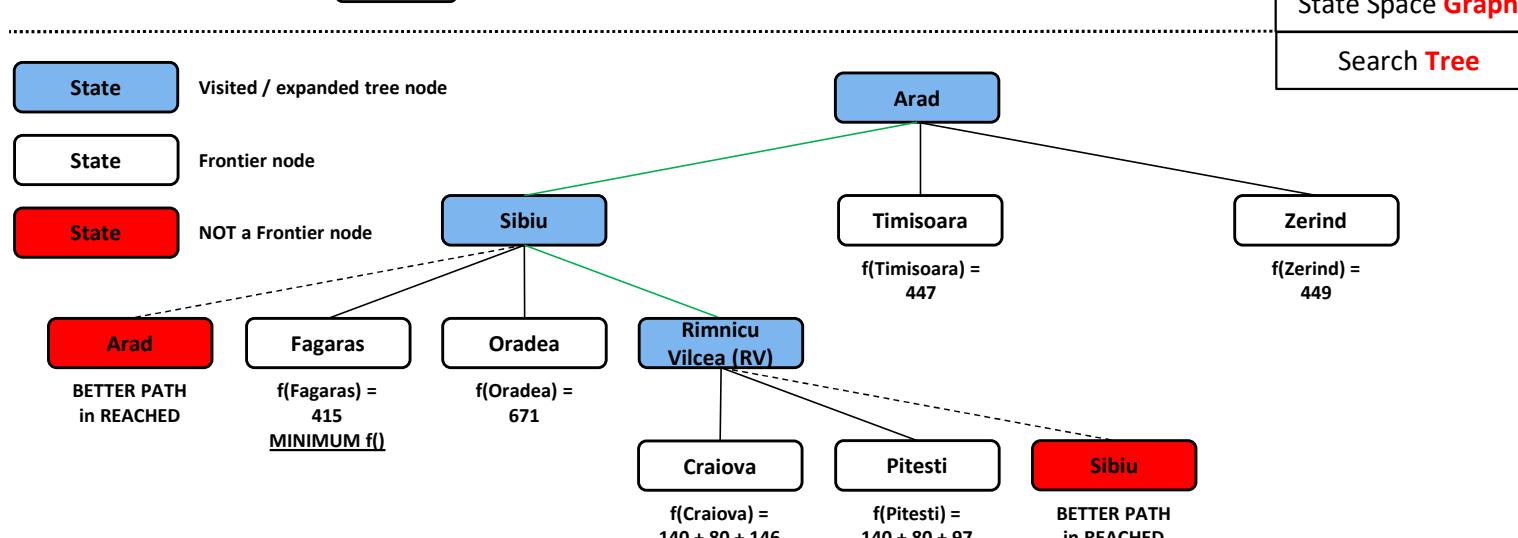
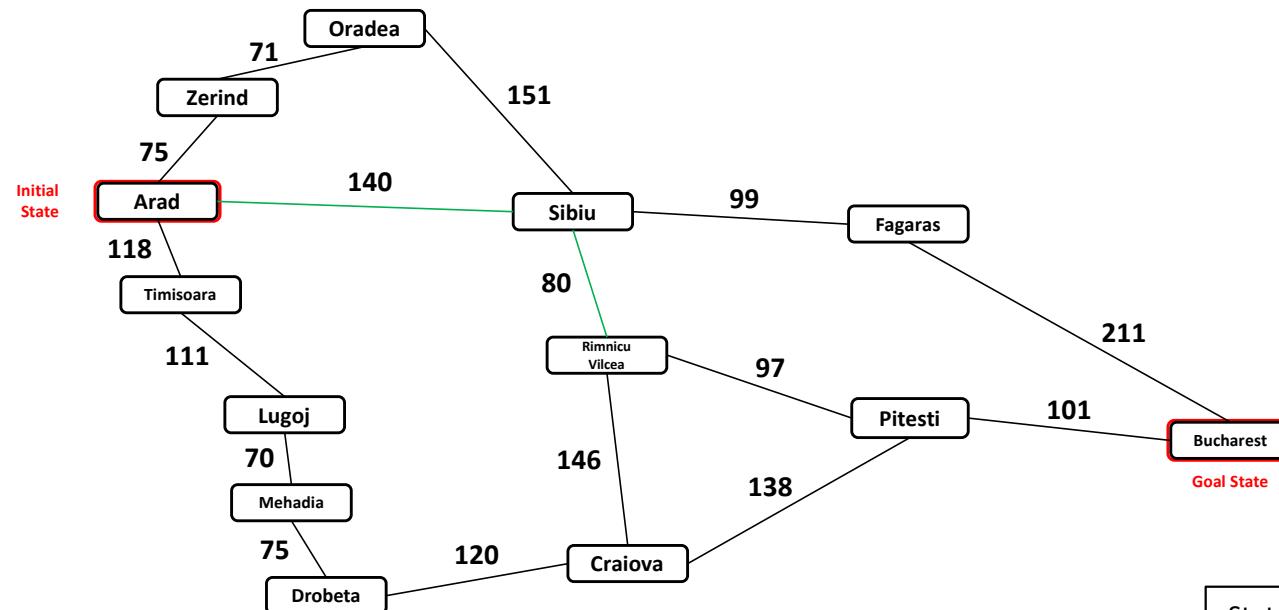
Dracula's Roadtrip: A* Search



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

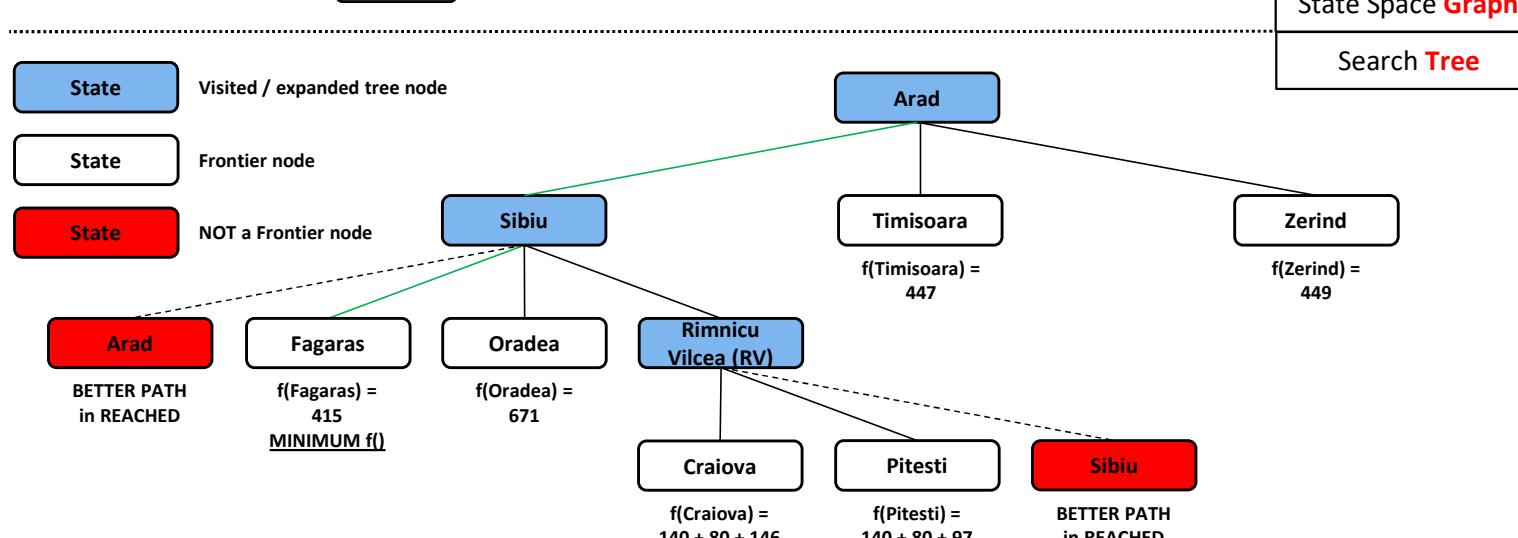
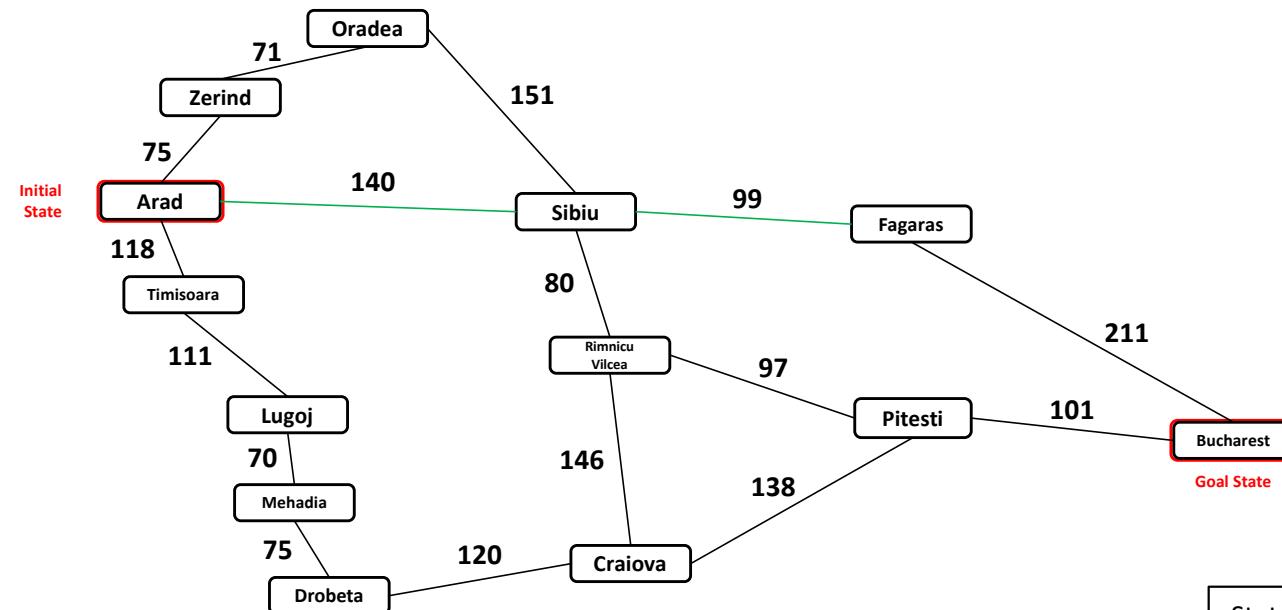
Dracula's Roadtrip: A* Search



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

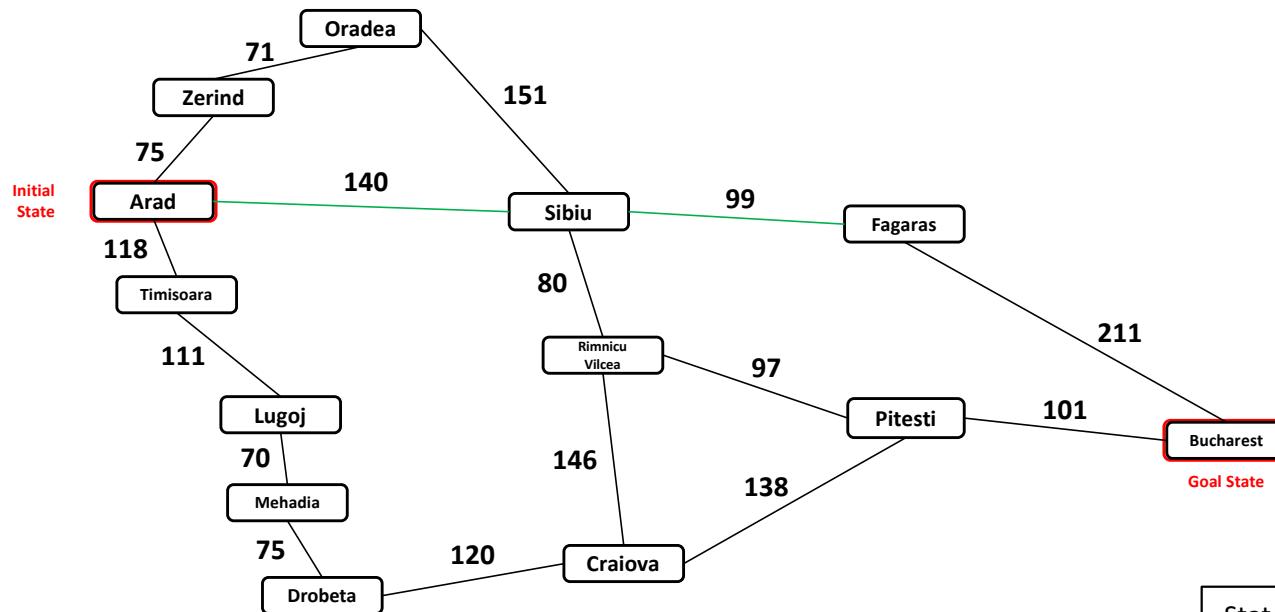
Dracula's Roadtrip: A* Search



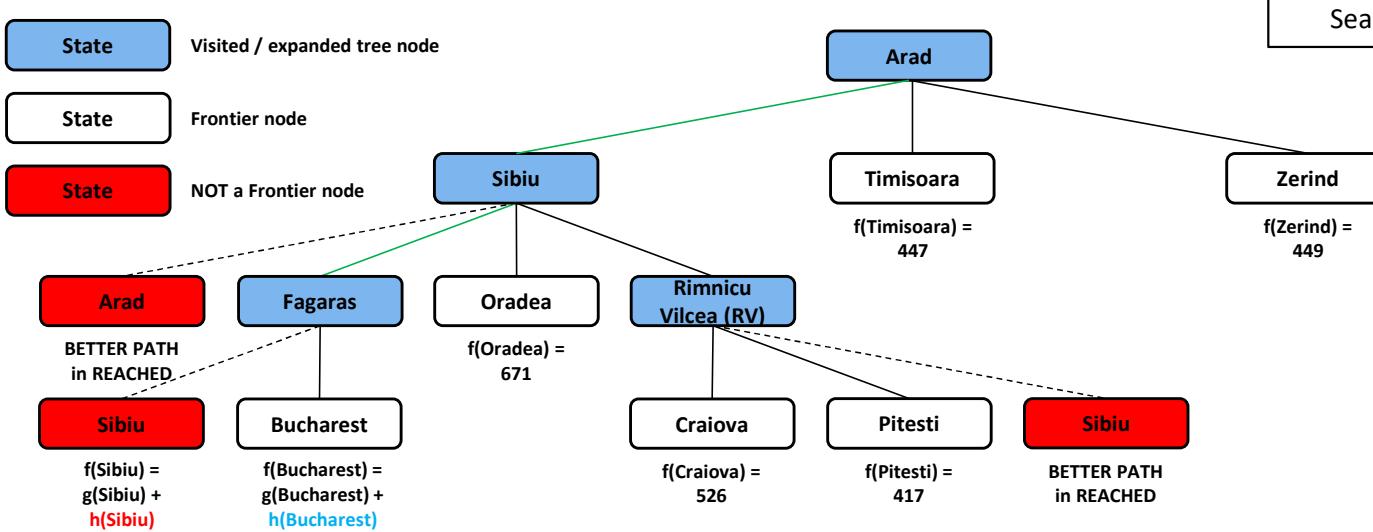
Straight-line distance to Bucharest ($h(\text{State})$):

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

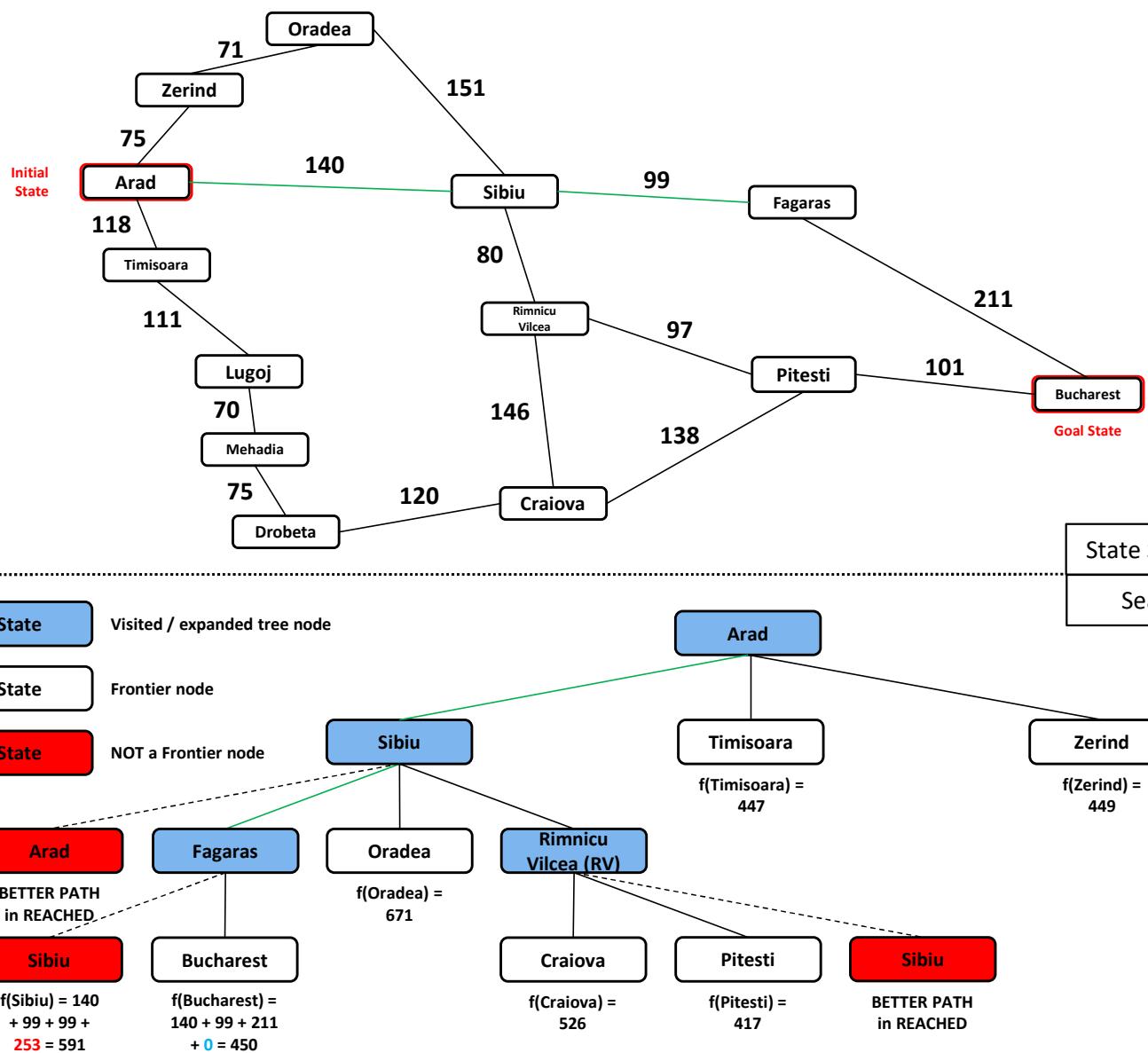
Dracula's Roadtrip: A* Search



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

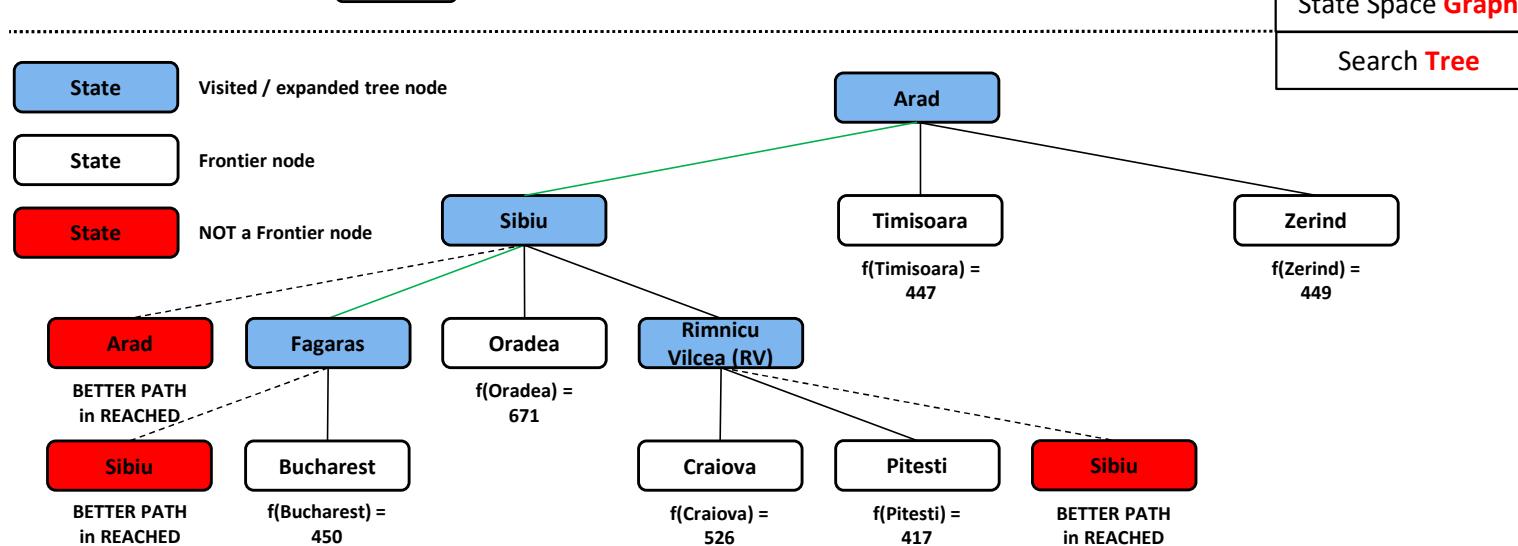
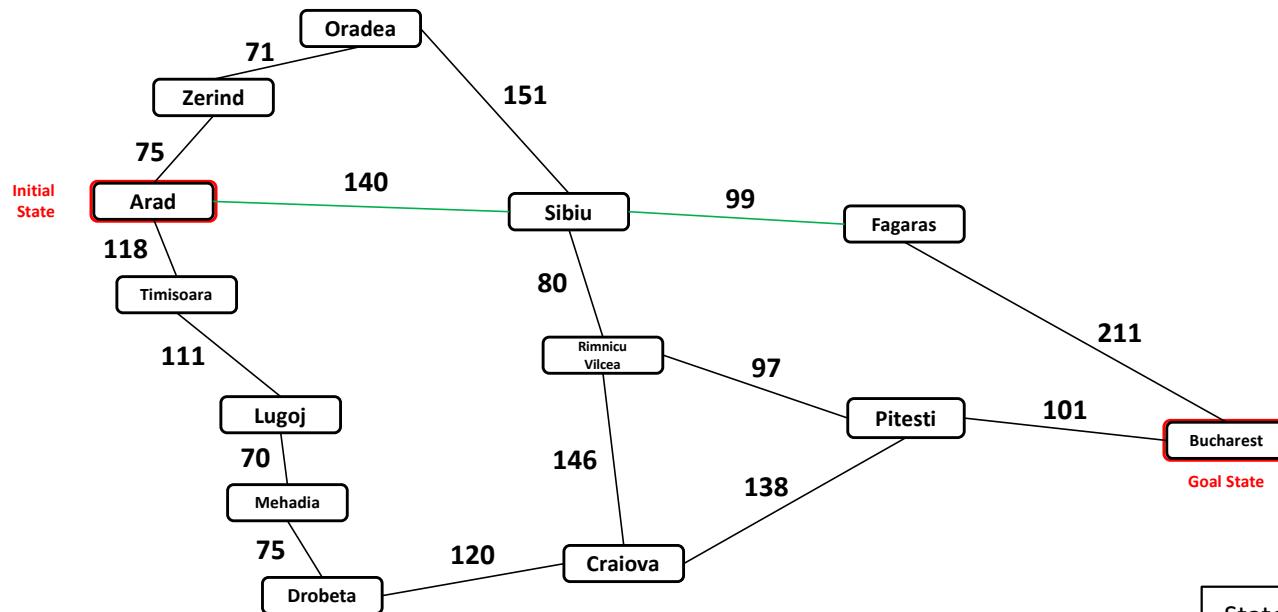


Dracula's Roadtrip: A* Search



<u>Straight-line distance to Bucharest (h(State)):</u>	
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

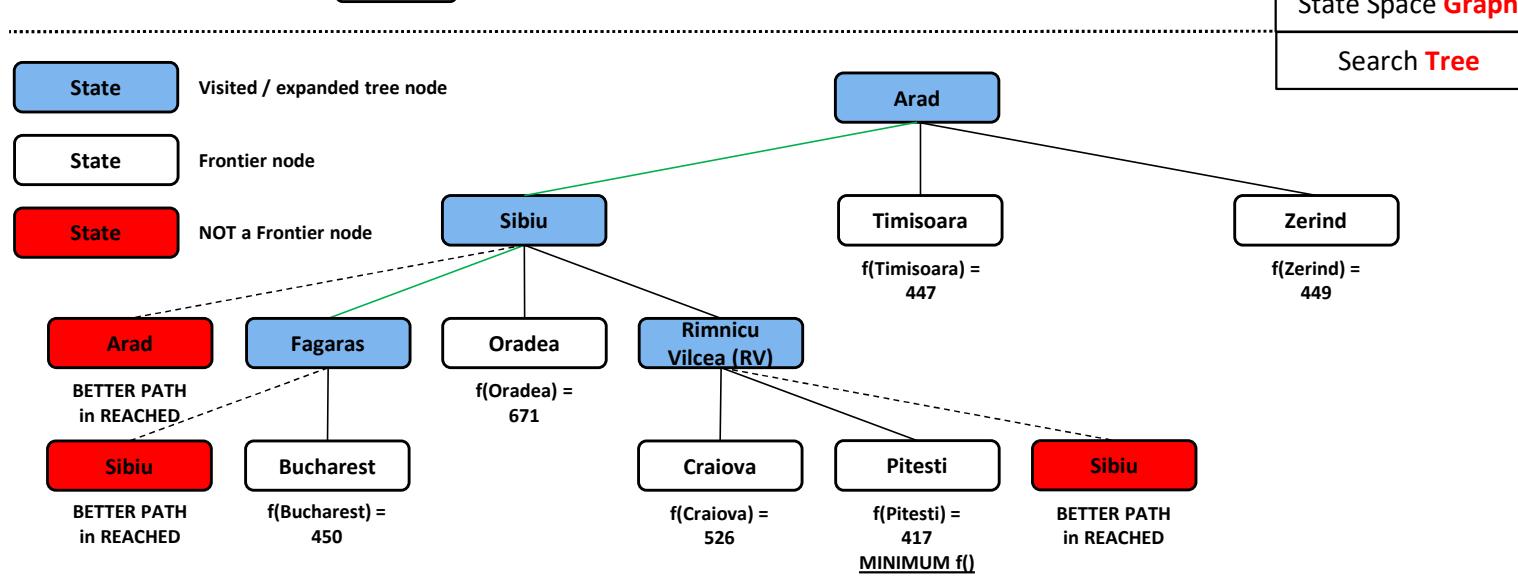
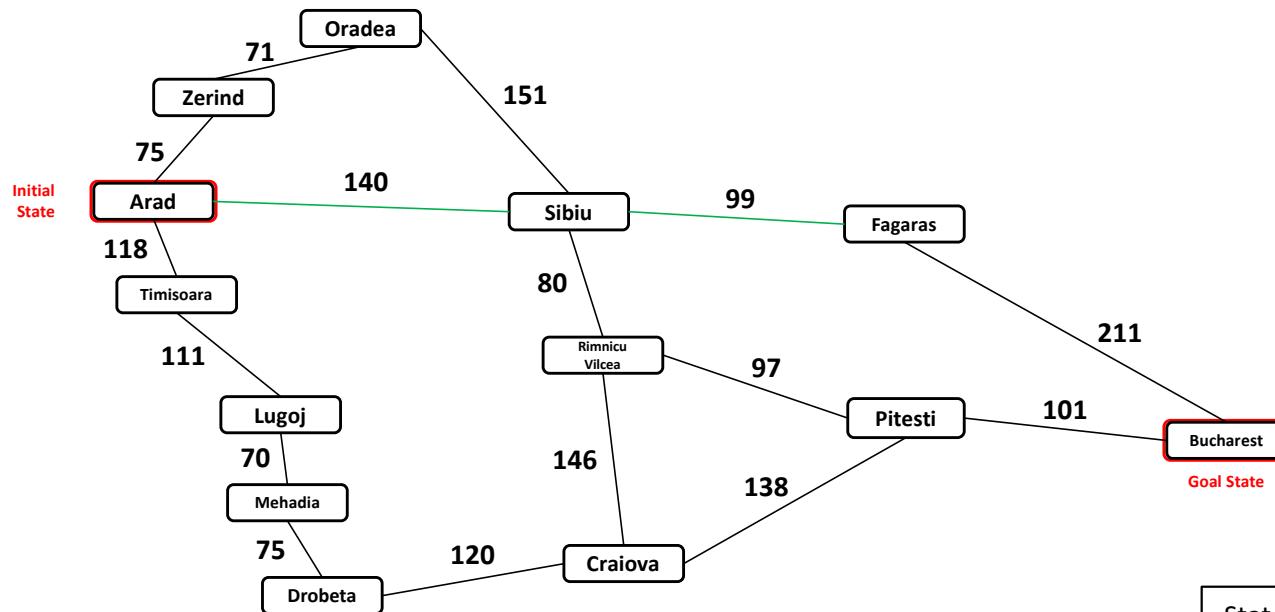
Dracula's Roadtrip: A* Search



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

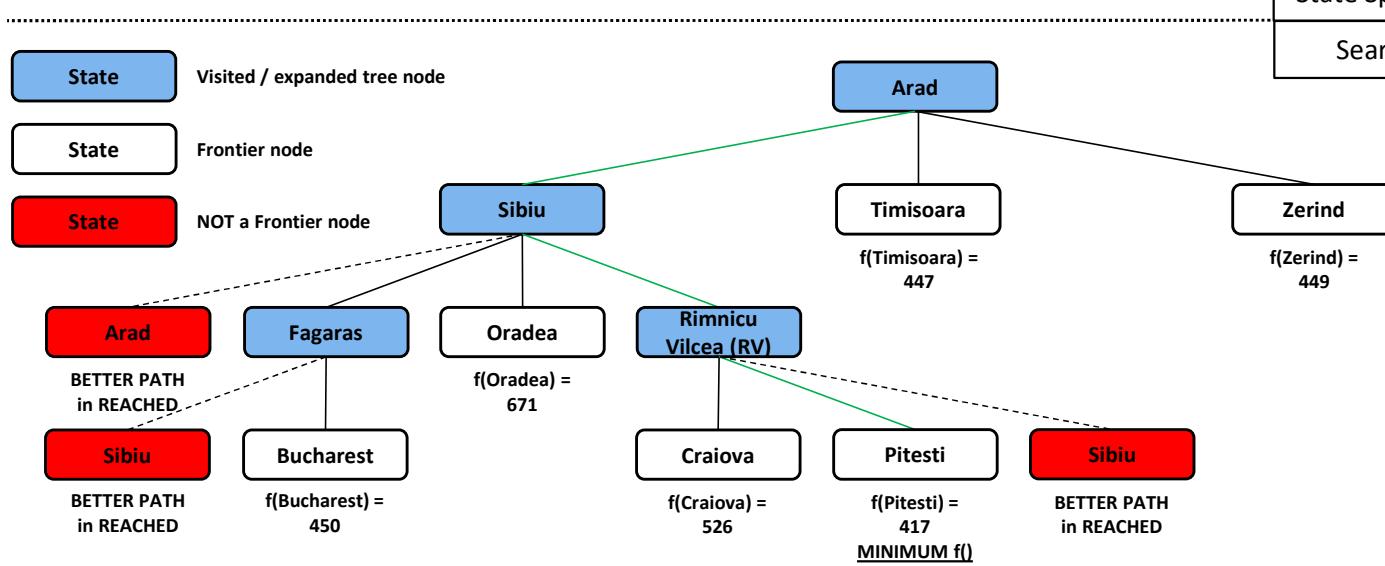
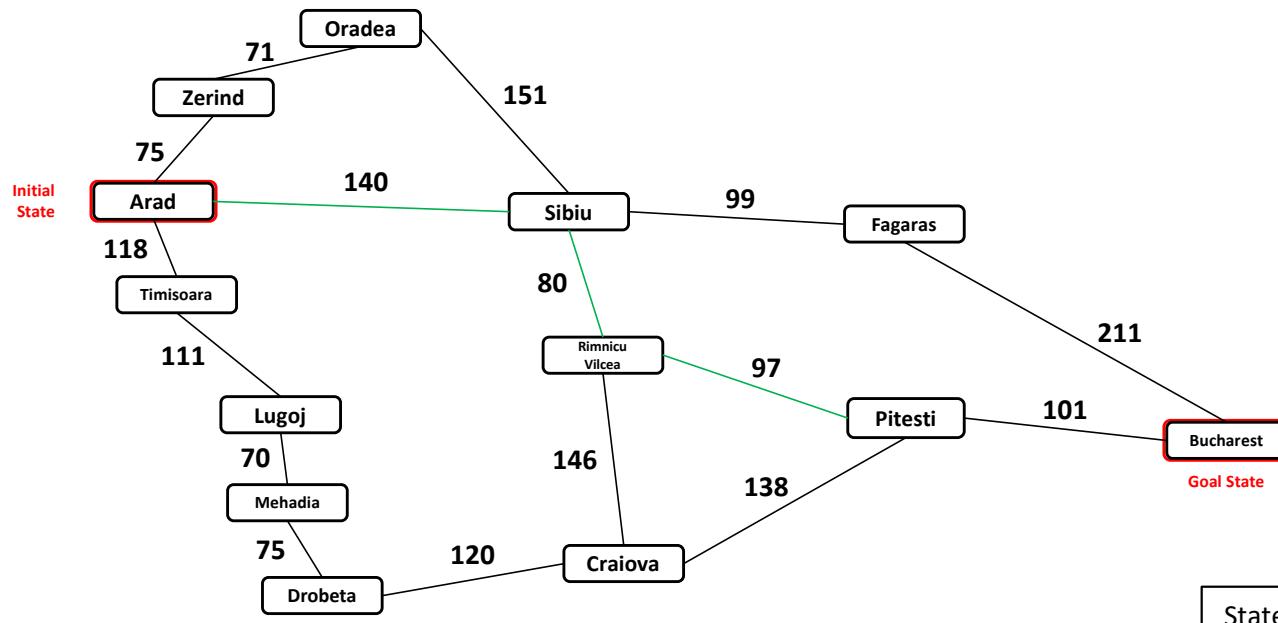
Dracula's Roadtrip: A* Search



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

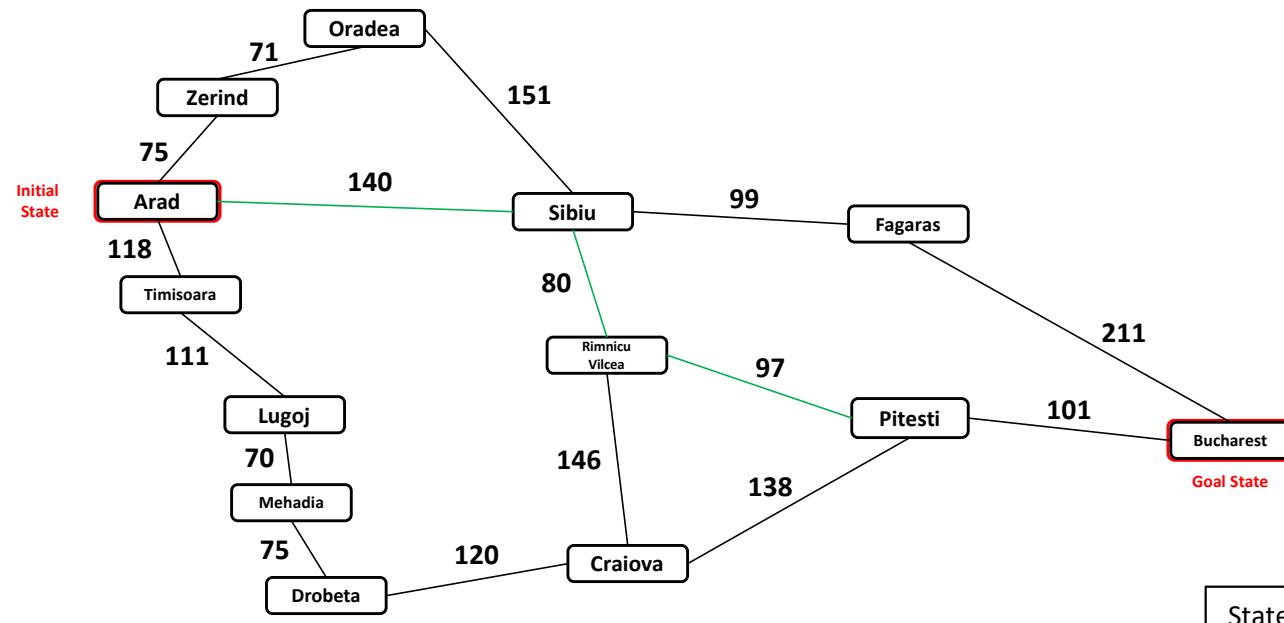
Dracula's Roadtrip: A* Search



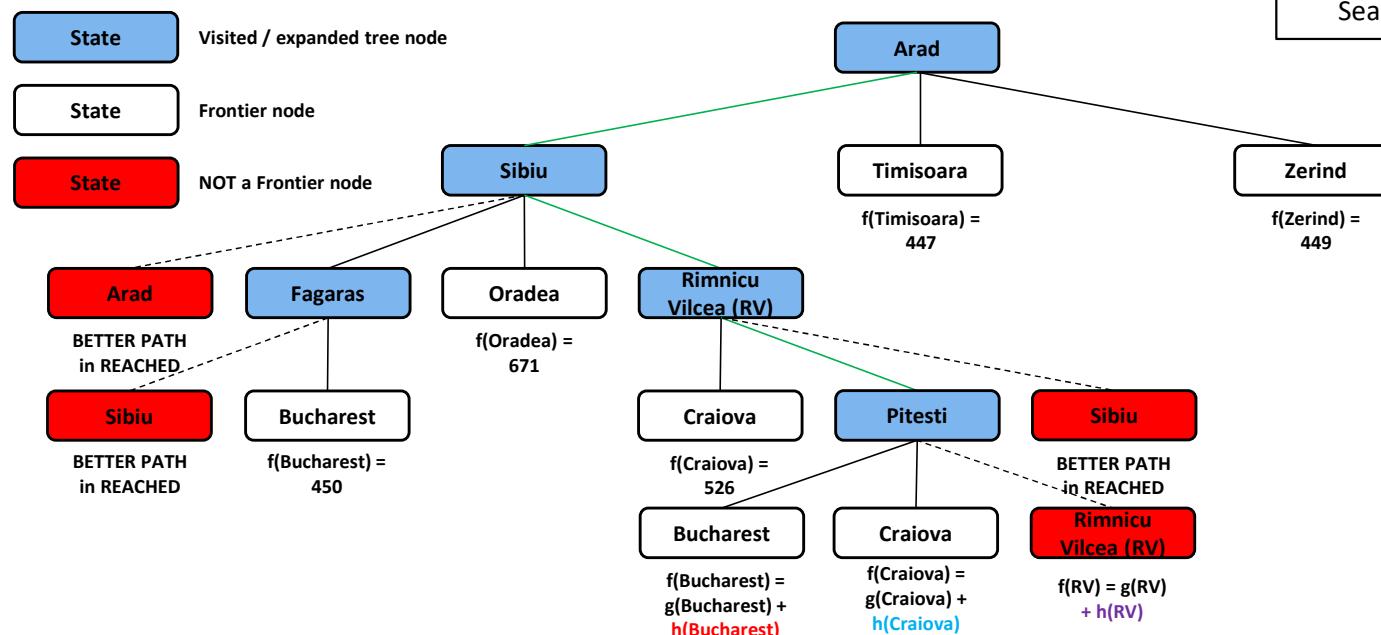
Straight-line distance to Bucharest ($h(\text{State})$):

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

Dracula's Roadtrip: A* Search



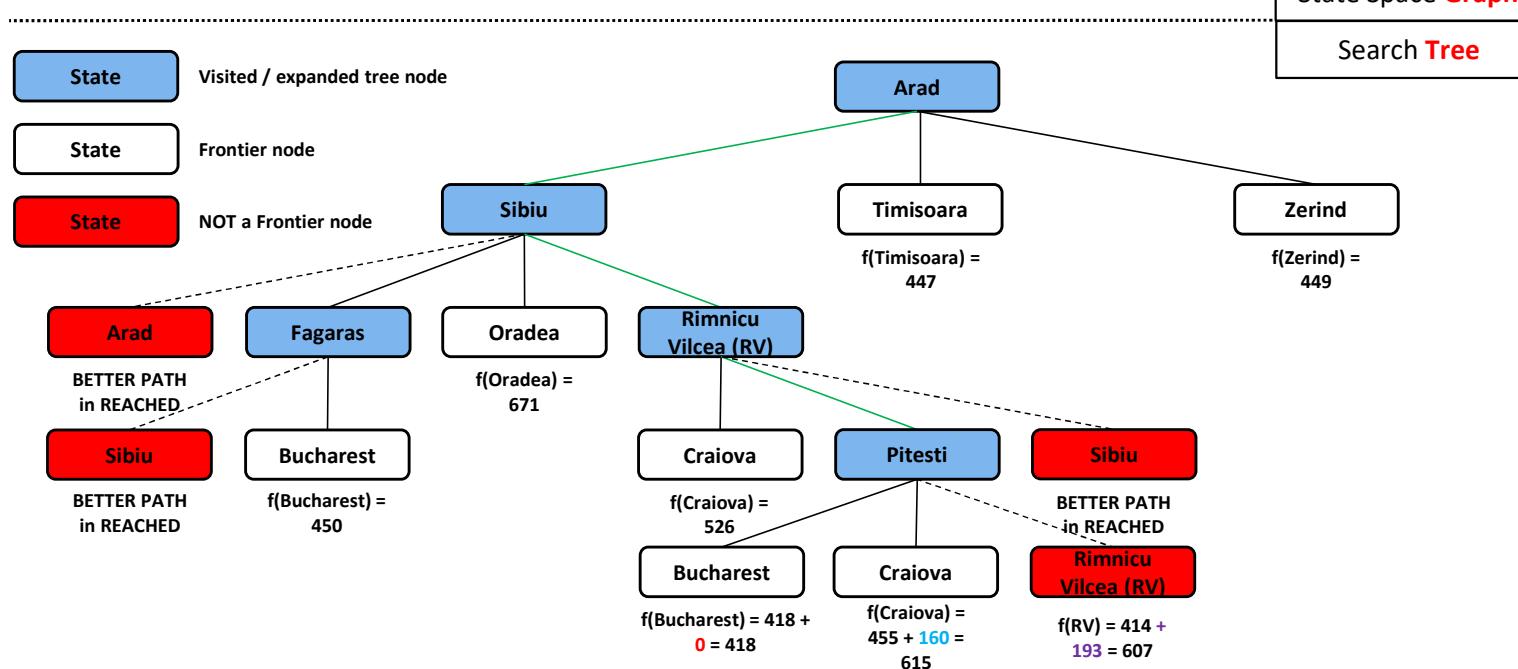
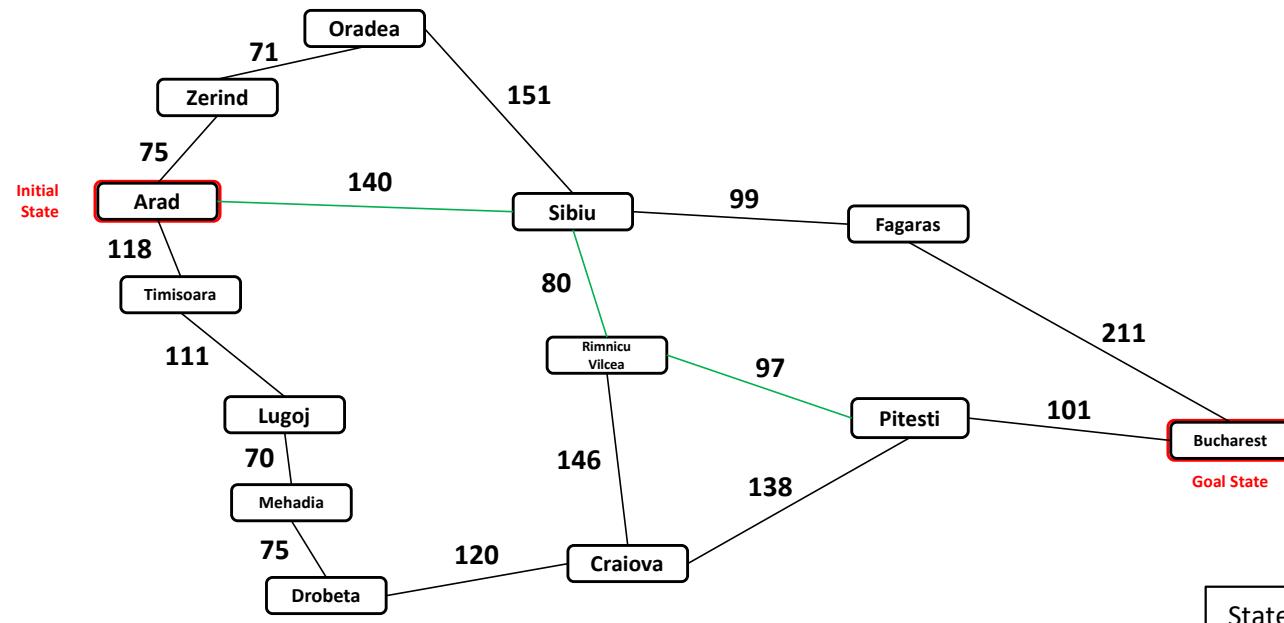
State Space Graph
Search Tree



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

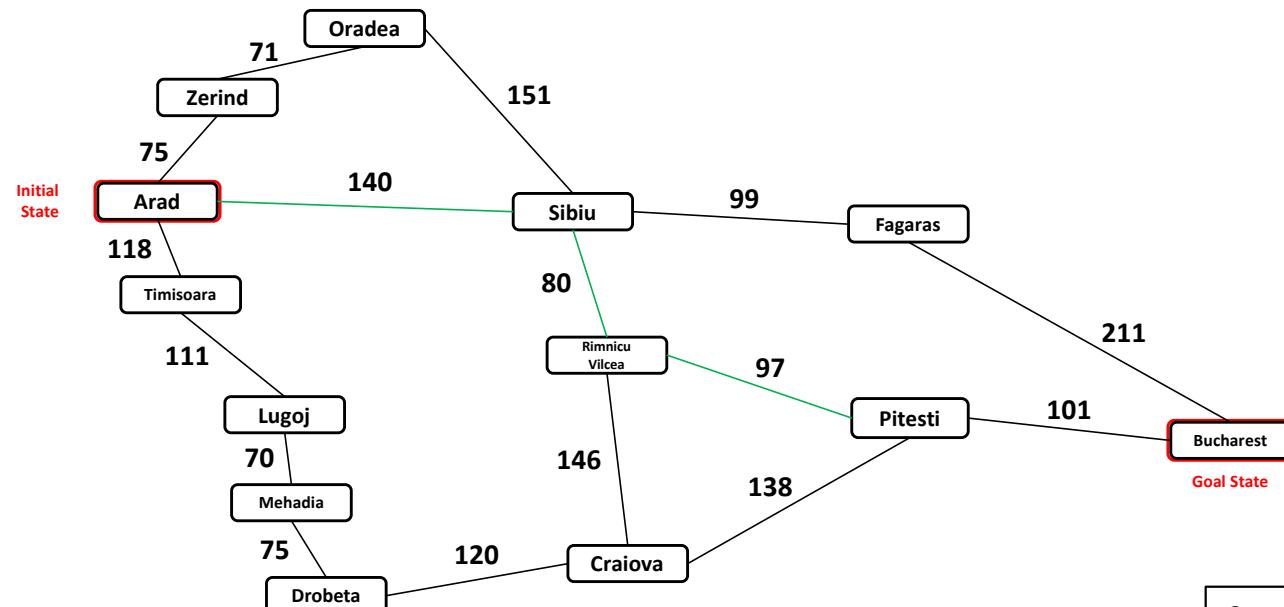
Dracula's Roadtrip: A* Search



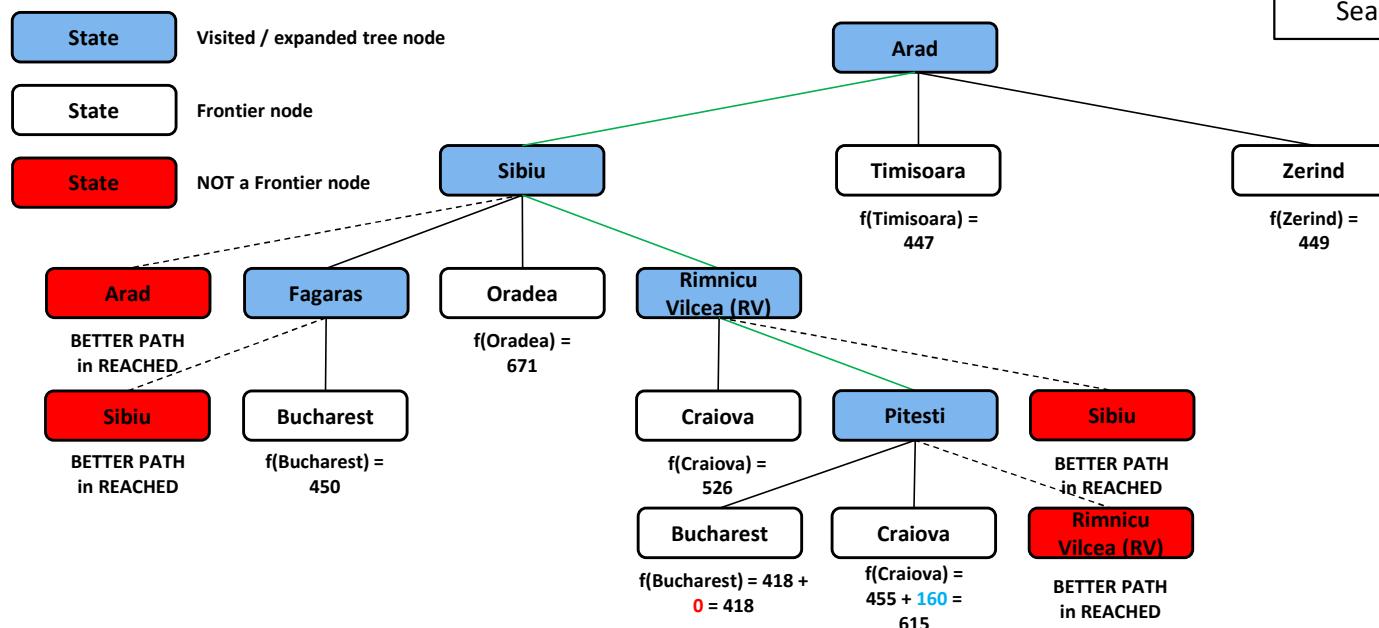
Straight-line distance to Bucharest (h(State)):

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

Dracula's Roadtrip: A* Search



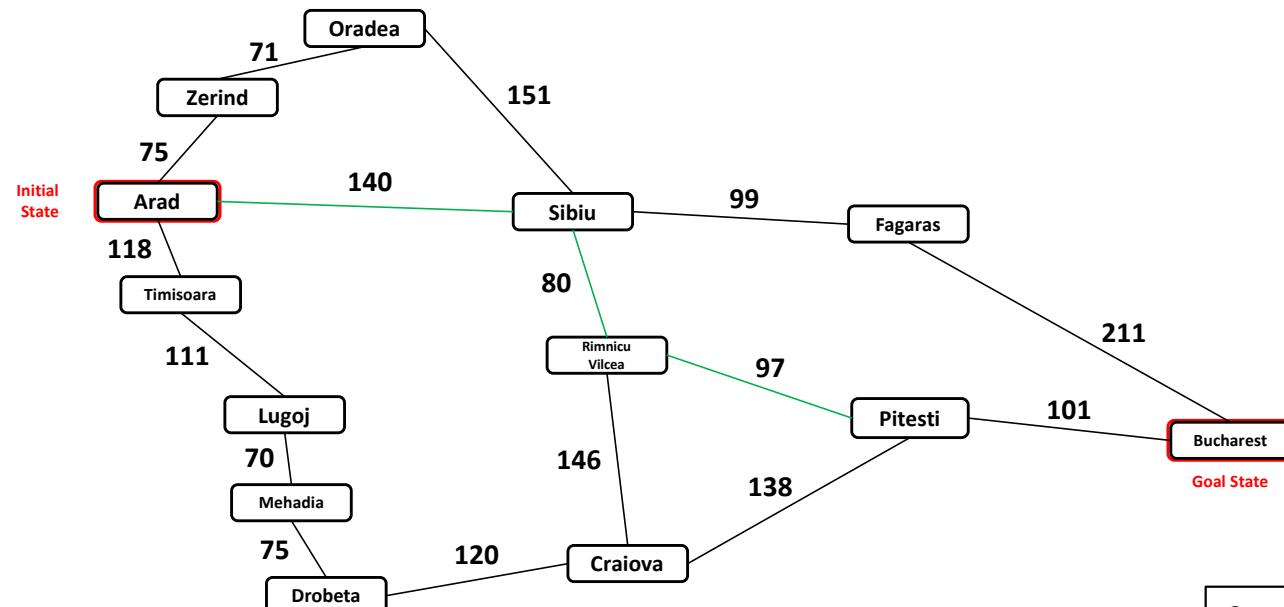
State Space Graph
Search Tree



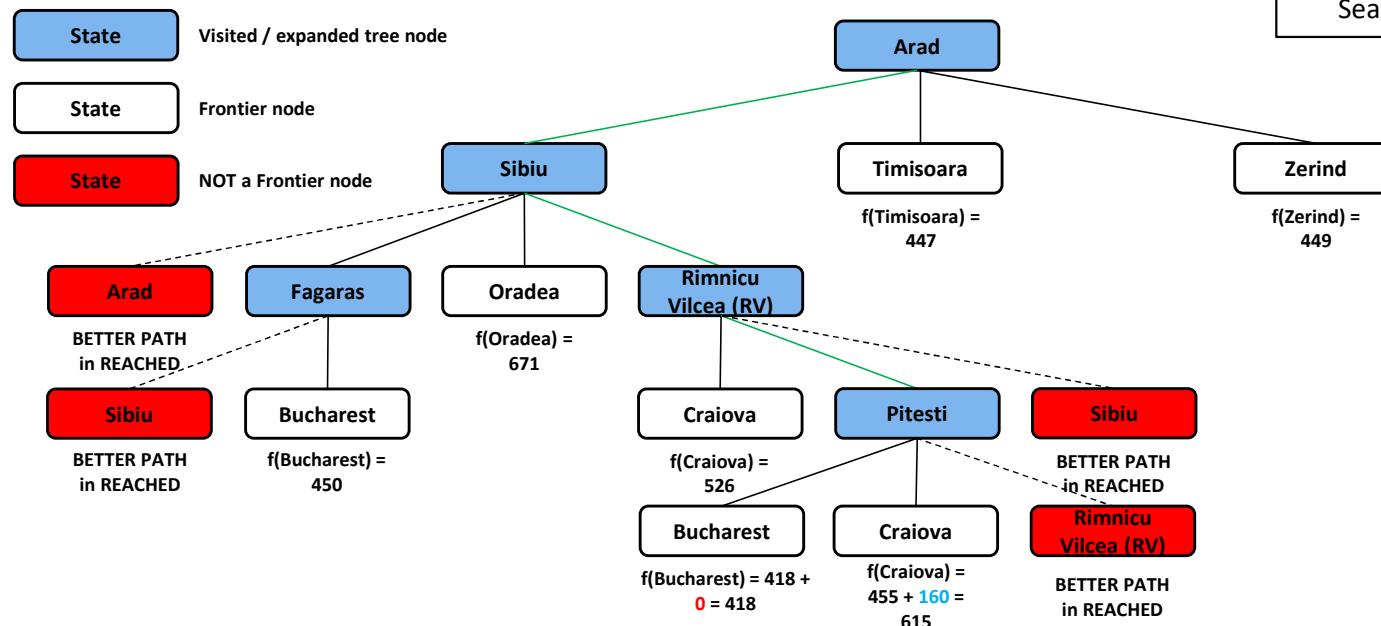
Straight-line distance to Bucharest ($h(\text{State})$):

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

Dracula's Roadtrip: A* Search



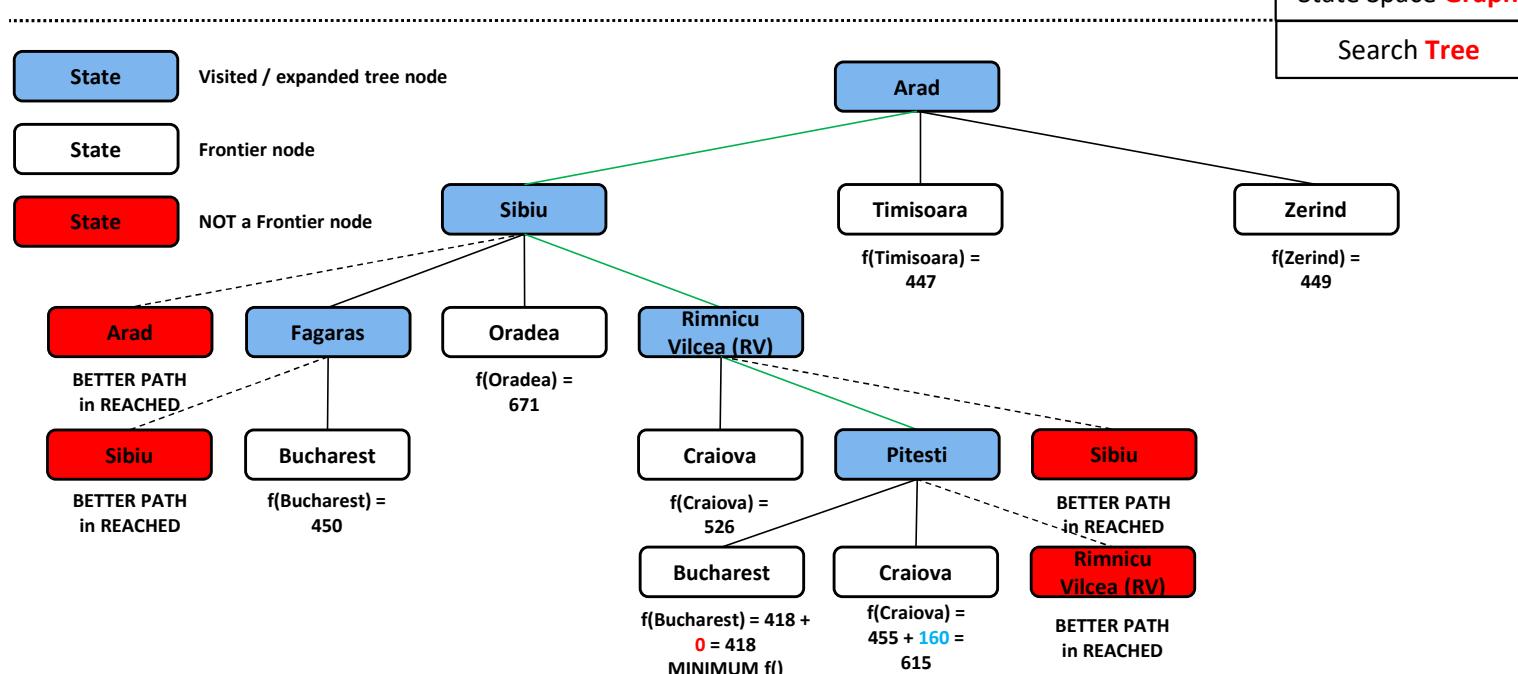
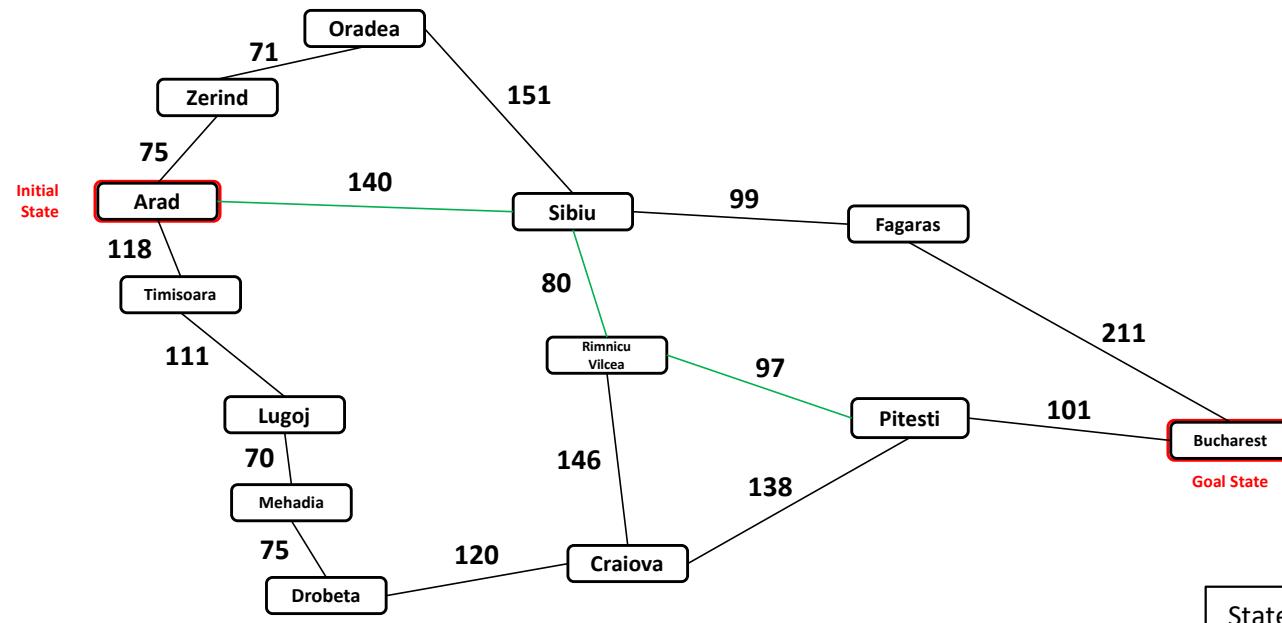
State Space **Graph**
Search **Tree**



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

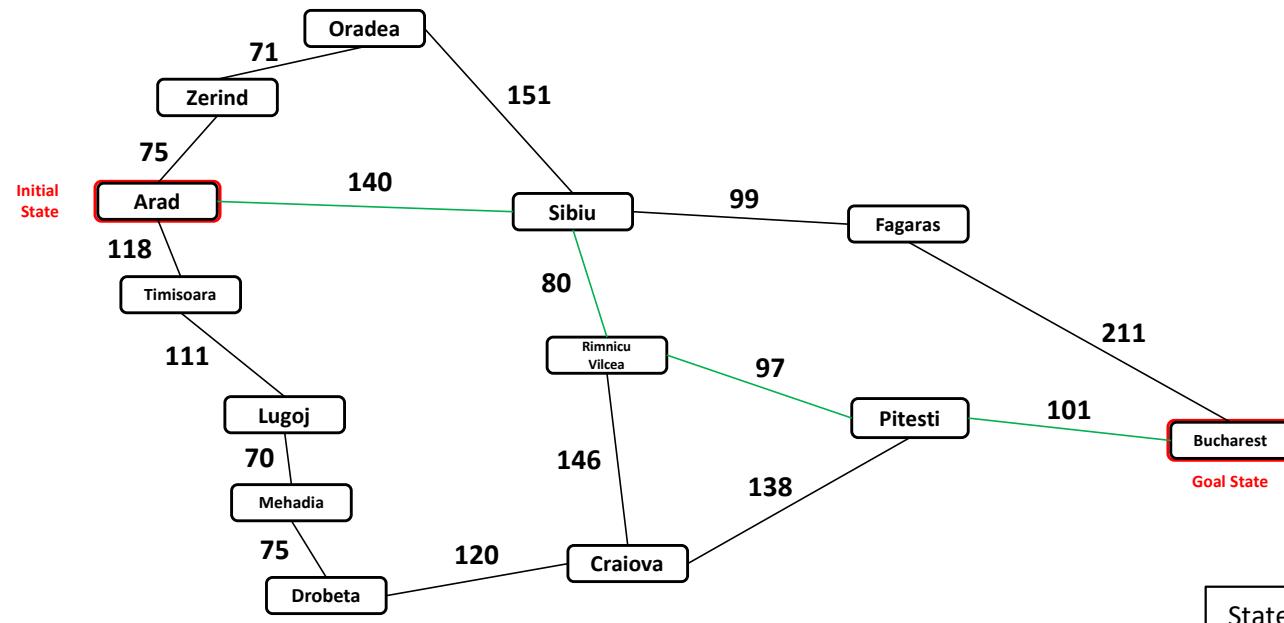
Dracula's Roadtrip: A* Search



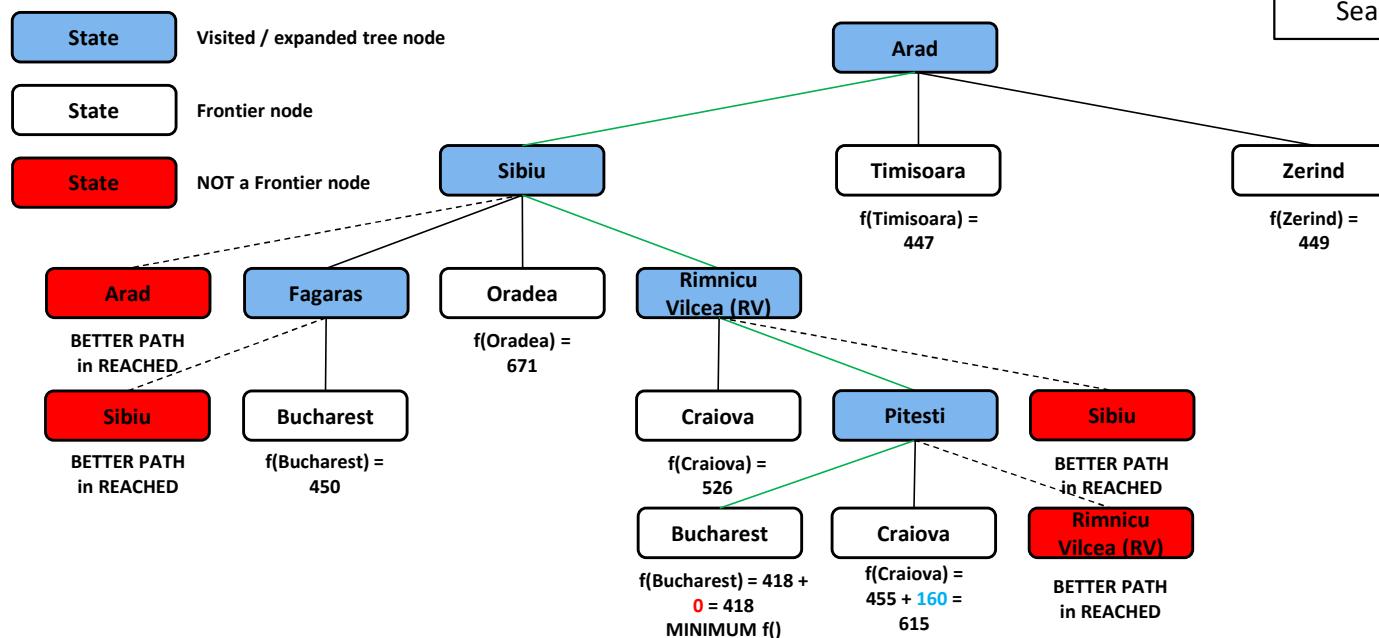
Straight-line distance to Bucharest ($h(\text{State})$):

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

Dracula's Roadtrip: A* Search



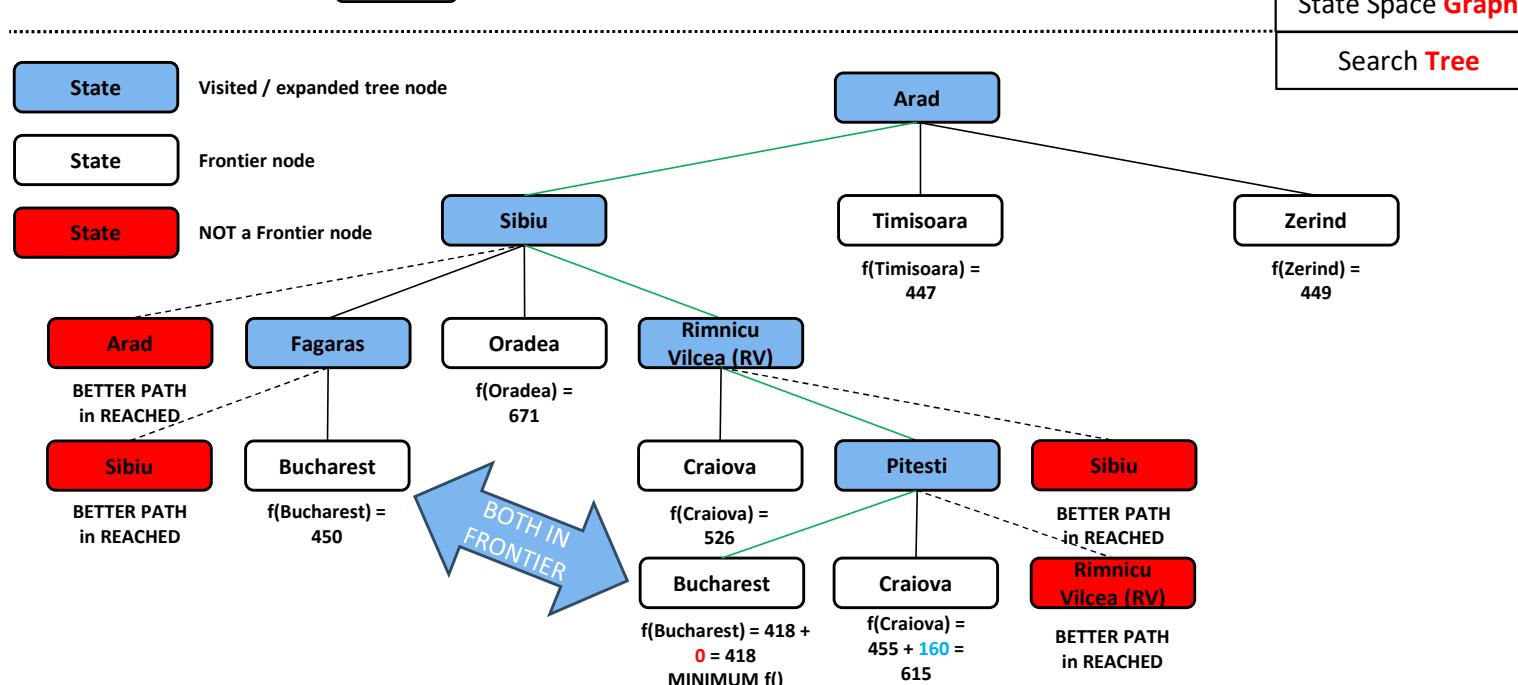
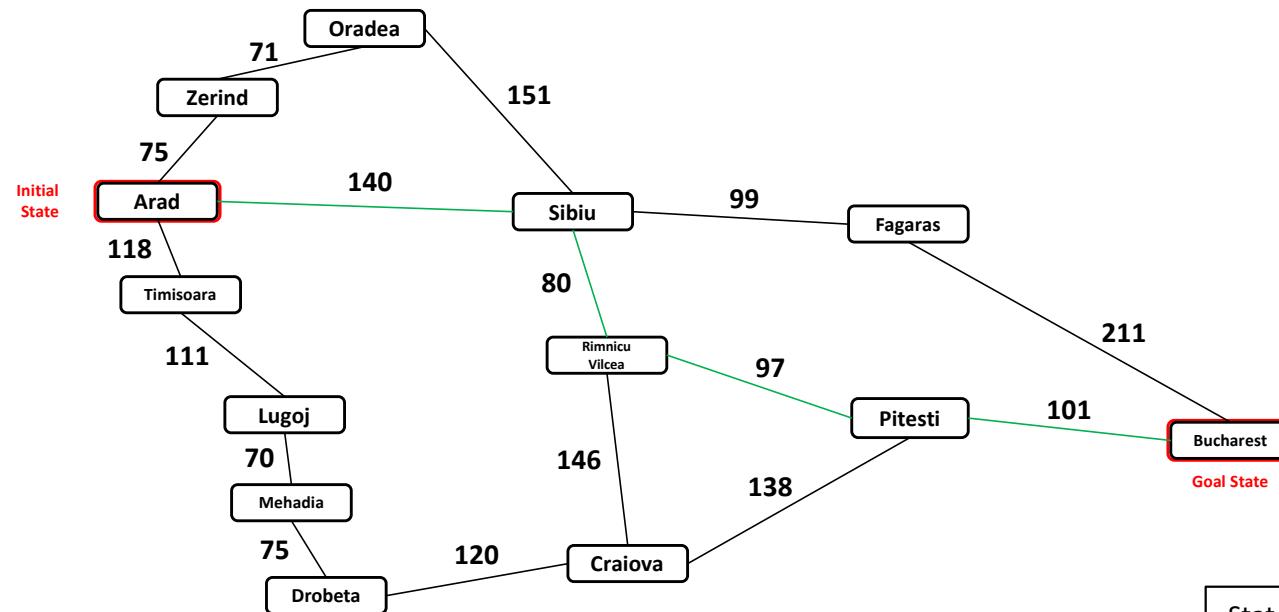
State Space Graph
Search Tree



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

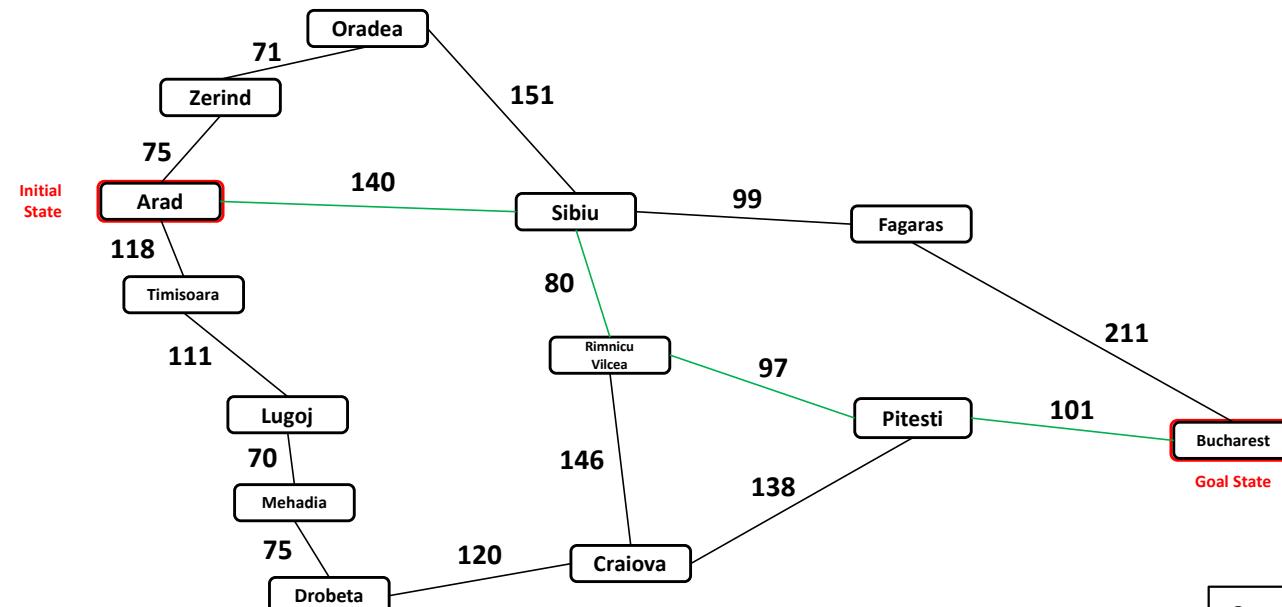
Dracula's Roadtrip: A* Search



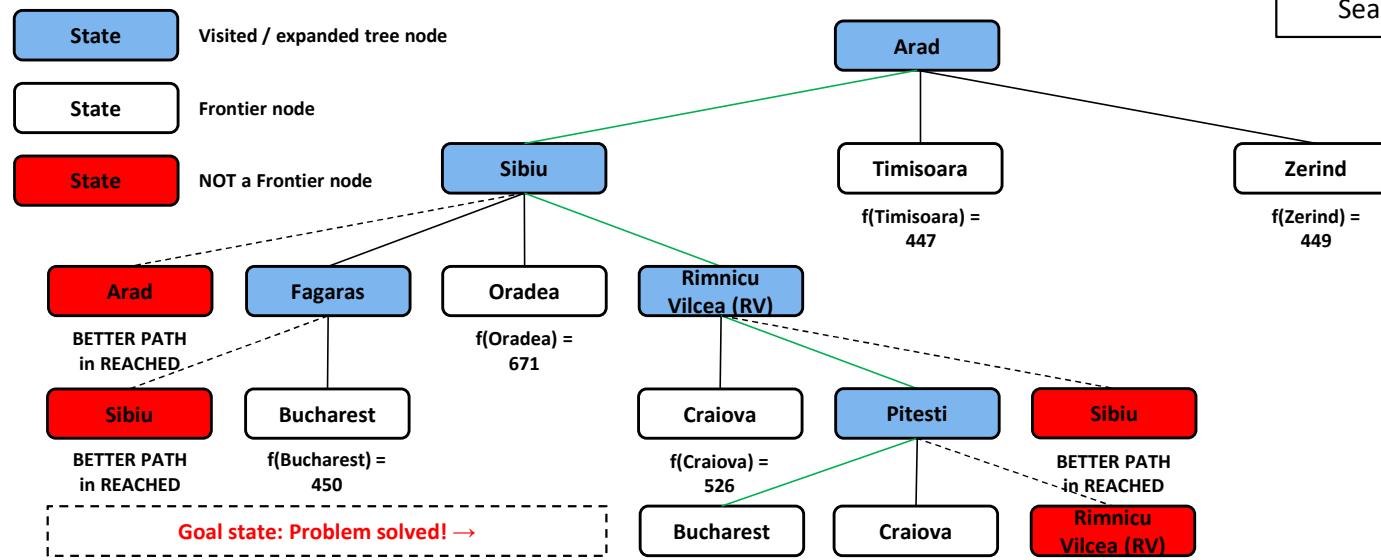
Straight-line distance to Bucharest ($h(\text{State})$):

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

Dracula's 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

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

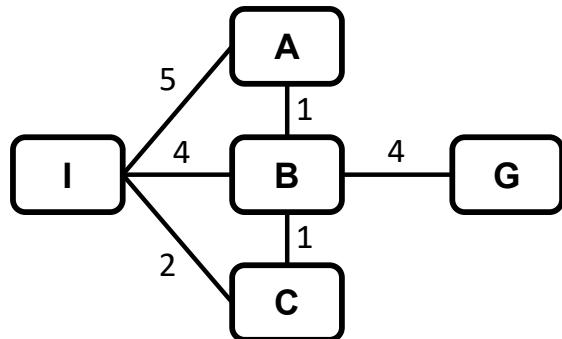
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value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

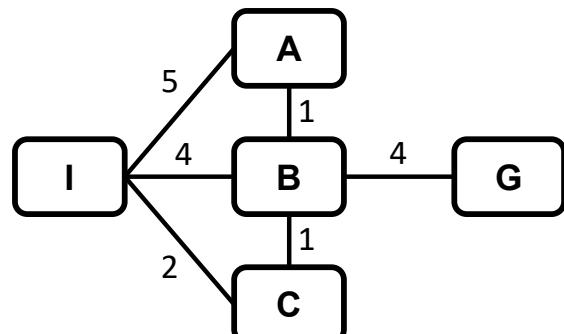
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

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

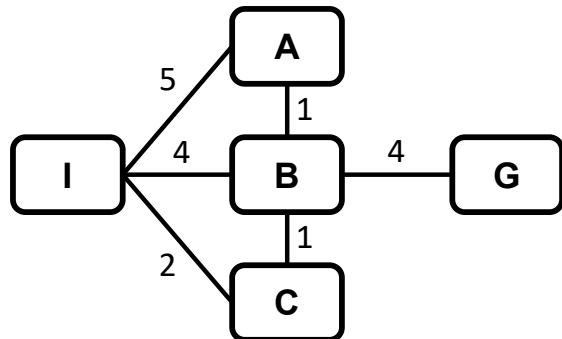
if *s* is not in *reached* **or** *child*.PATH-COST $<$ *reached*[*s*].PATH-COST **then**

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

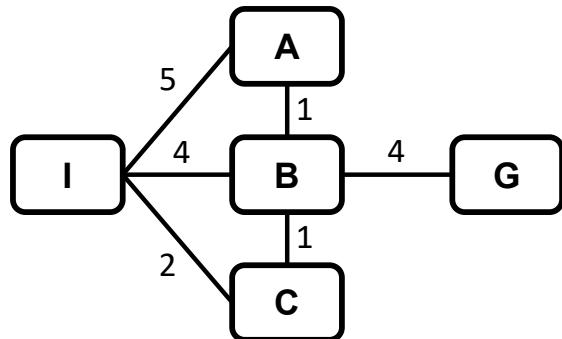
node \rightarrow

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

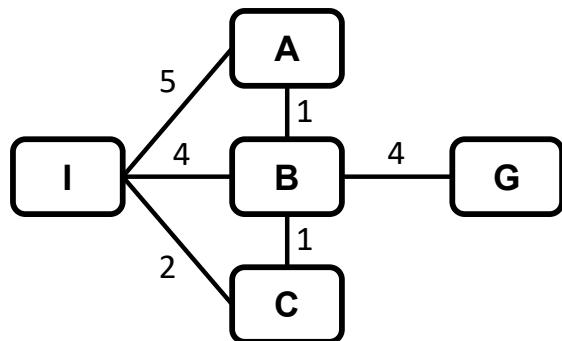


I



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	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

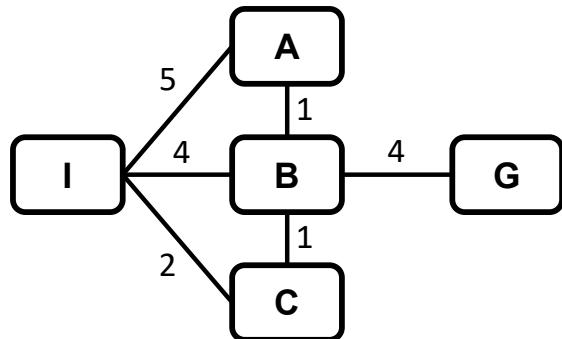


$$f(I) = h(I) = 7$$



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*

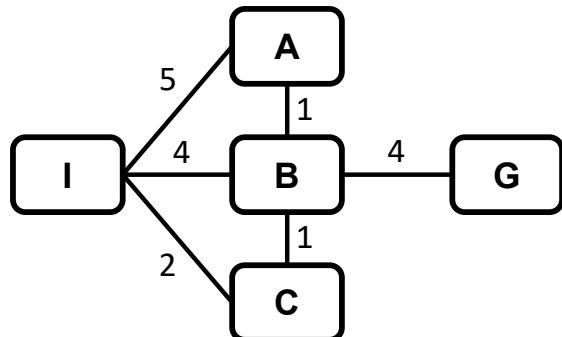
 add *child* to *frontier*

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**

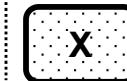
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

return failure

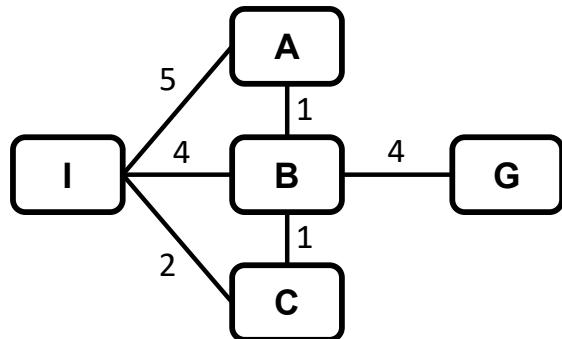


Path cost 0



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*

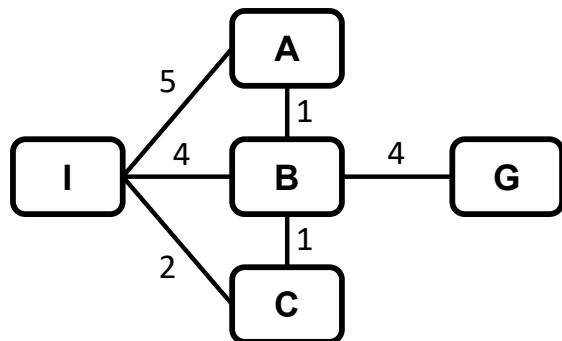
 add *child* to *frontier*

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*

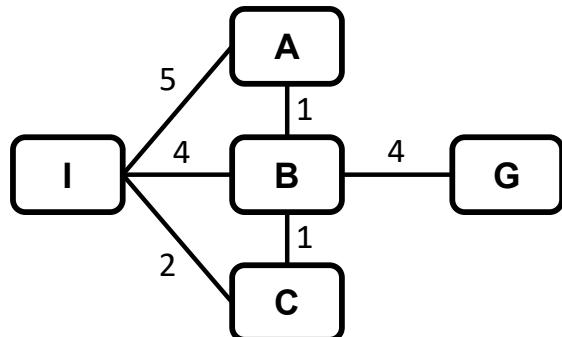
 add *child* to *frontier*

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*

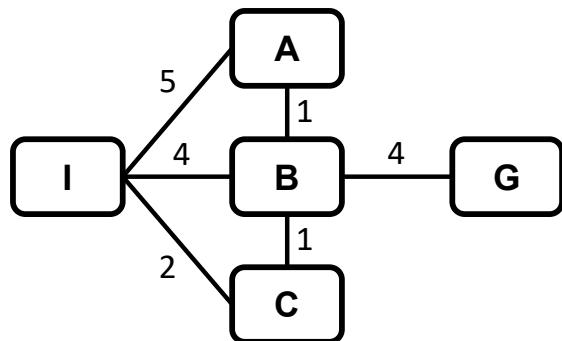
 add *child* to *frontier*

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*

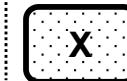
if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

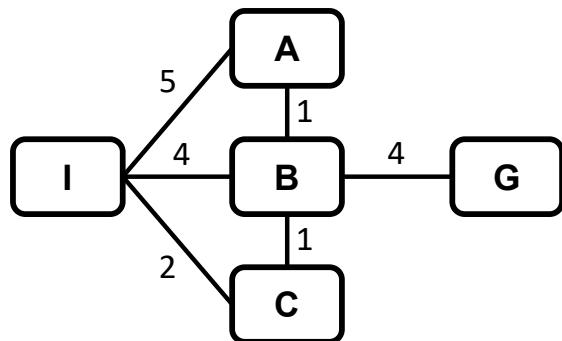
return failure

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*

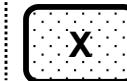
if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

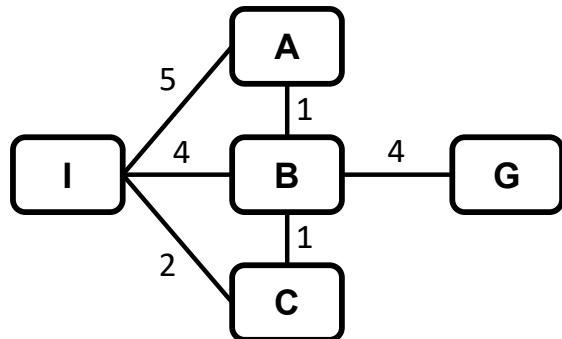
return failure

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*

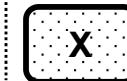
if *s* is not in *reached* **or** *child.PATH-COST* $<$ *reached*[*s*].*PATH-COST* **then**

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

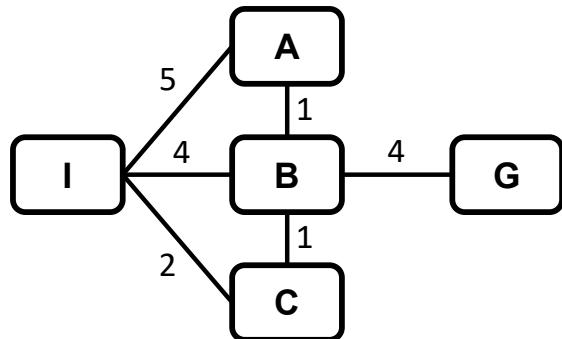
return failure

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 \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*

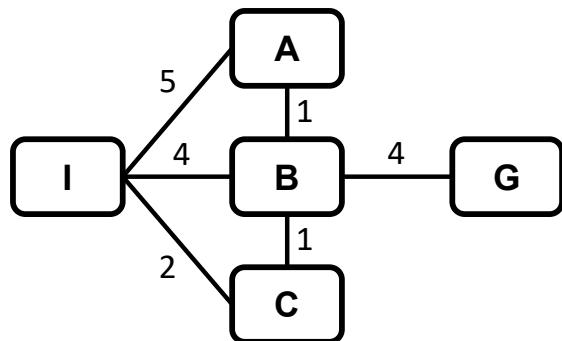
return failure

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

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**

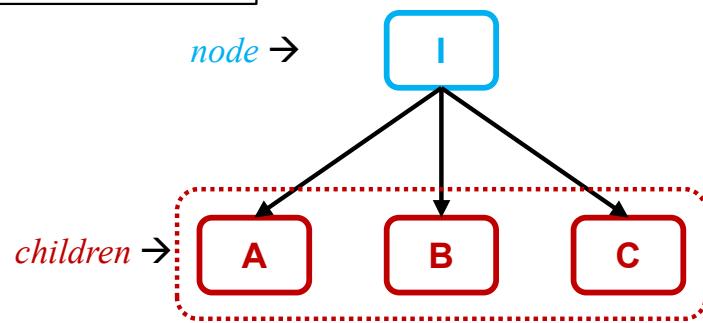
reached[s] \leftarrow *child*

 add *child* to *frontier*

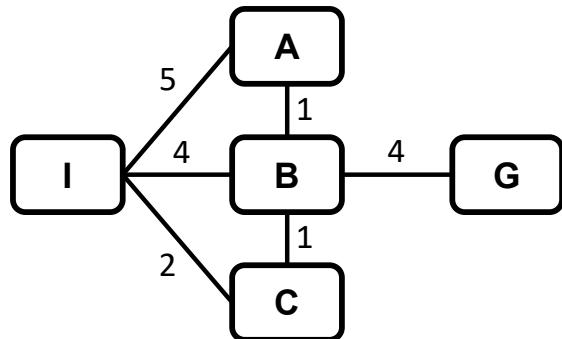
return failure

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

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*

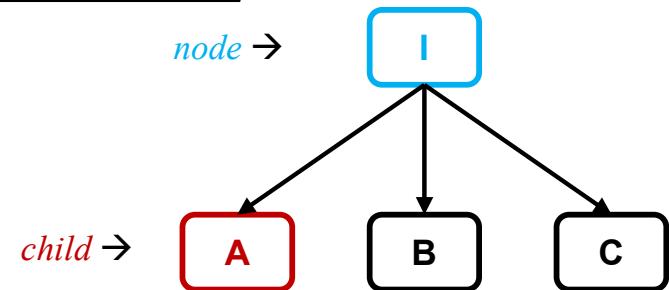
 add *child* to *frontier*

return failure

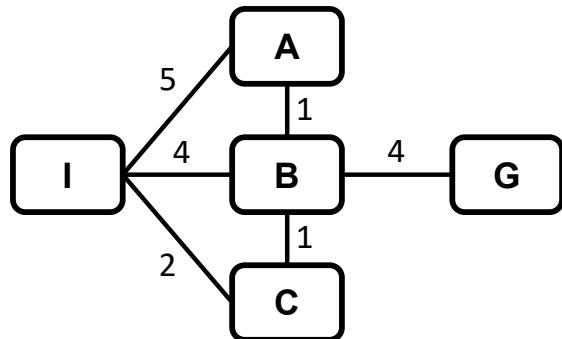
Frontier	Parent					
	Node					
	<i>f</i> (Node)					

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

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					
	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

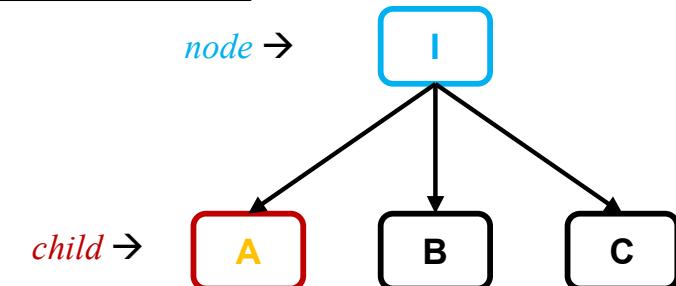
$s \leftarrow \text{child.STATE}$

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

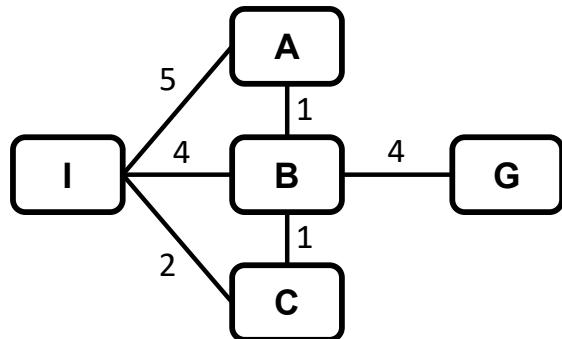
reached[s] \leftarrow *child*

 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 \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

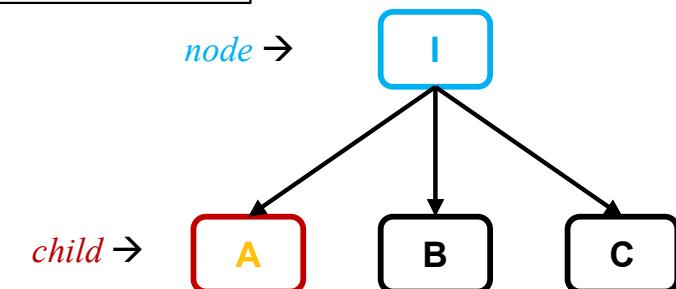
$s \leftarrow \text{child.STATE}$

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

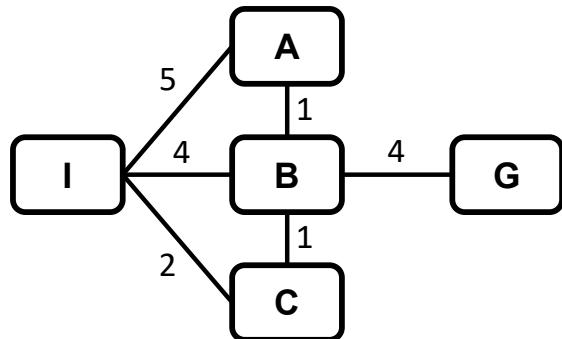
reached[s] \leftarrow *child*

 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

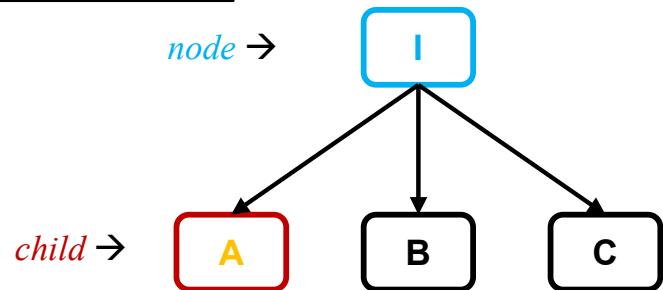
Frontier	Parent					
	Node					
	$f(\text{Node})$					

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

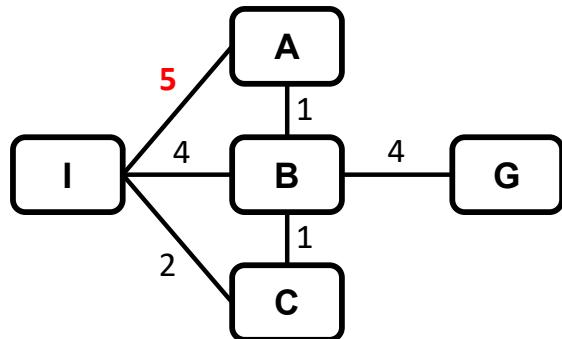
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

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

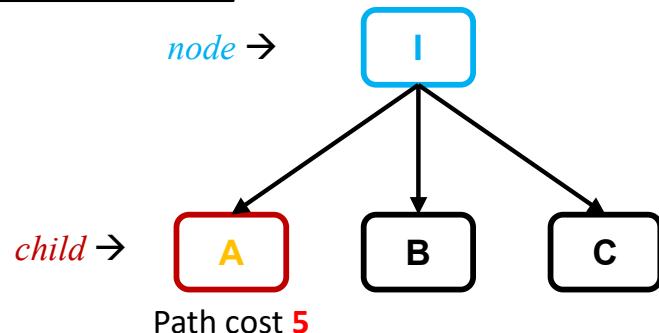
s \leftarrow *child*.STATE

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

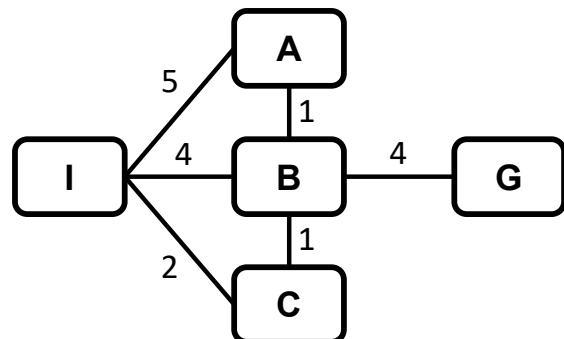
reached[*s*] \leftarrow *child*

 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*

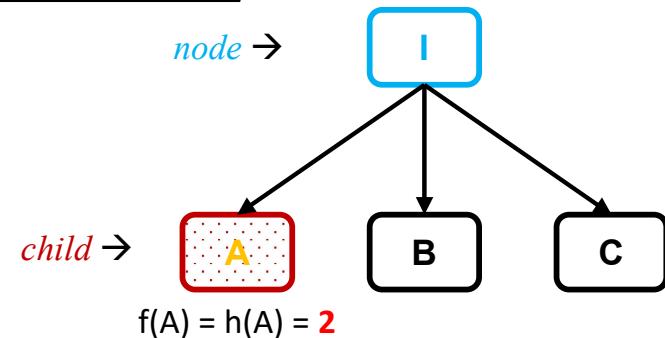
 add *child* to *frontier*

 return failure

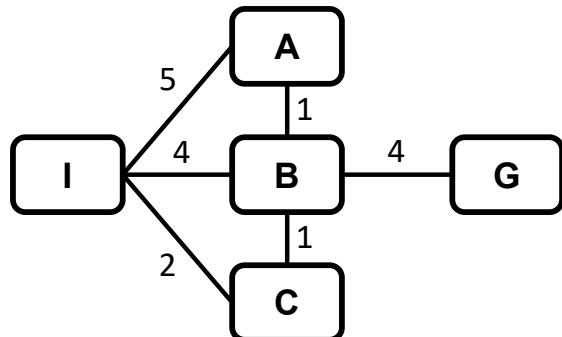
Frontier	Parent	I					
	Node	A					
	<i>f</i> (Node)	2					

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

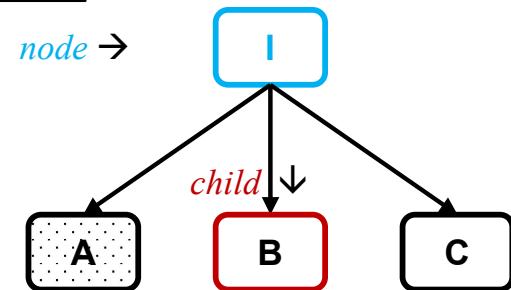
s \leftarrow *child*.STATE

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

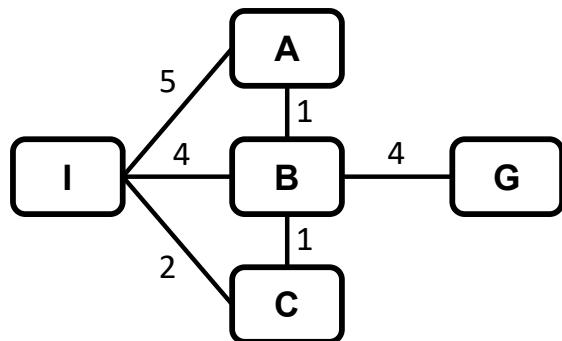
reached[*s*] \leftarrow *child*

 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 \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

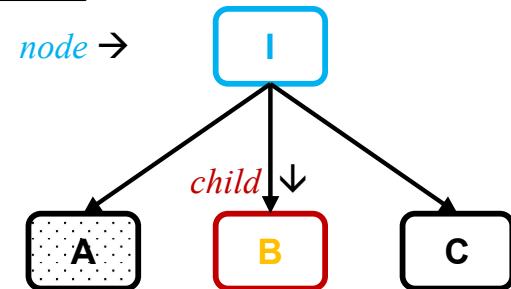
s \leftarrow *child*.STATE

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

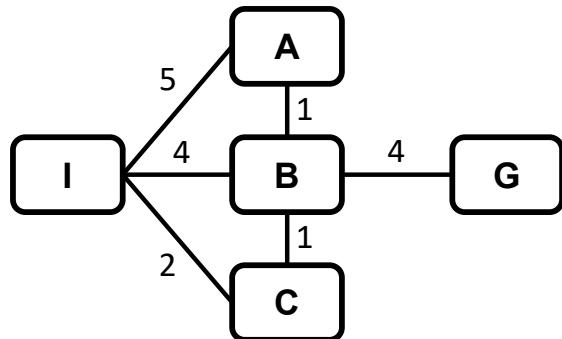
reached[*s*] \leftarrow *child*

 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 \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

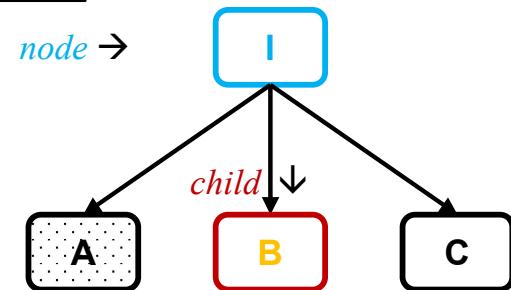
s \leftarrow *child*.STATE

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

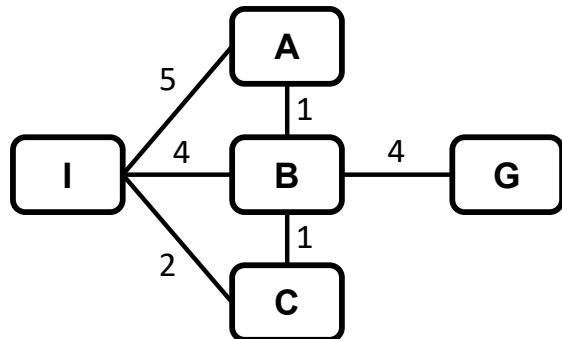
reached[*s*] \leftarrow *child*

 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

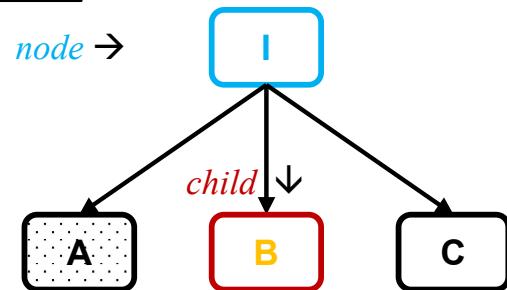
Frontier	Parent	I					
	Node	A					
	f(Node)	2					

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

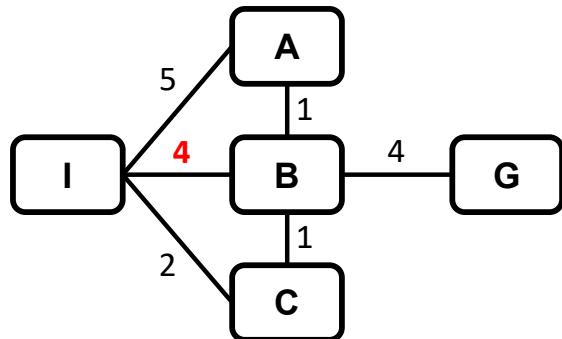
State Space Graph	Frontier / Reached
Algorithm	Search Tree

```

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

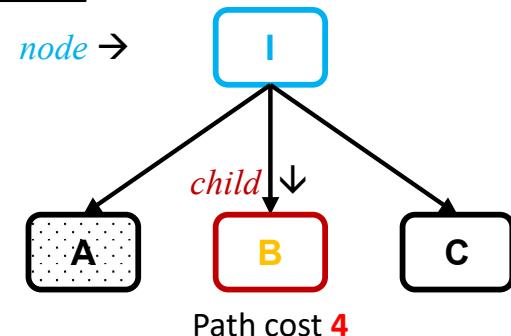
s \leftarrow *child*.STATE

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

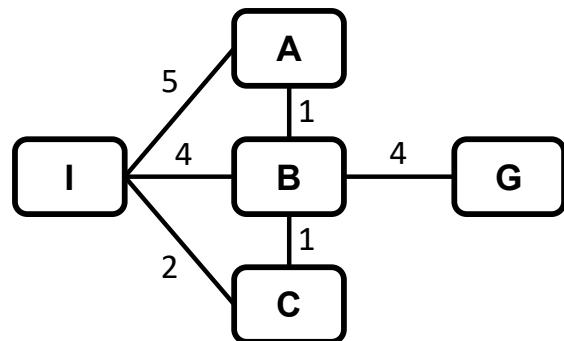
reached[*s*] \leftarrow *child*

 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

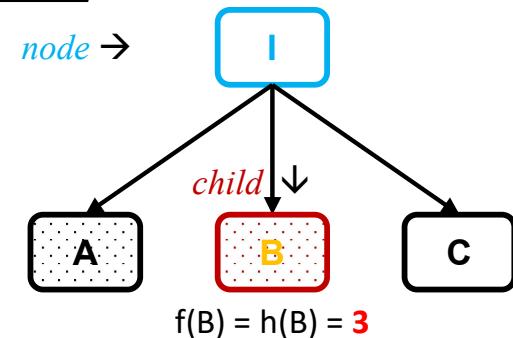
s \leftarrow *child*.STATE

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

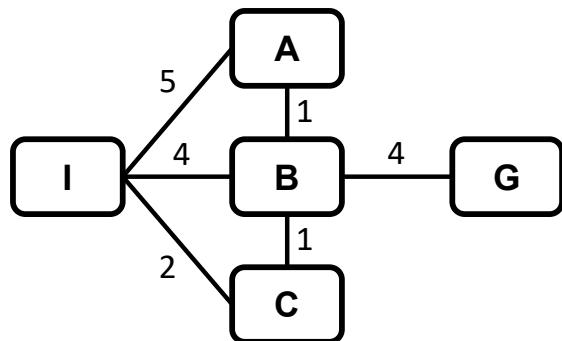
reached[*s*] \leftarrow *child*

 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*

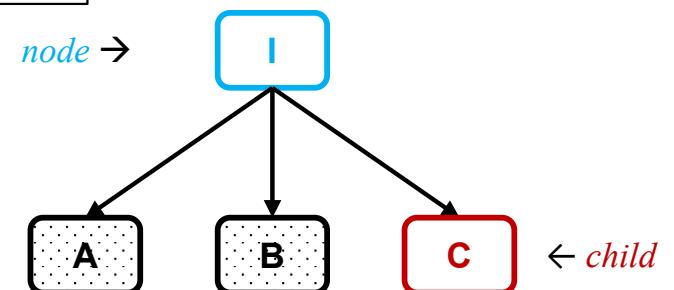
 add *child* to *frontier*

return failure

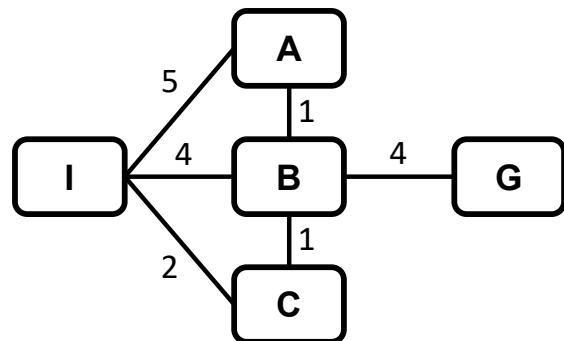
State Space Graph	Frontier / Reached
Algorithm	Search Tree

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			



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

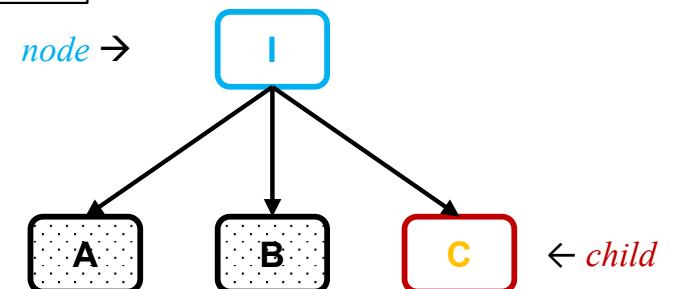
s \leftarrow *child*.STATE

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

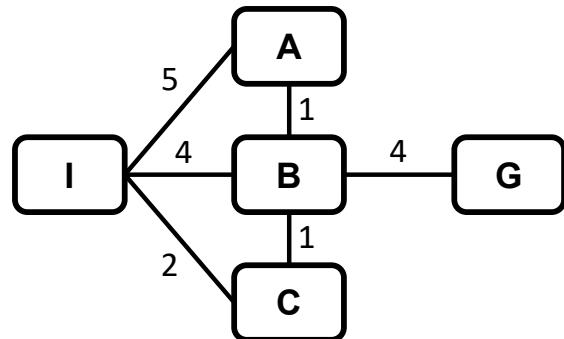
reached[*s*] \leftarrow *child*

 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 \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

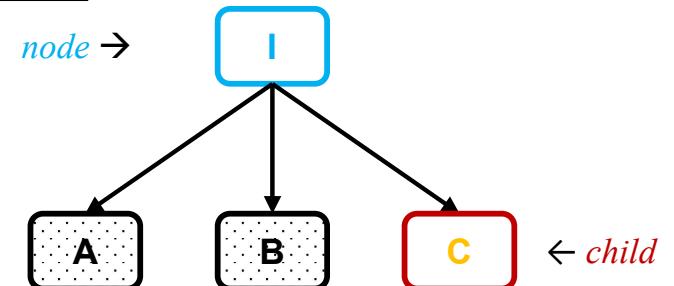
s \leftarrow *child*.STATE

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

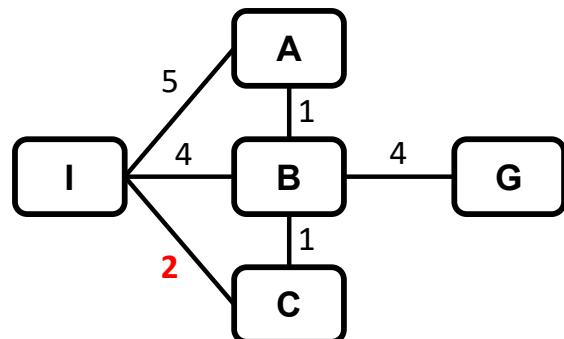
reached[*s*] \leftarrow *child*

 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	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

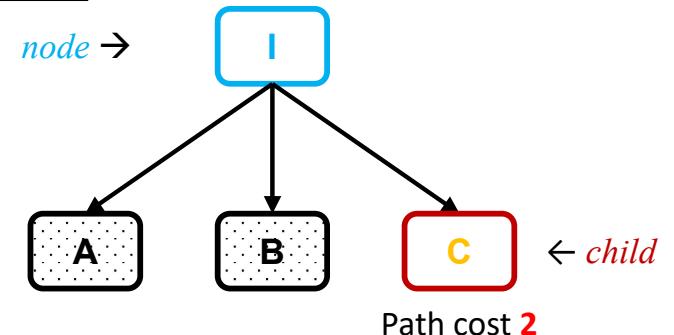
s \leftarrow *child*.STATE

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

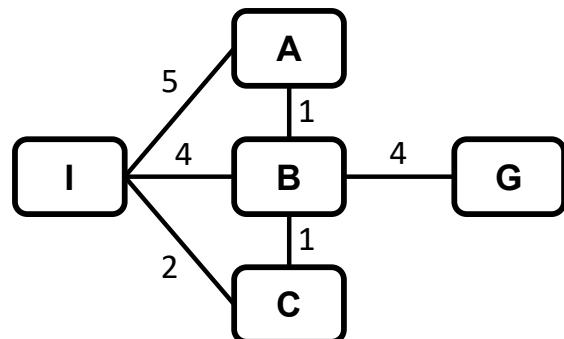
reached[*s*] \leftarrow *child*

 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

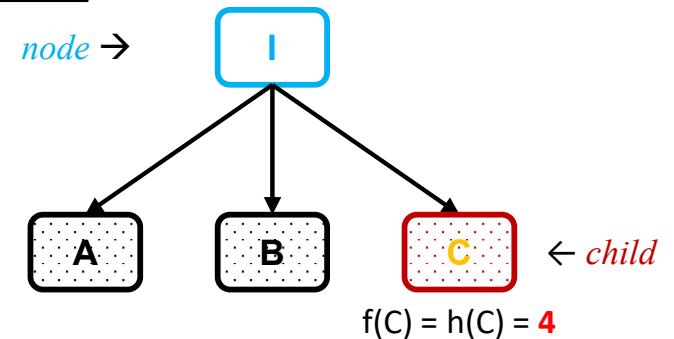
Frontier	Parent	I	I	I		
	Node	A	B	C		
	$f(\text{Node})$	2	3	4		

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

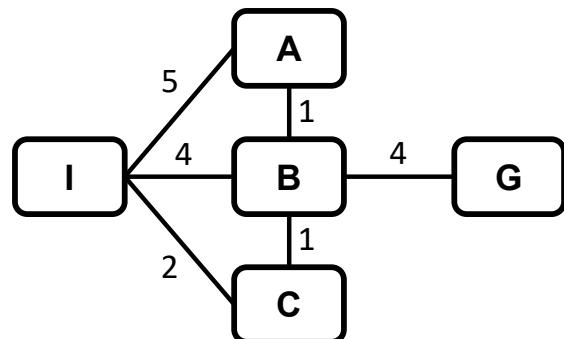
```

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

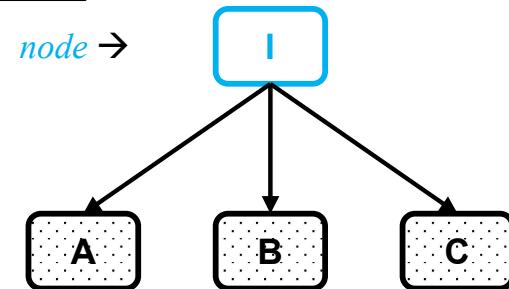
s \leftarrow *child.STATE*

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

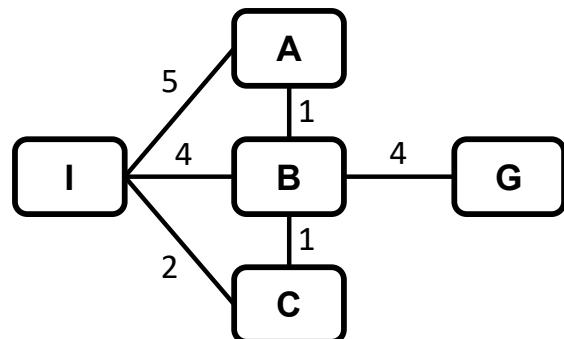
reached[s] \leftarrow *child*

 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**

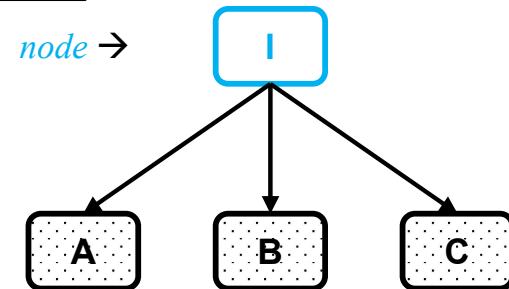
s \leftarrow *child.STATE*

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

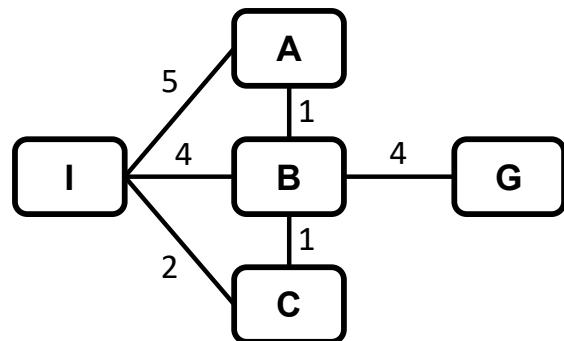
reached[s] \leftarrow *child*

 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**

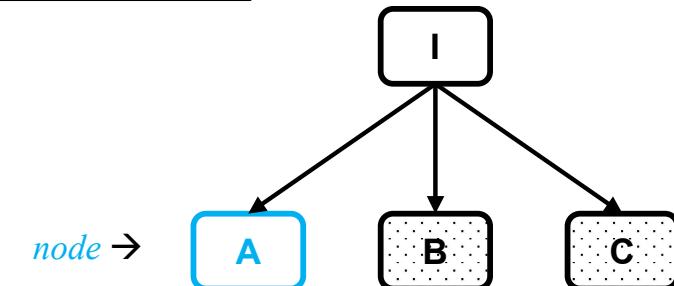
s \leftarrow *child*.STATE

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

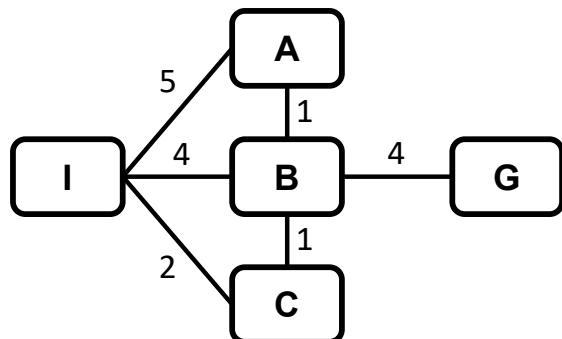
reached[*s*] \leftarrow *child*

 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

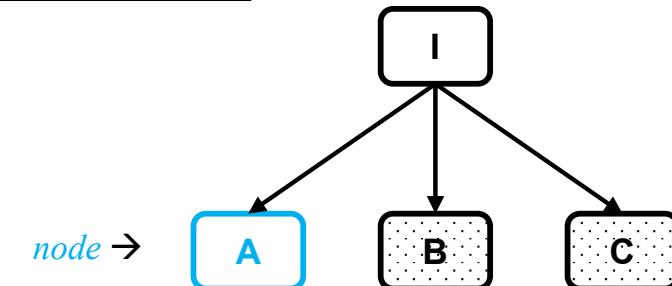
s \leftarrow *child*.STATE

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

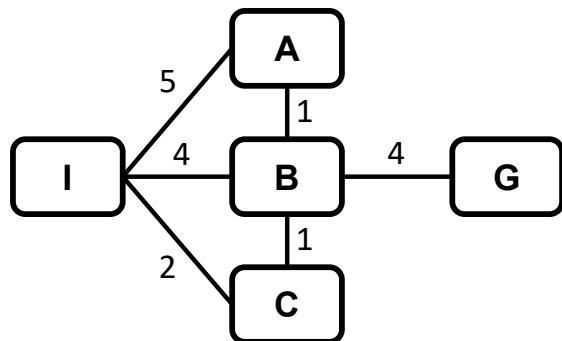
reached[*s*] \leftarrow *child*

 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**

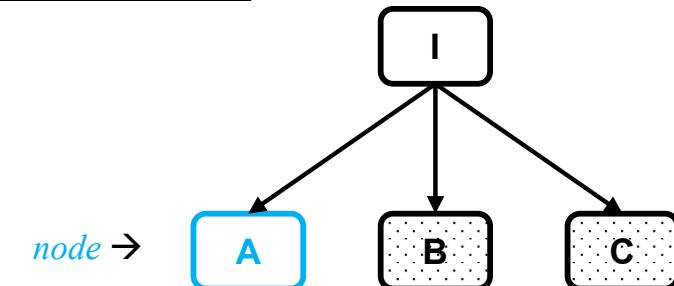
s \leftarrow *child.STATE*

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

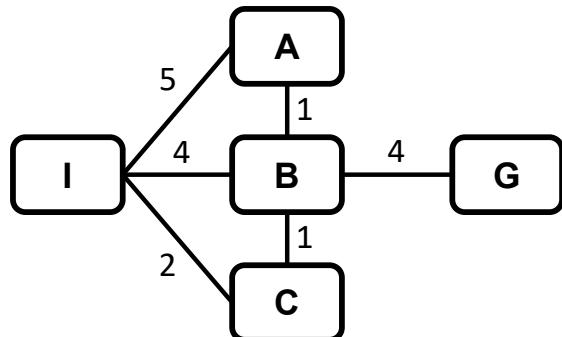
reached[s] \leftarrow *child*

 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

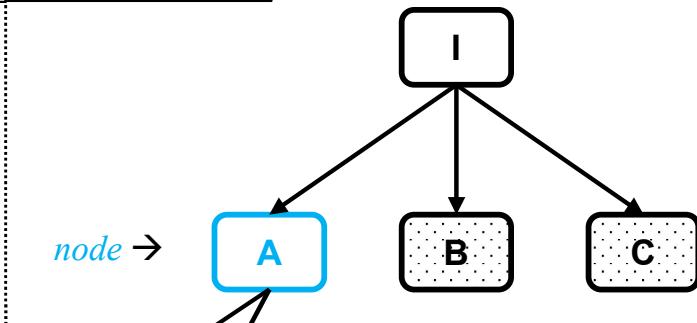
s \leftarrow *child*.STATE

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

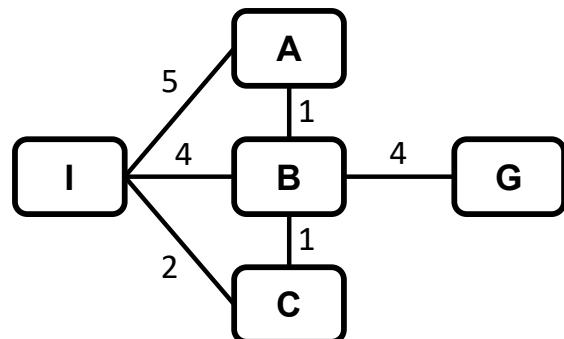
reached[*s*] \leftarrow *child*

 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

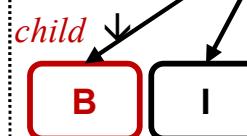
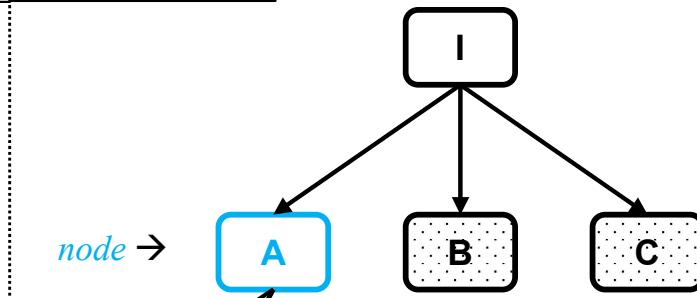
s \leftarrow *child*.STATE

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

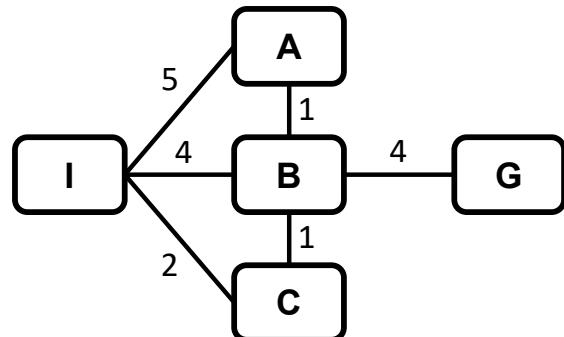
reached[*s*] \leftarrow *child*

 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					
$f(\text{Node})$	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

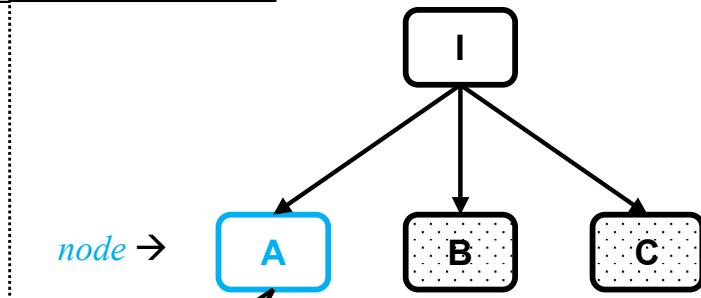
s \leftarrow *child*.STATE

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

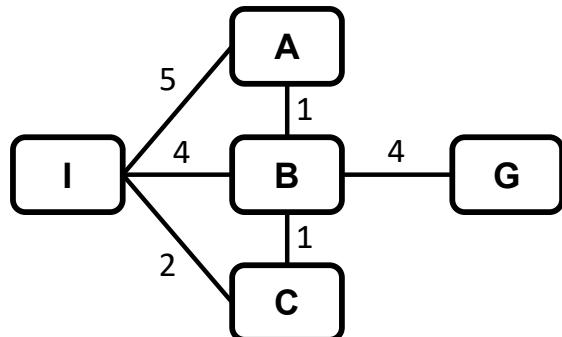
reached[*s*] \leftarrow *child*

 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

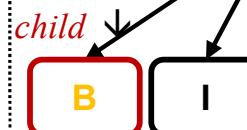
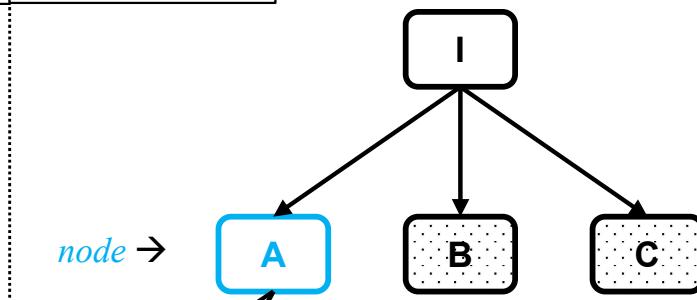
s \leftarrow *child*.STATE

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

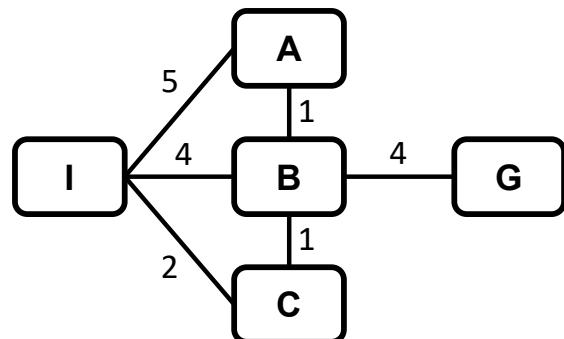
reached[*s*] \leftarrow *child*

 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

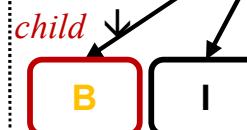
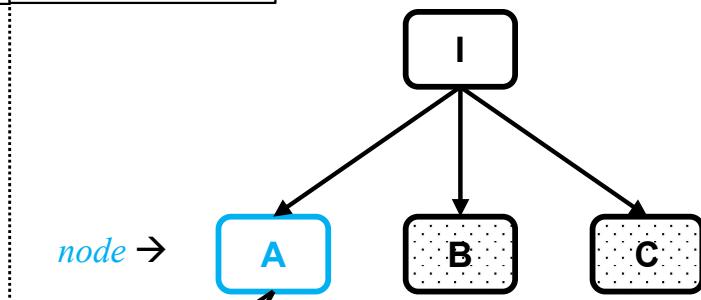
s \leftarrow *child*.STATE

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

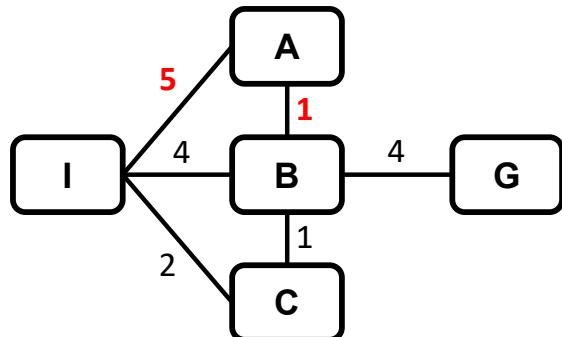
reached[*s*] \leftarrow *child*

 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

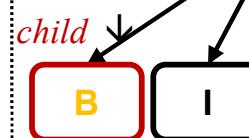
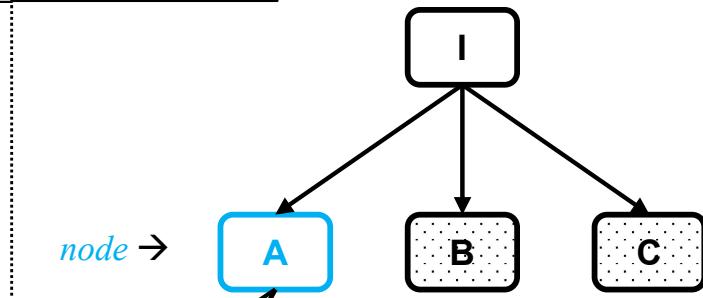
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

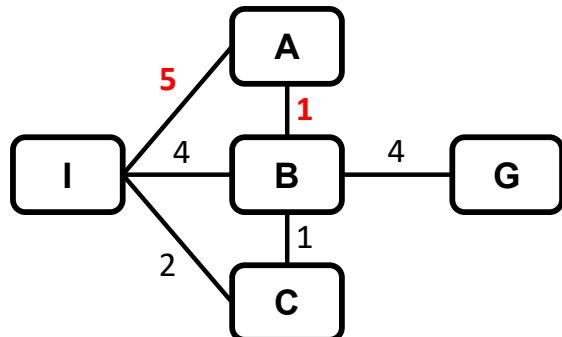
 return failure



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

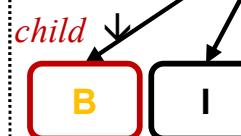
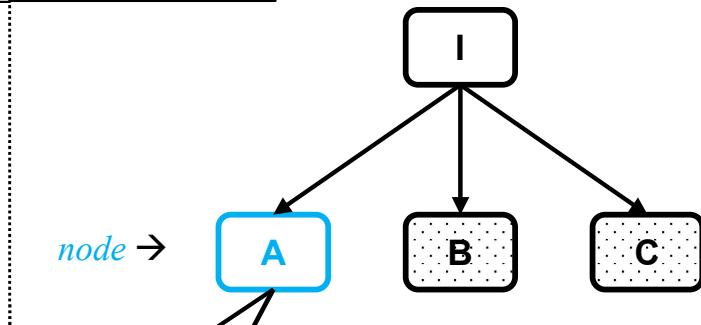
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

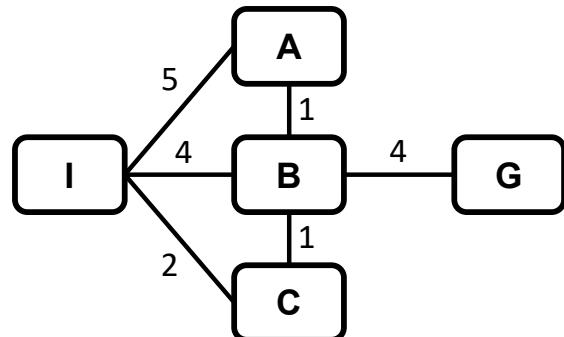
 return failure



Path cost
 $6 > 4$



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

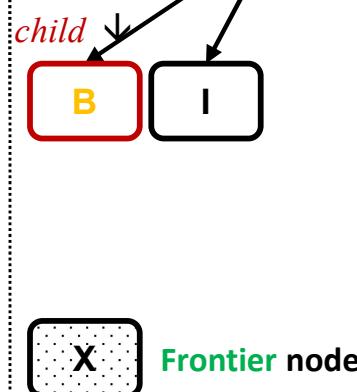
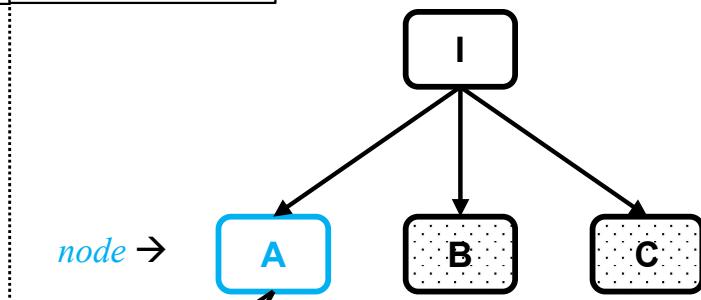
s \leftarrow *child*.STATE

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

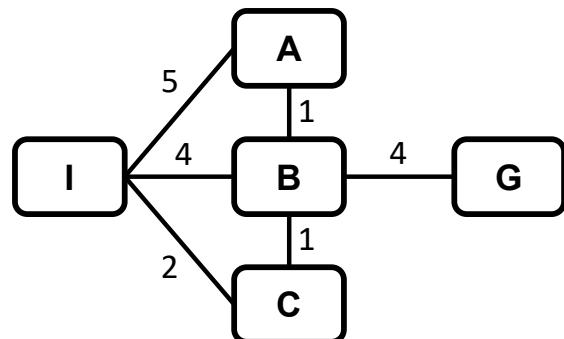
reached[*s*] \leftarrow *child*

 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

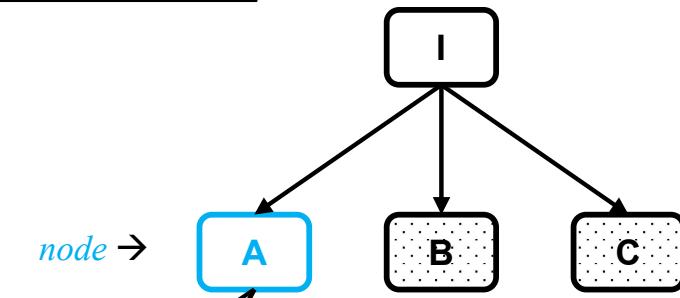
s \leftarrow *child*.STATE

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

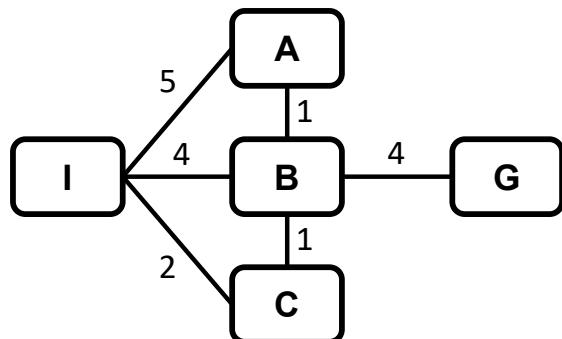
reached[*s*] \leftarrow *child*

 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

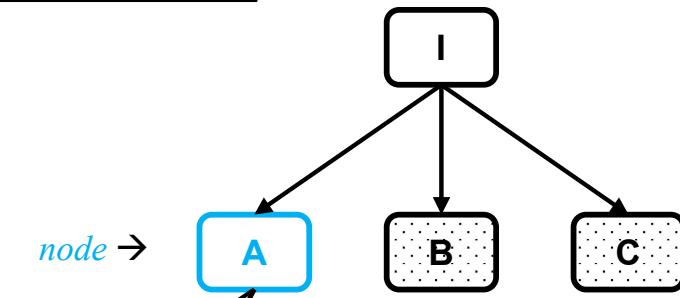
s \leftarrow *child*.STATE

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

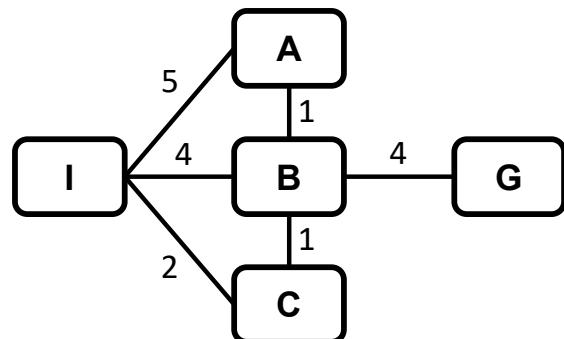
reached[*s*] \leftarrow *child*

 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

node \leftarrow POP(*frontier*)

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

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

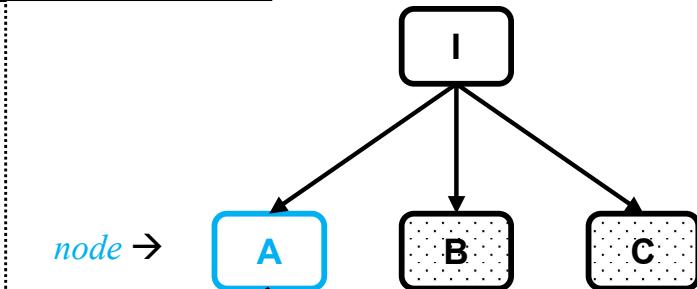
s \leftarrow *child*.STATE

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

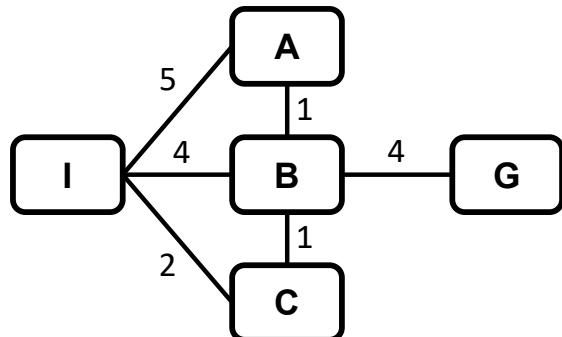
reached[*s*] \leftarrow *child*

 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

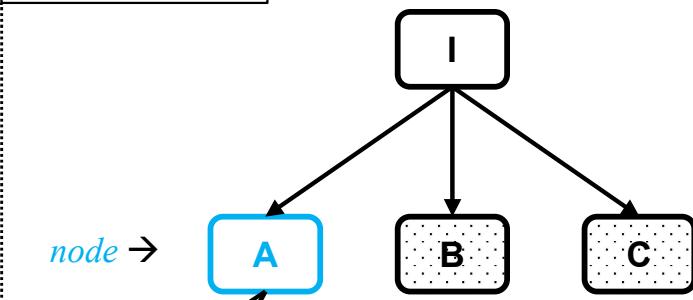
s \leftarrow *child*.STATE

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

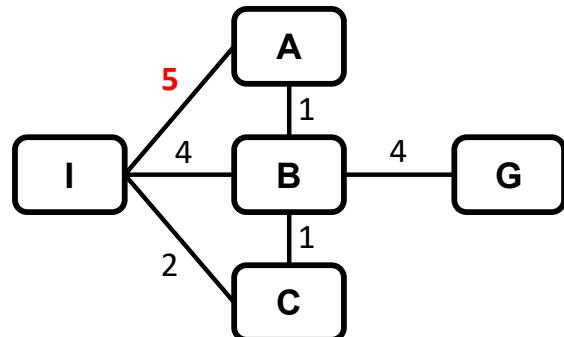
reached[*s*] \leftarrow *child*

 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(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

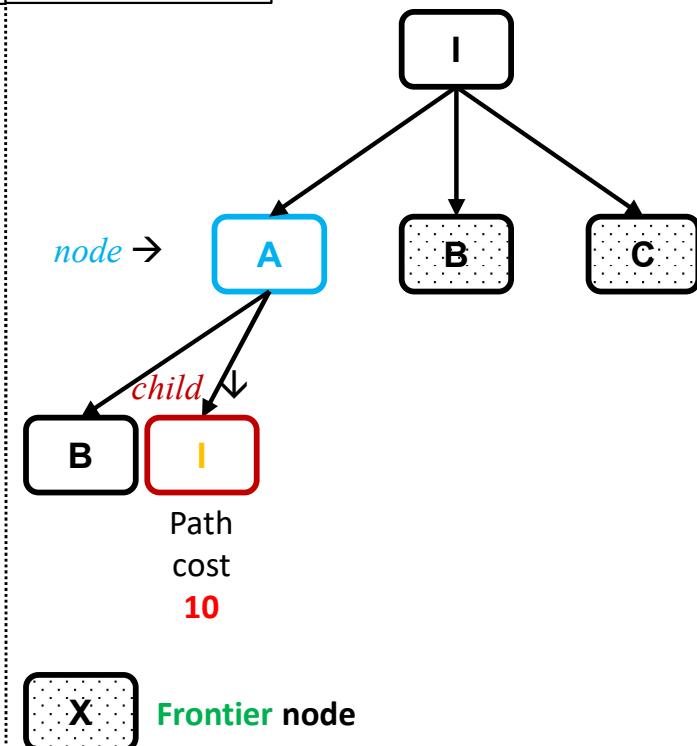
s \leftarrow *child*.STATE

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

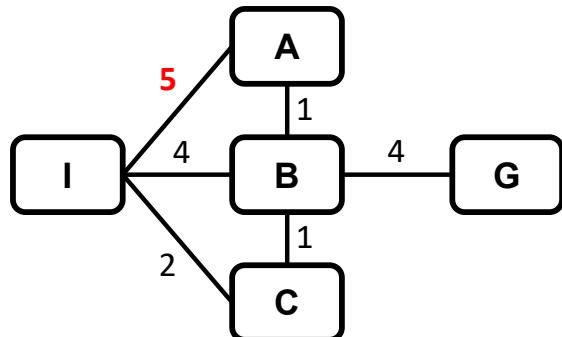
reached[*s*] \leftarrow *child*

 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

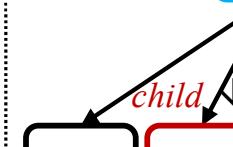
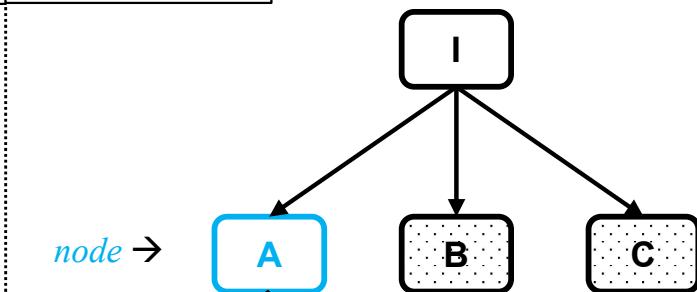
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

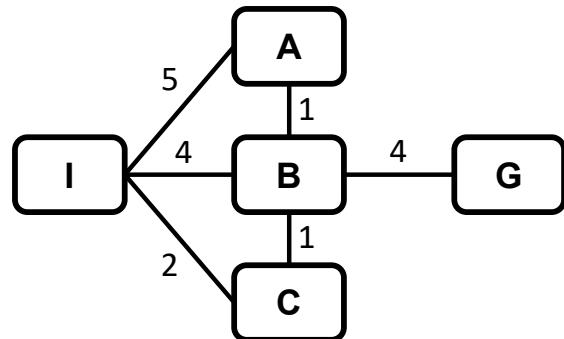
 return failure



Path
cost
10>0

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(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

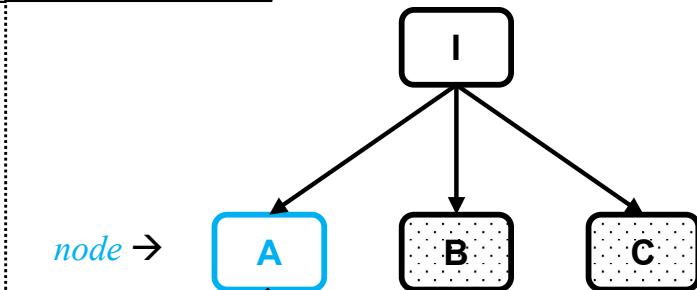
s \leftarrow *child*.STATE

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

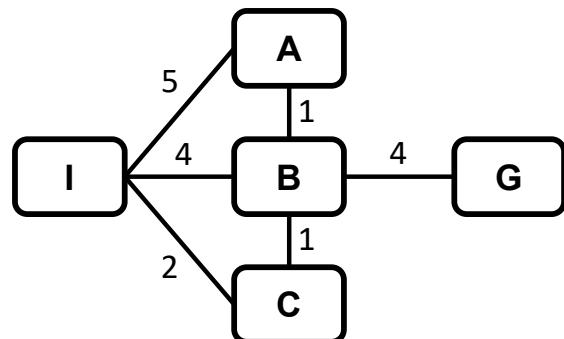
reached[*s*] \leftarrow *child*

 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

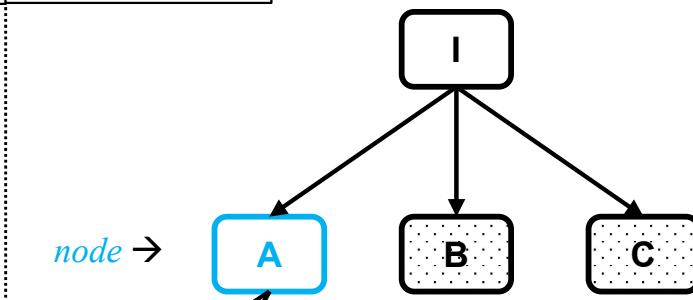
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

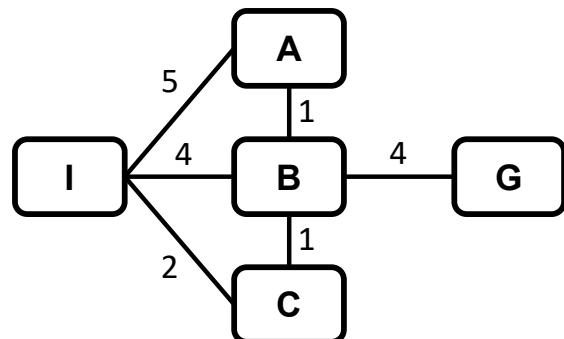
 add *child* to *frontier*

 return failure



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(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

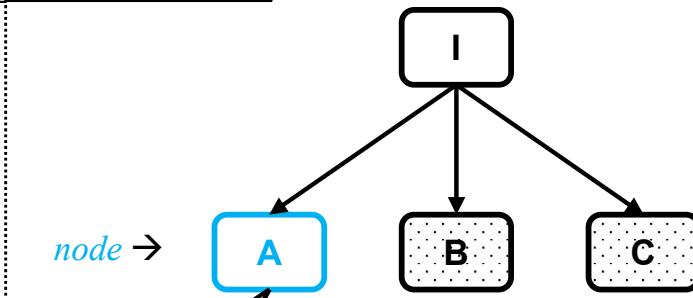
s \leftarrow *child*.STATE

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

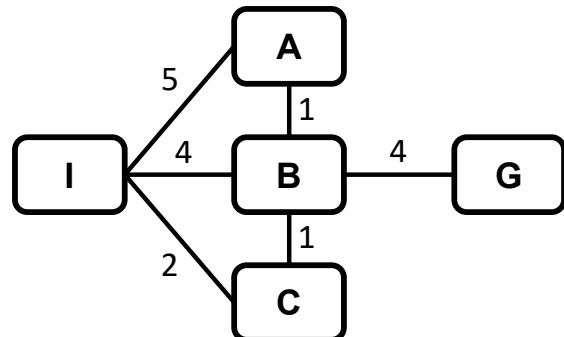
reached[*s*] \leftarrow *child*

 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

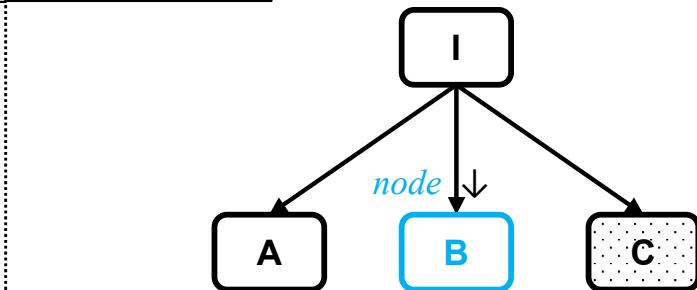
s \leftarrow *child*.STATE

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

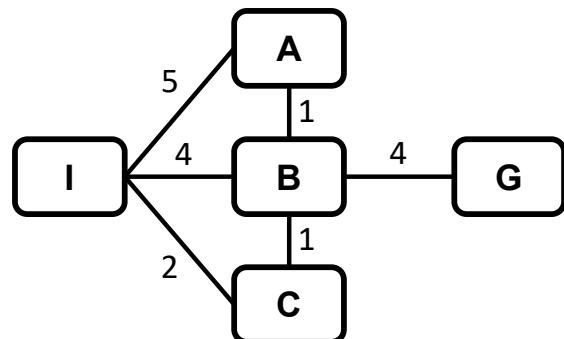
reached[*s*] \leftarrow *child*

 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

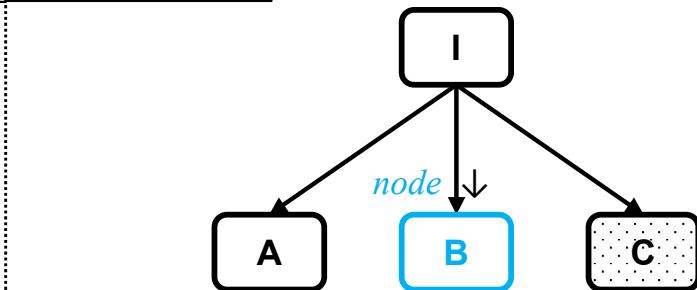
s \leftarrow *child*.STATE

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

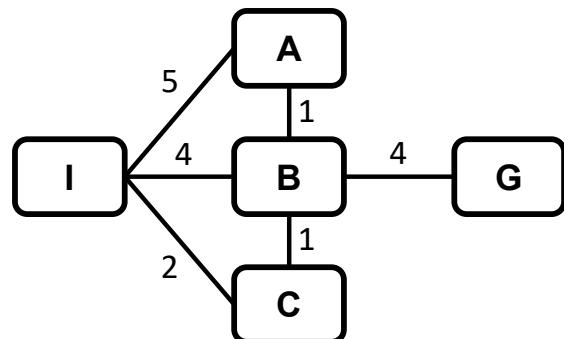
reached[*s*] \leftarrow *child*

 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

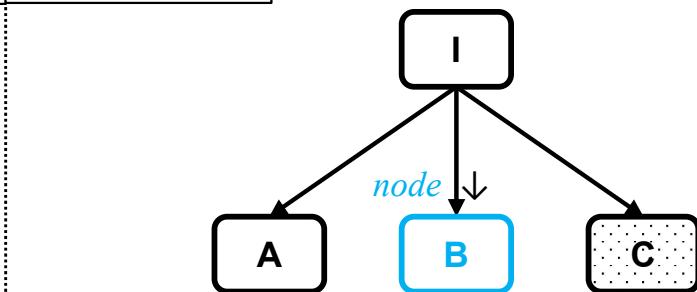
s \leftarrow *child*.STATE

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

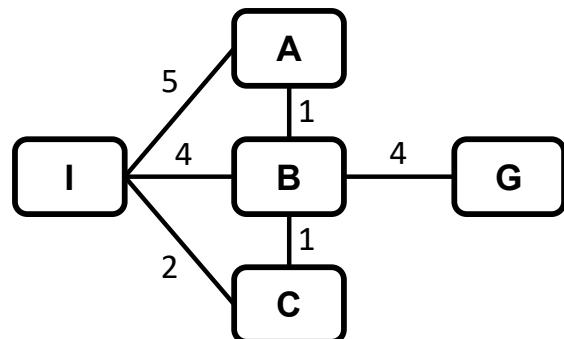
reached[*s*] \leftarrow *child*

 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

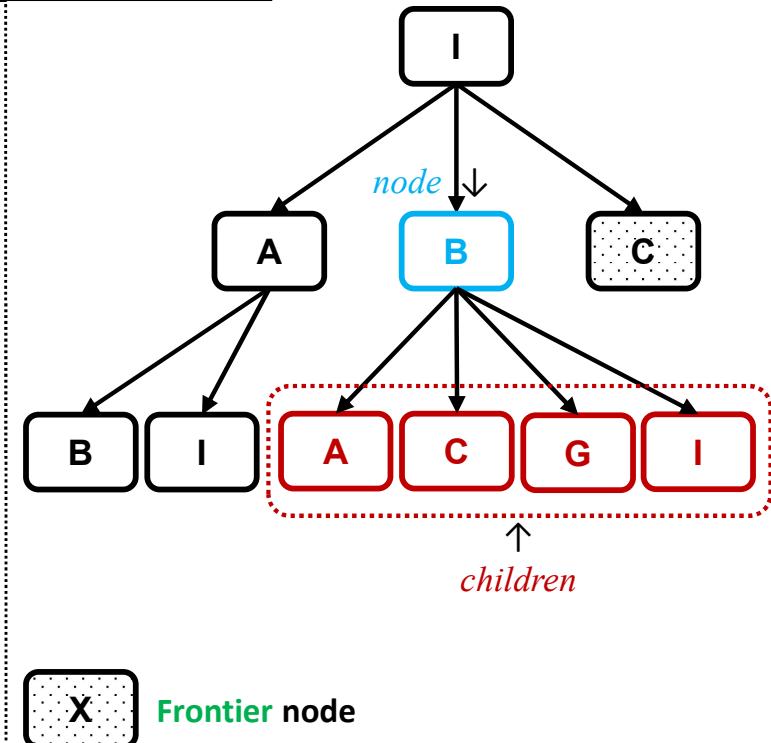
s \leftarrow *child*.STATE

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

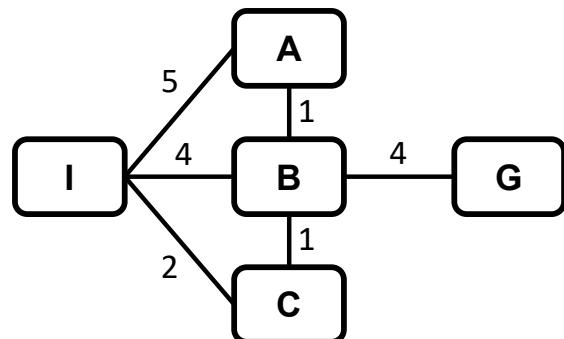
reached[*s*] \leftarrow *child*

 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

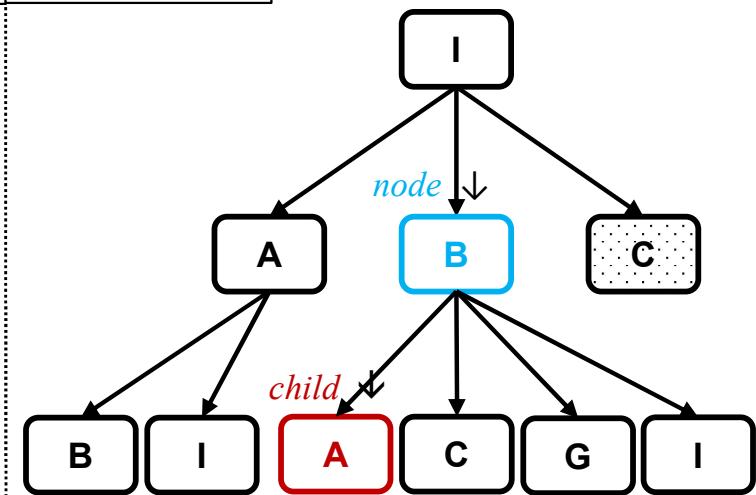
s \leftarrow *child*.STATE

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

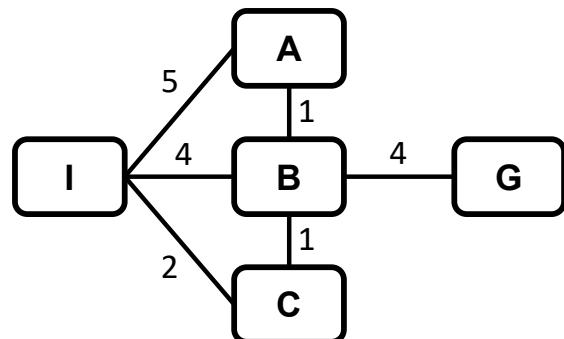
reached[*s*] \leftarrow *child*

 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

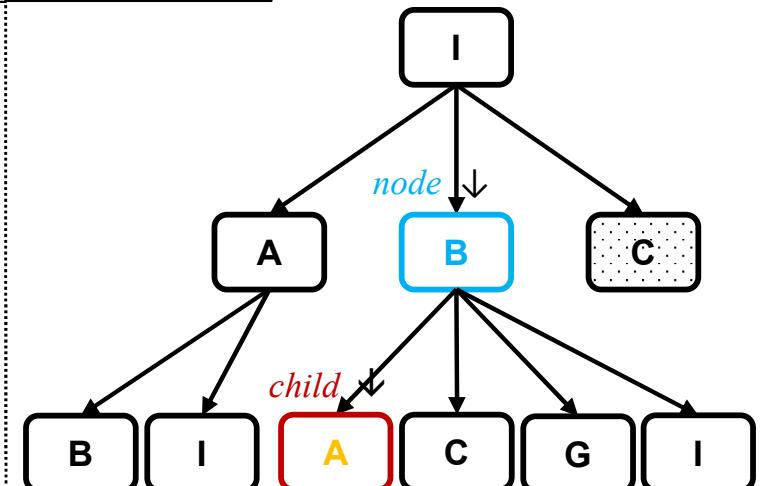
s \leftarrow *child*.STATE

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

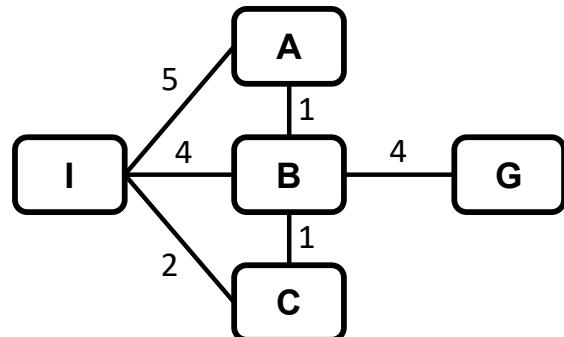
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



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

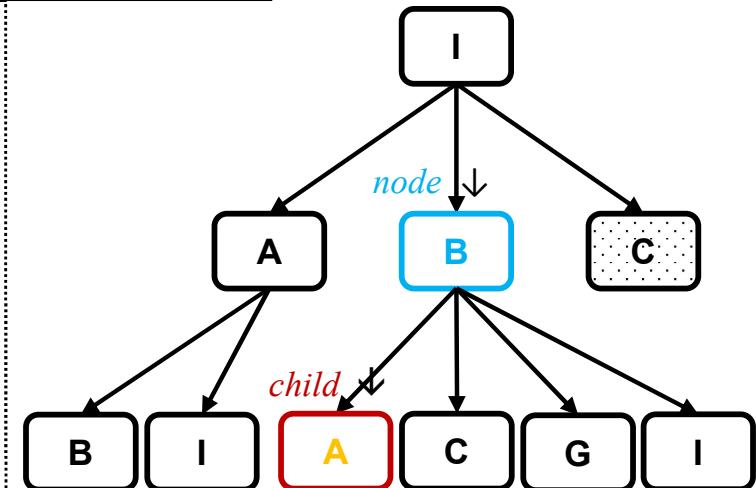
s \leftarrow *child*.STATE

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

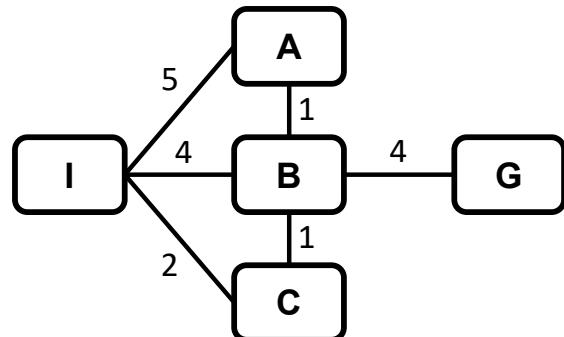
reached[*s*] \leftarrow *child*

 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

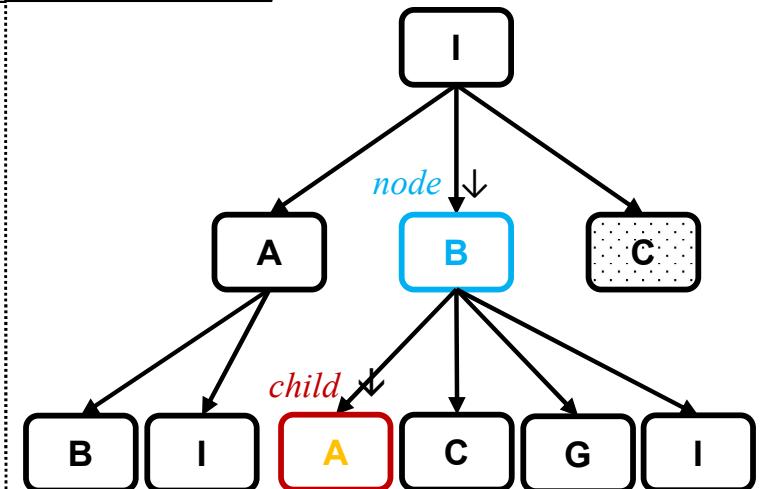
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

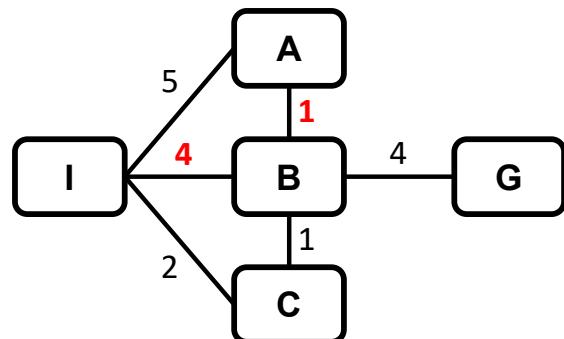
 add *child* to *frontier*

 return failure



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
```

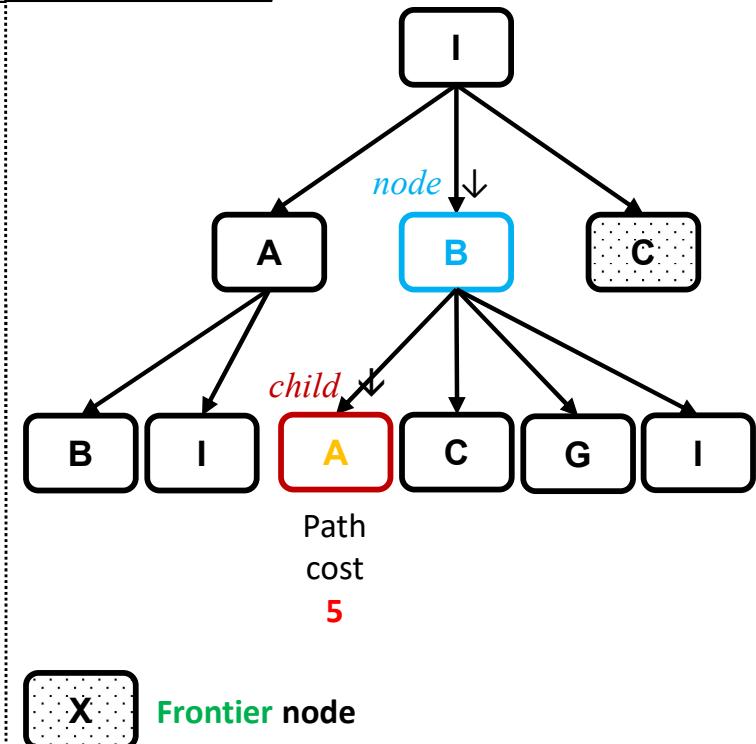
```
            s ← child.STATE
```

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

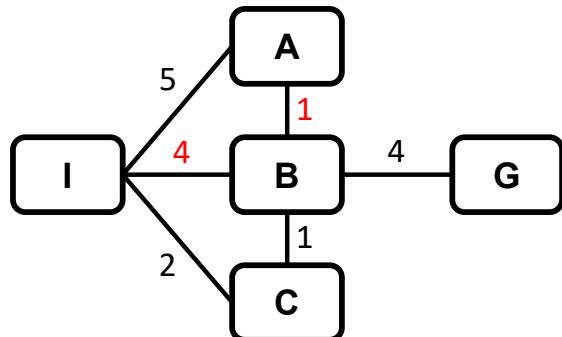
```
                reached[s] ← child
```

```
                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

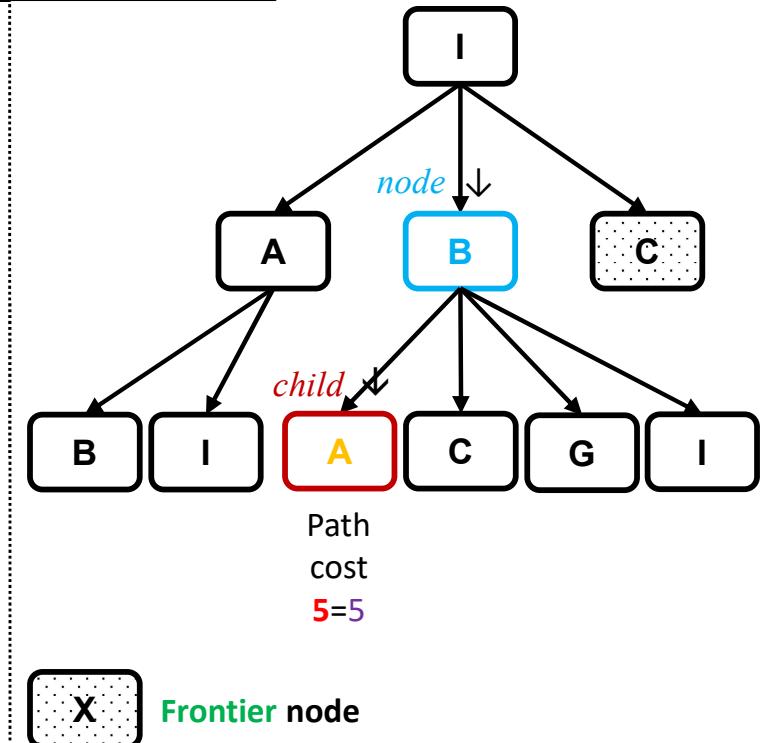
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

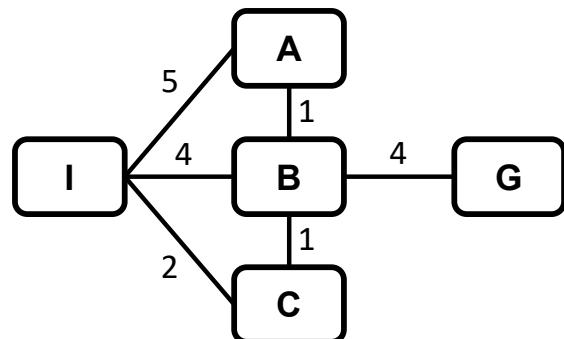
 add *child* to *frontier*

 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

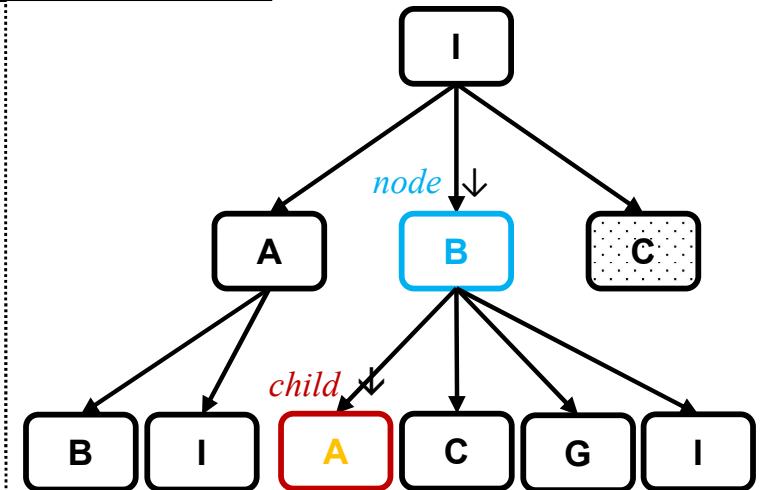
s \leftarrow *child*.STATE

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

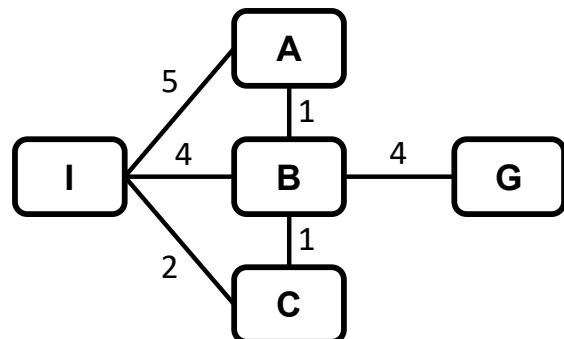
reached[*s*] \leftarrow *child*

 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 \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

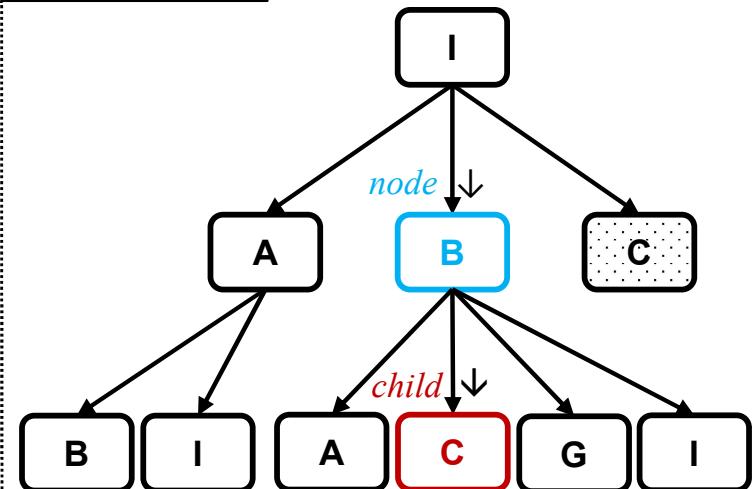
s \leftarrow *child*.STATE

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

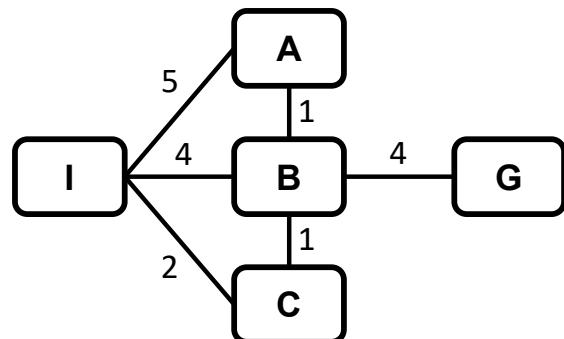
reached[*s*] \leftarrow *child*

 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

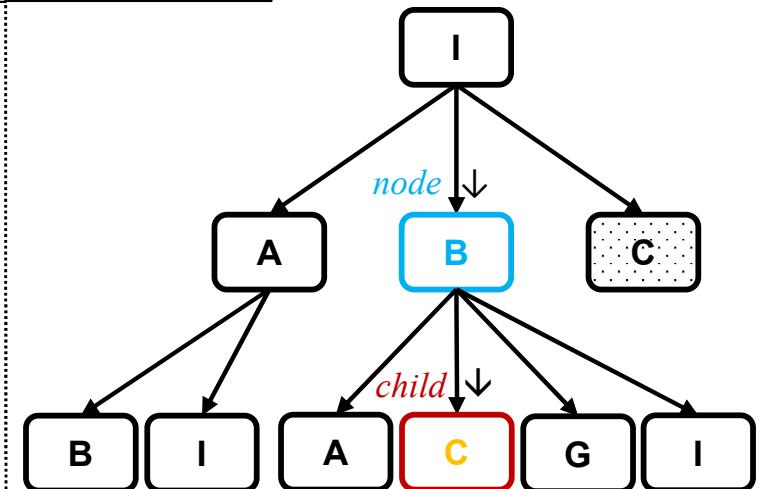
s \leftarrow *child*.STATE

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

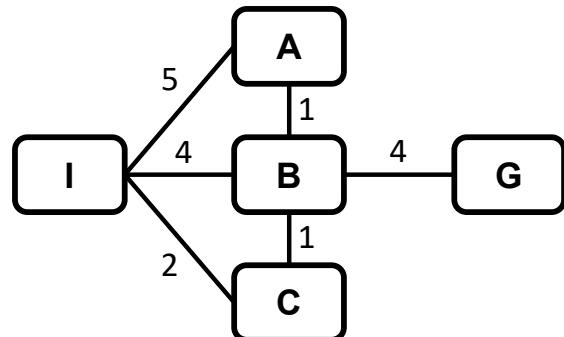
reached[*s*] \leftarrow *child*

 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

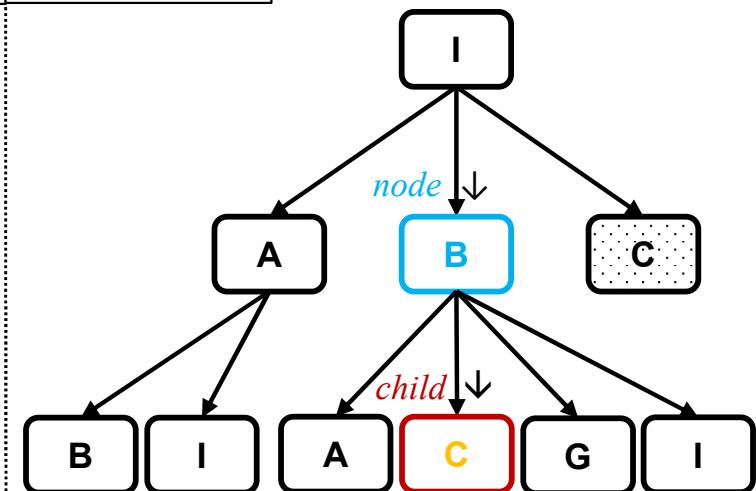
s \leftarrow *child*.STATE

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

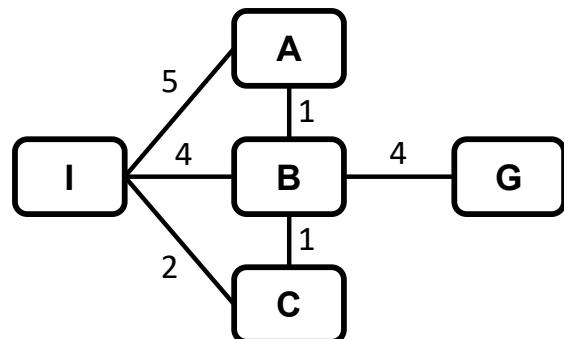
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



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

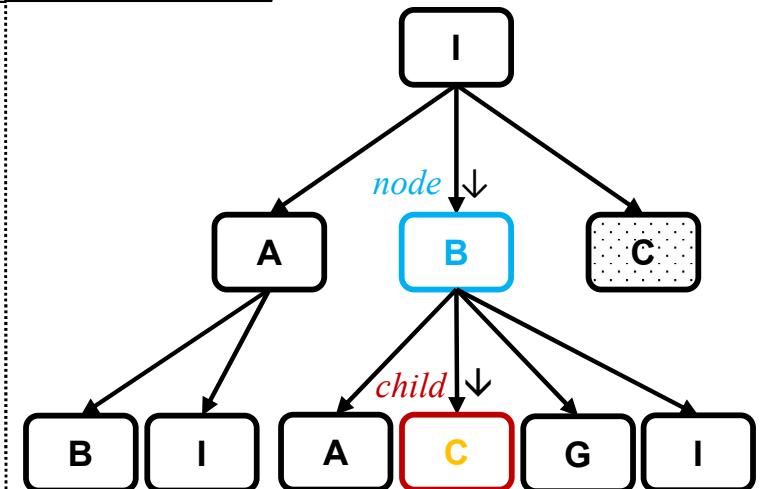
s \leftarrow *child*.STATE

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

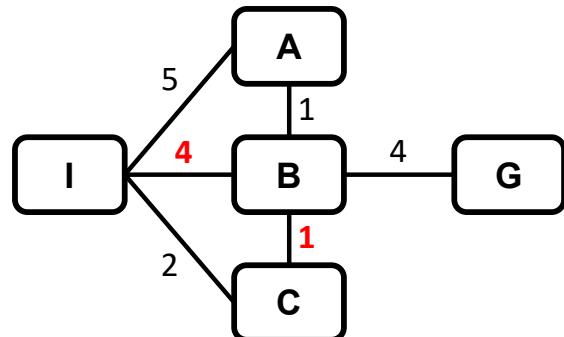
reached[*s*] \leftarrow *child*

 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 \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

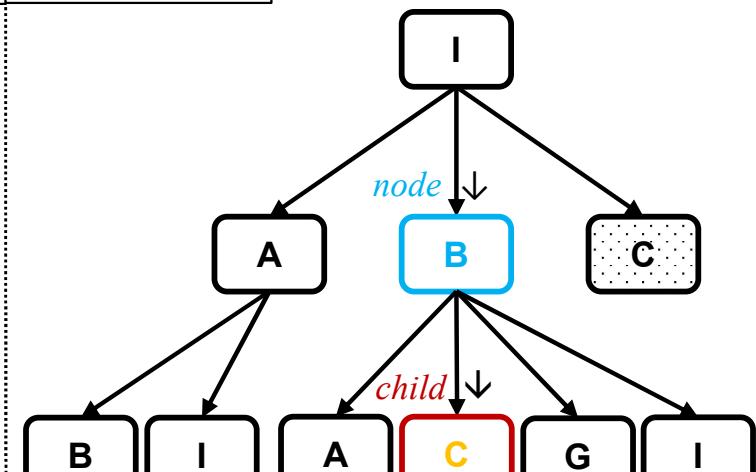
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

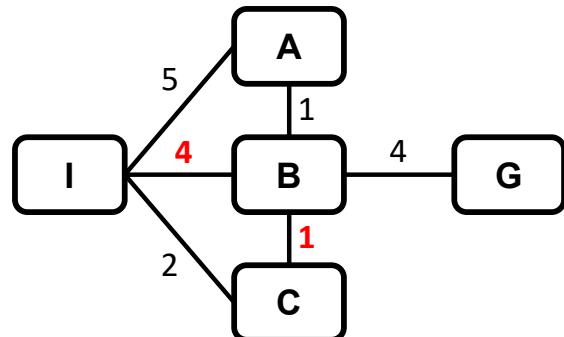
 return failure



Path cost
5

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

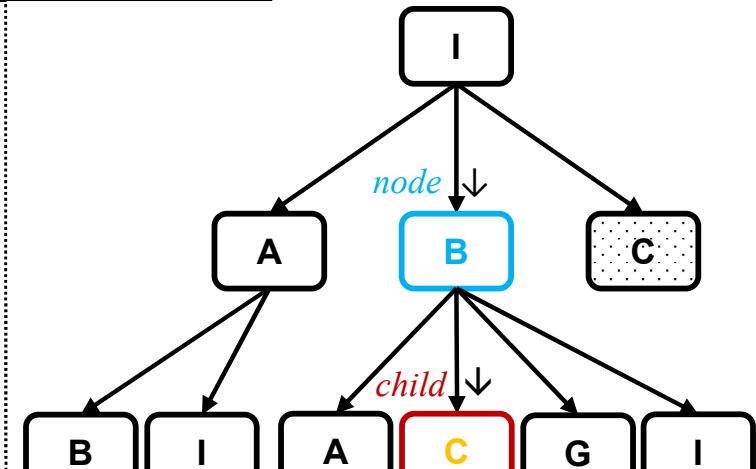
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

 add *child* to *frontier*

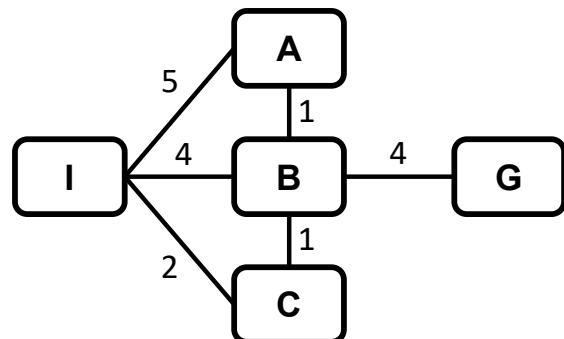
 return failure



X Frontier node

Path cost
5>2

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

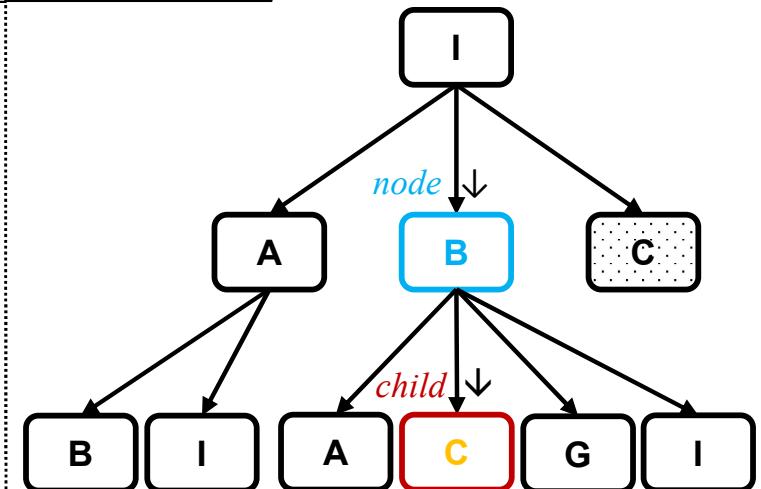
s \leftarrow *child*.STATE

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

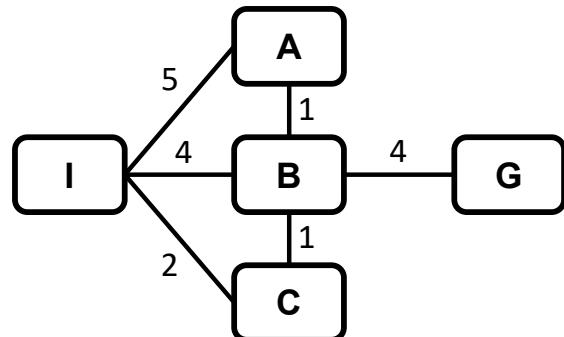
reached[*s*] \leftarrow *child*

 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

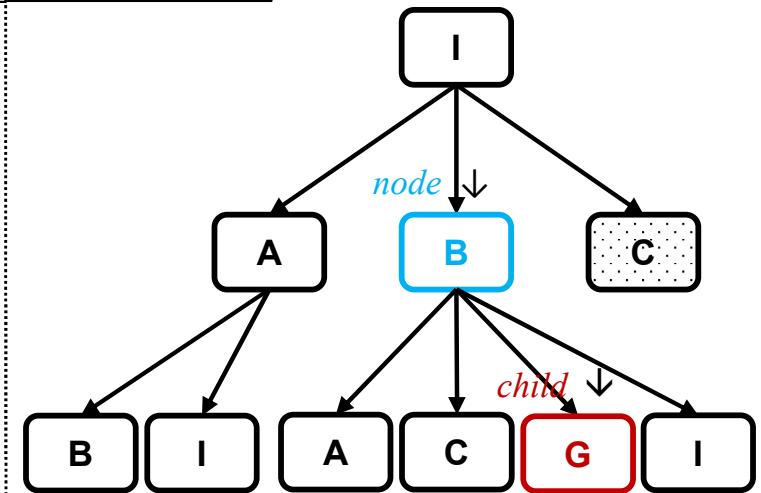
s \leftarrow *child*.STATE

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

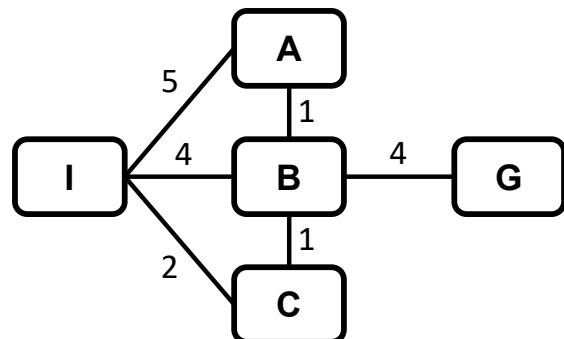
reached[*s*] \leftarrow *child*

 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

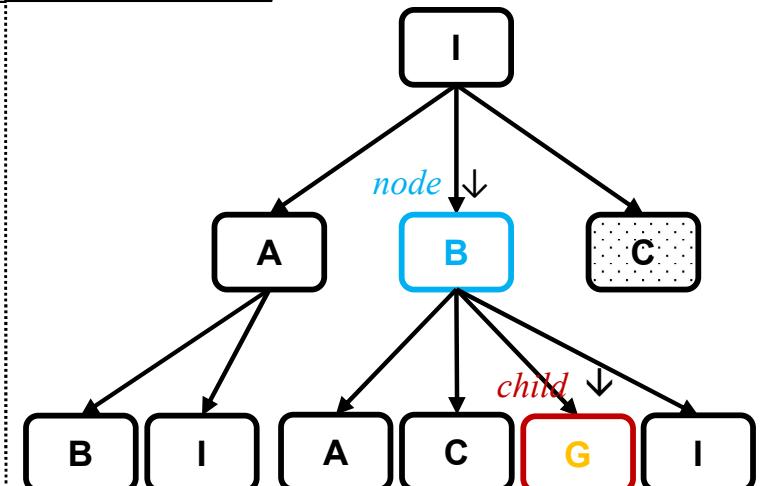
s \leftarrow *child*.STATE

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

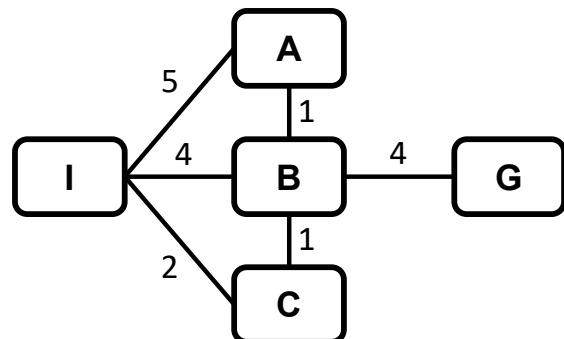
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



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

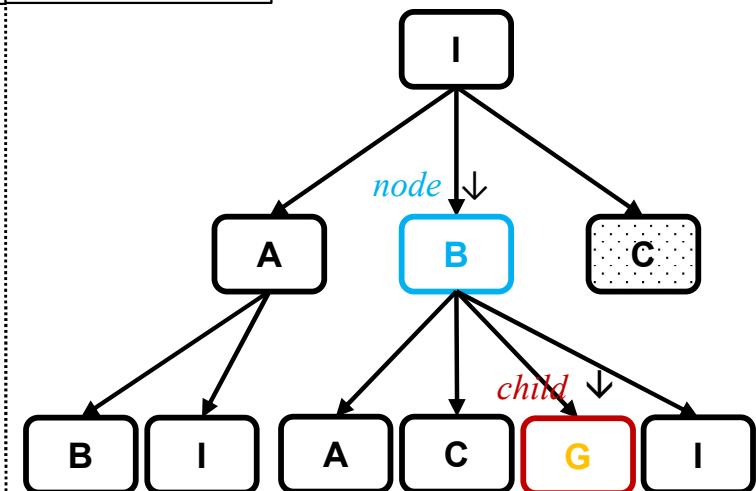
s \leftarrow *child*.STATE

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

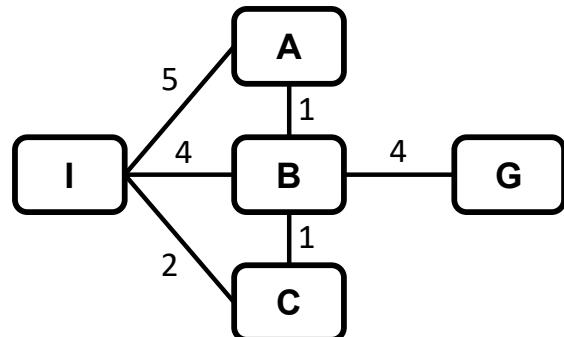
reached[*s*] \leftarrow *child*

 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

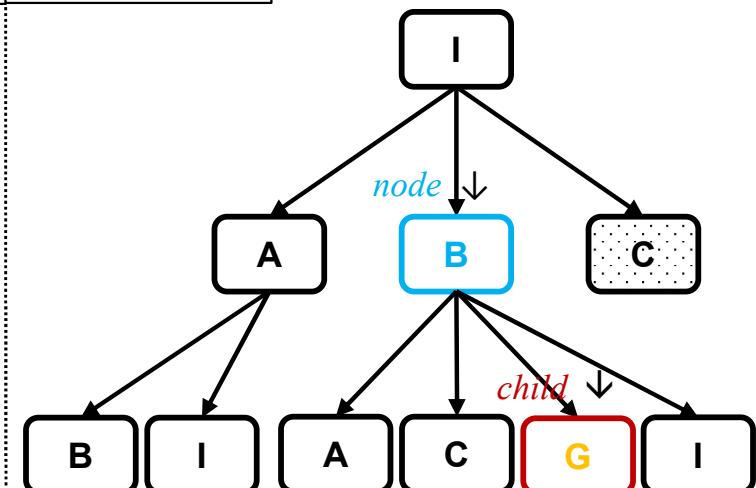
s \leftarrow *child*.STATE

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

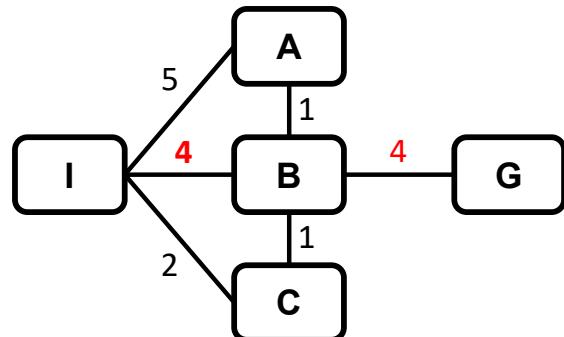
reached[*s*] \leftarrow *child*

 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	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

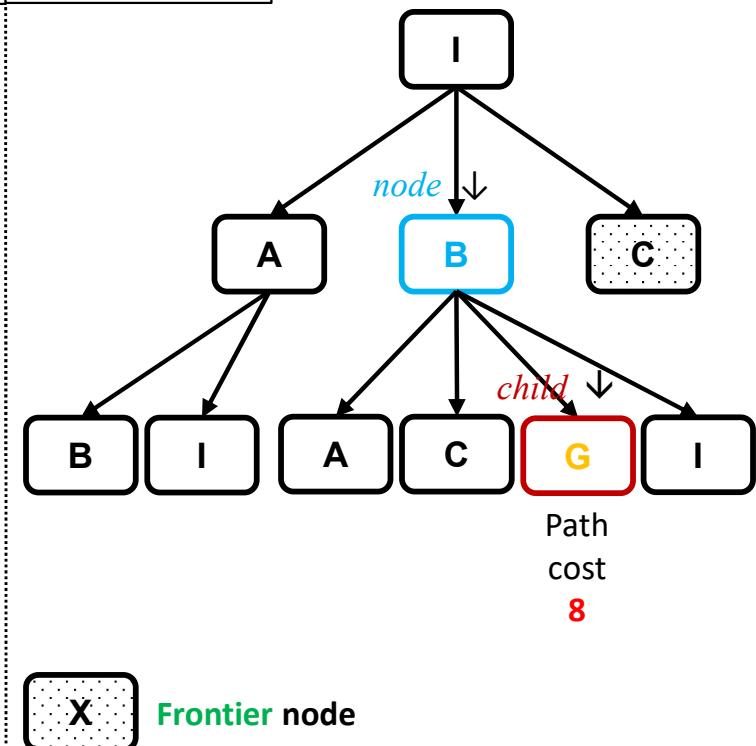
s \leftarrow *child*.STATE

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

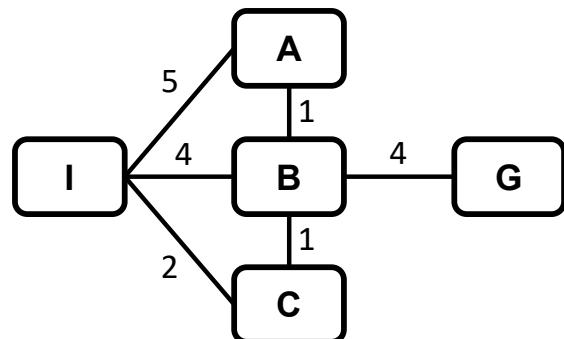
reached[*s*] \leftarrow *child*

 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

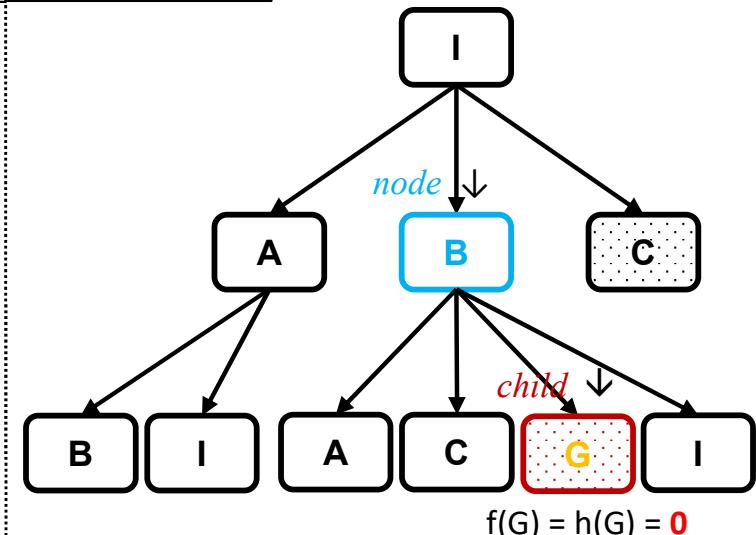
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

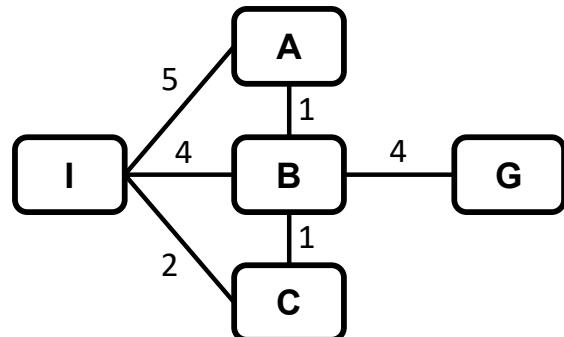
 add *child* to *frontier*

 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

node \leftarrow POP(*frontier*)

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

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

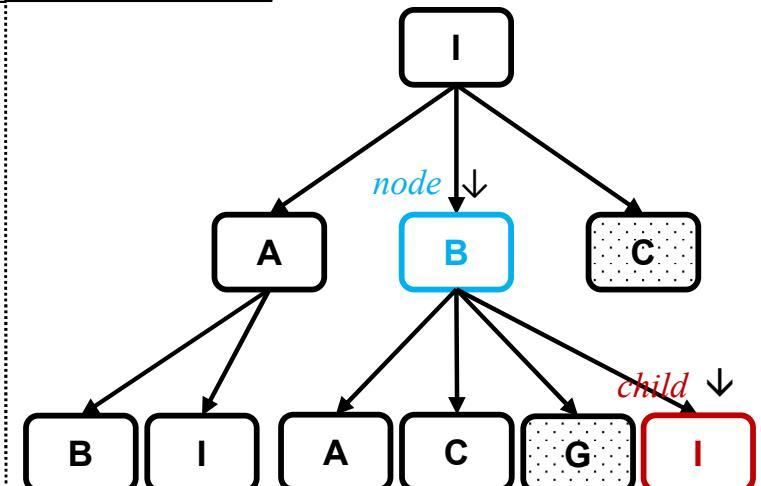
s \leftarrow *child*.STATE

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

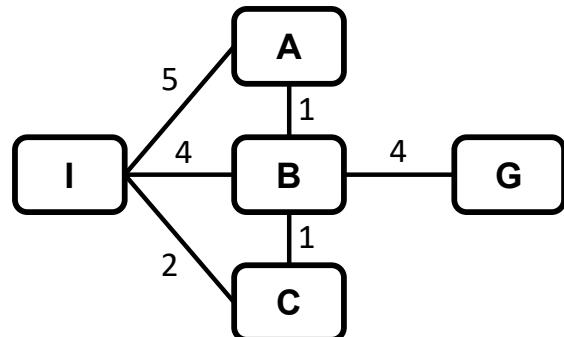
reached[*s*] \leftarrow *child*

 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
```

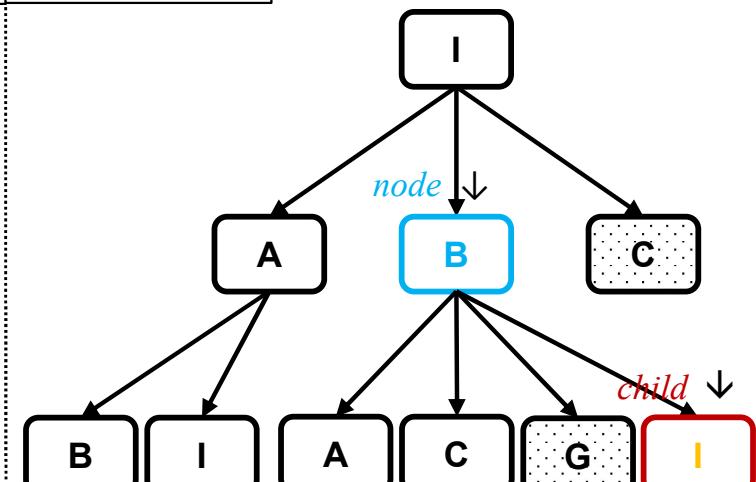
```
             $s \leftarrow child.STATE$ 
```

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

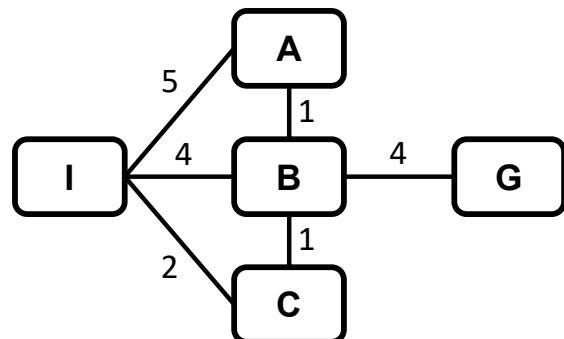
```
                reached[s] ← child
```

```
                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	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

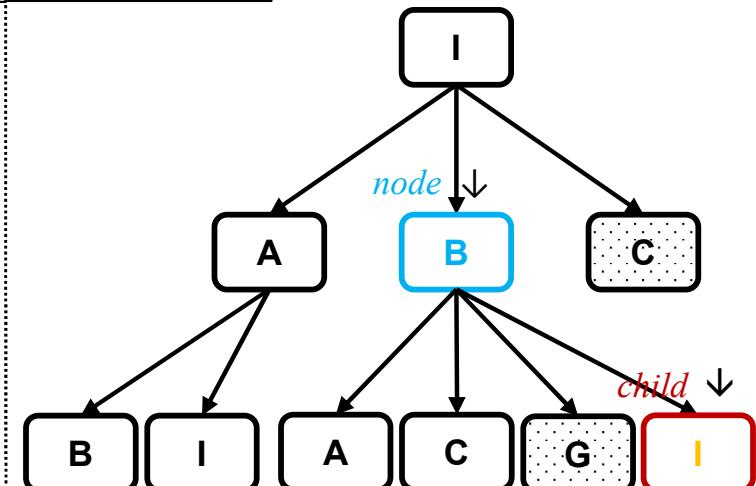
s \leftarrow *child*.STATE

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

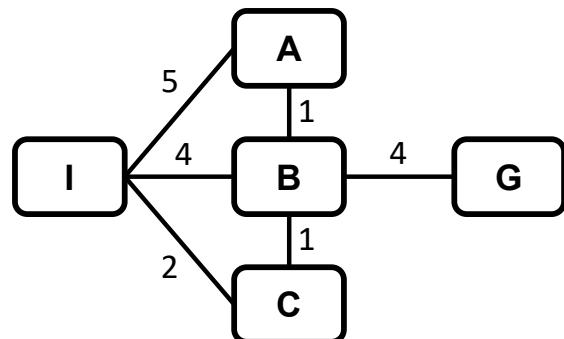
reached[*s*] \leftarrow *child*

 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

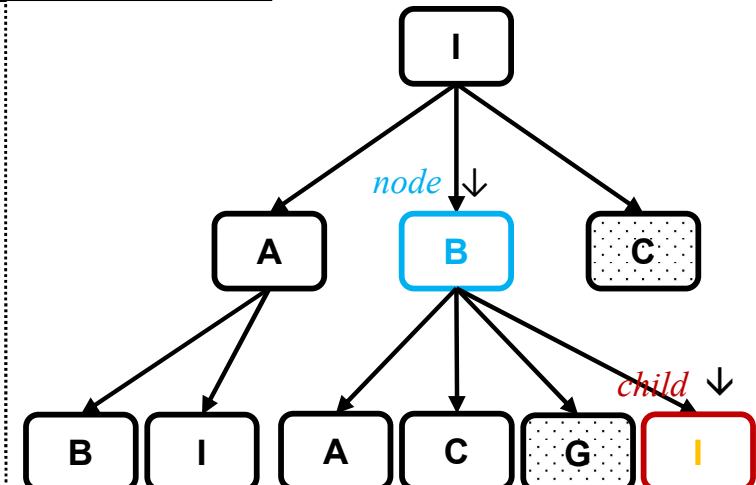
s \leftarrow *child*.STATE

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

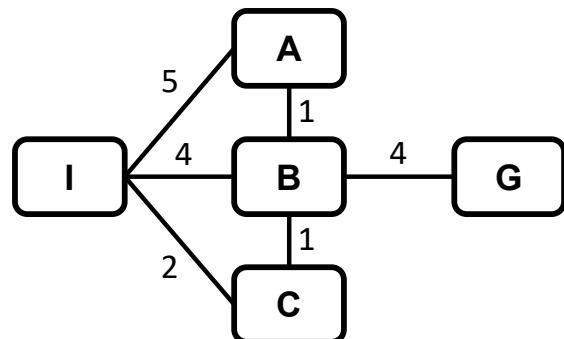
reached[*s*] \leftarrow *child*

 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

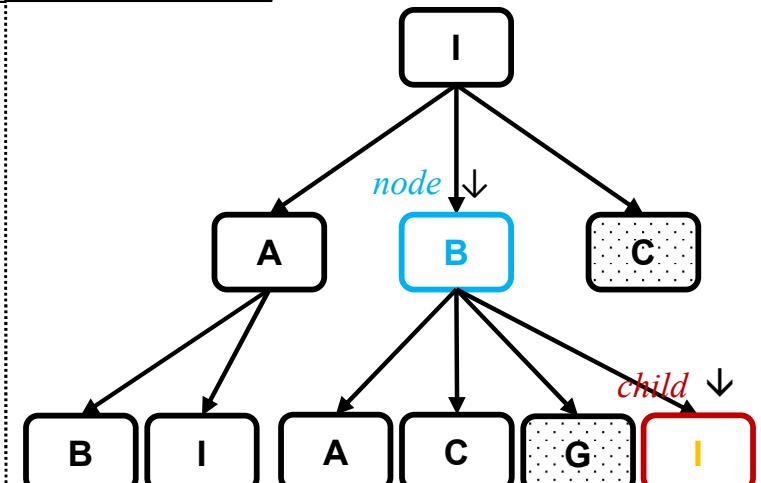
s \leftarrow *child*.STATE

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

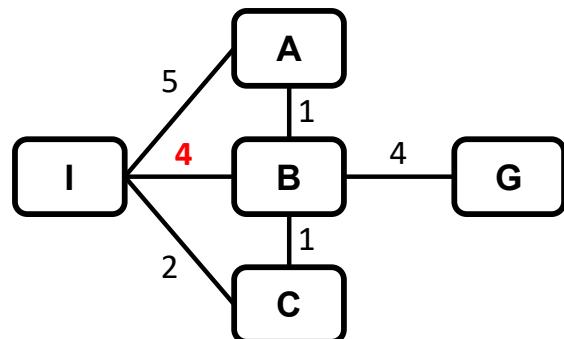
reached[*s*] \leftarrow *child*

 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

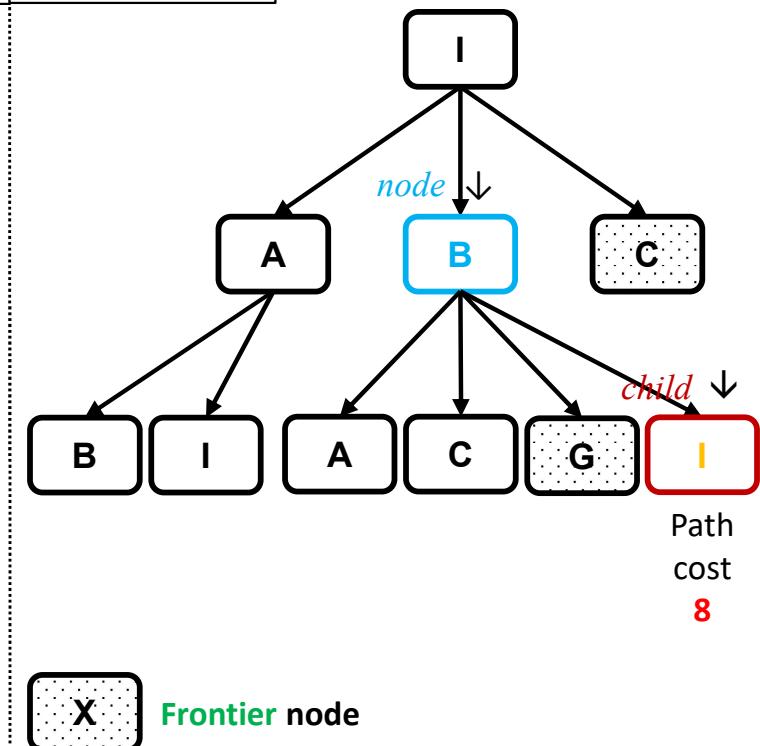
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

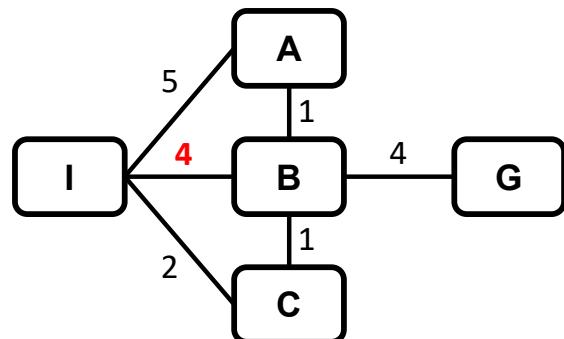
 add *child* to *frontier*

 return failure



Path cost
8

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

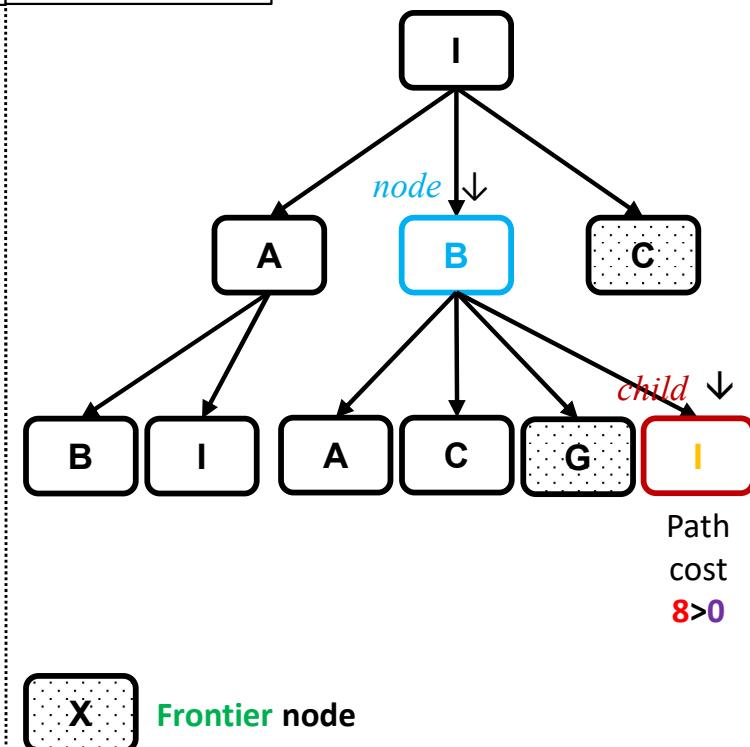
s \leftarrow *child*.STATE

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

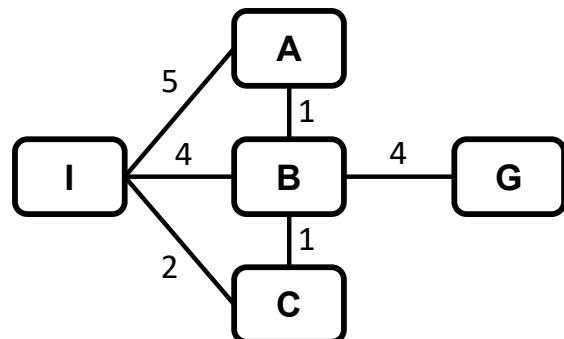
reached[*s*] \leftarrow *child*

 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
```

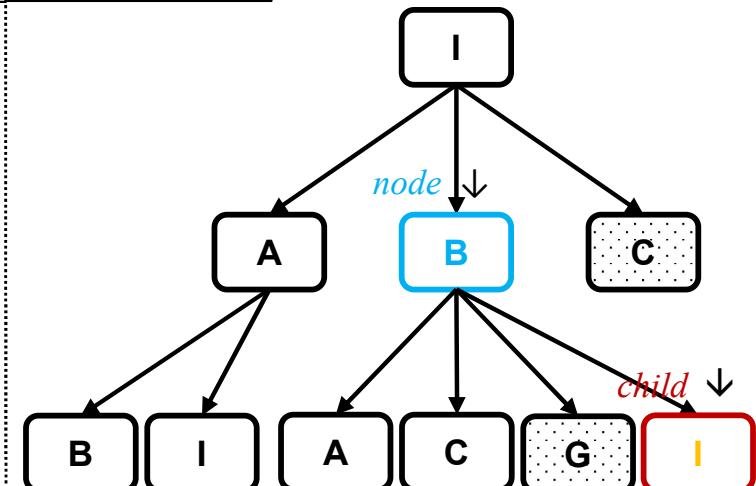
```
             $s \leftarrow child.STATE$ 
```

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

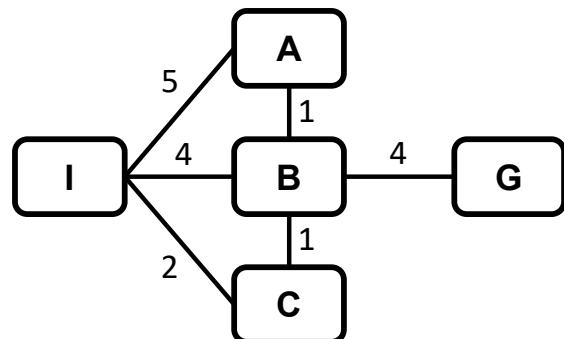
```
                reached[ $s$ ] ←  $child$ 
```

```
                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	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

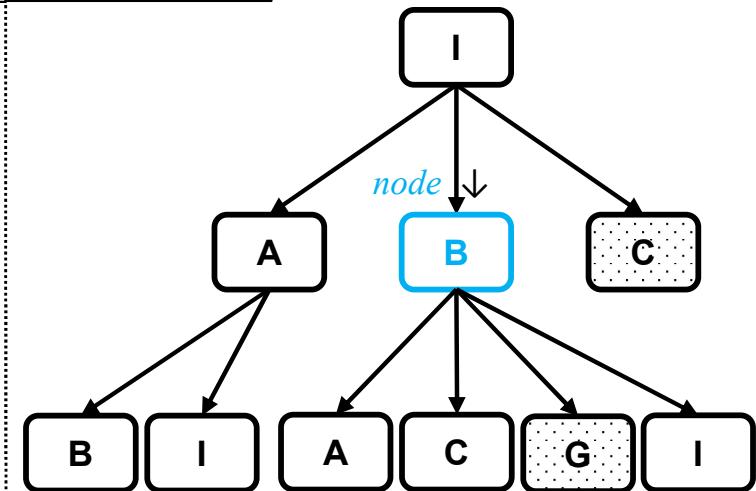
s \leftarrow *child*.STATE

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

reached[*s*] \leftarrow *child*

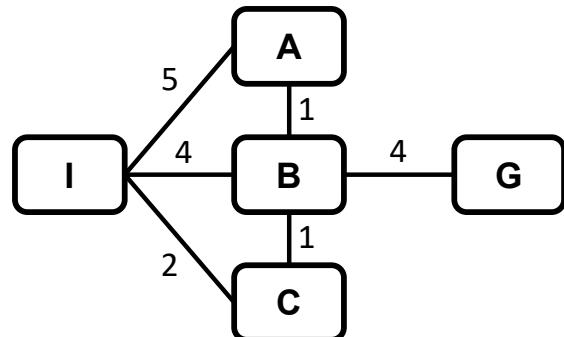
 add *child* to *frontier*

 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

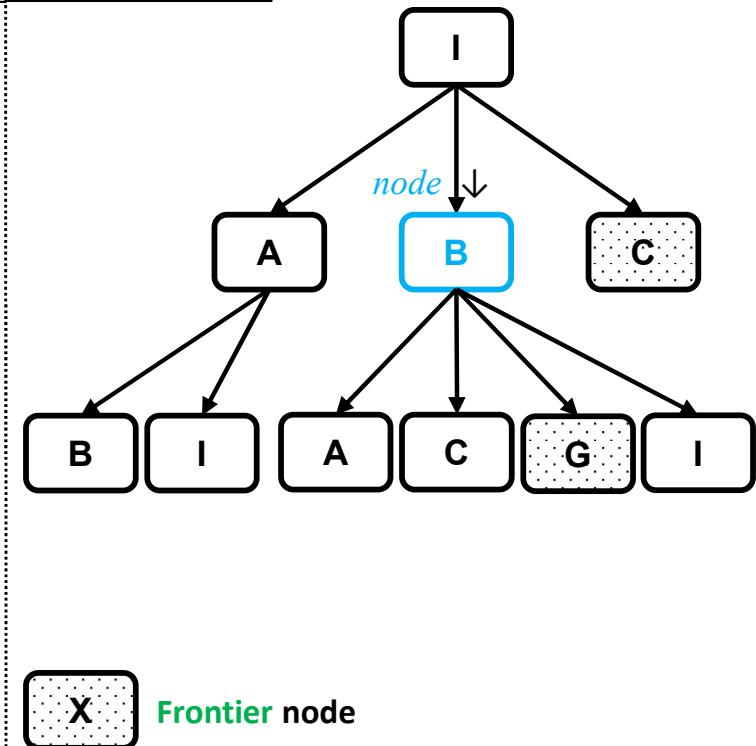
s \leftarrow *child*.STATE

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

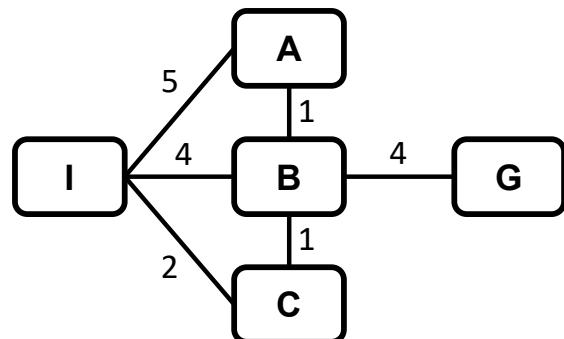
reached[*s*] \leftarrow *child*

 add *child* to *frontier*

 return failure



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

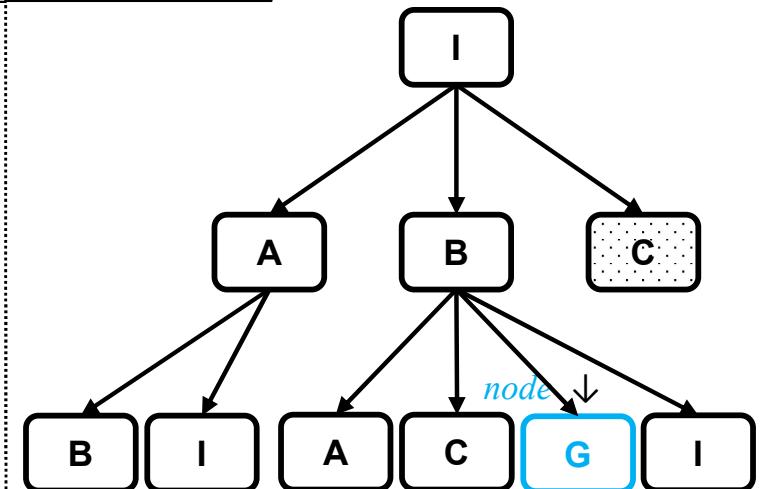
s \leftarrow *child*.STATE

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

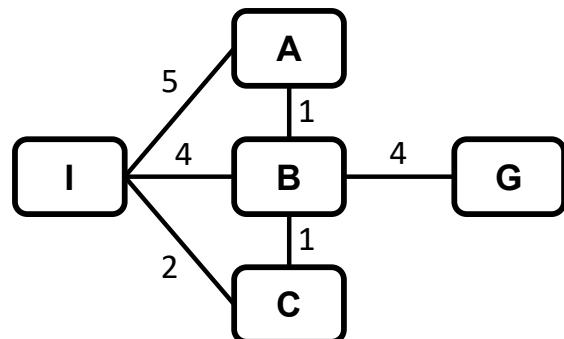
reached[*s*] \leftarrow *child*

 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

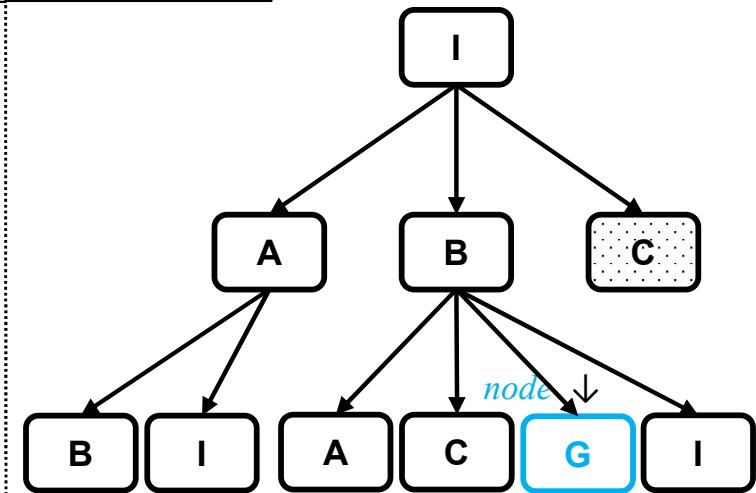
s \leftarrow *child*.STATE

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

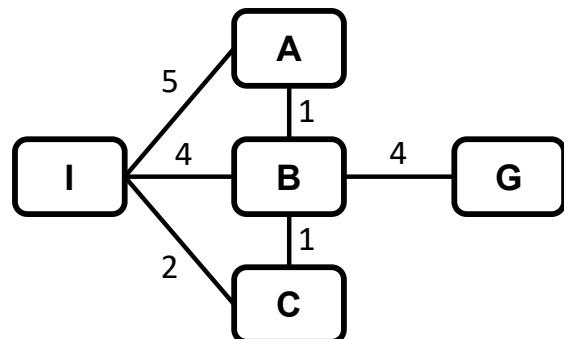
reached[*s*] \leftarrow *child*

 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

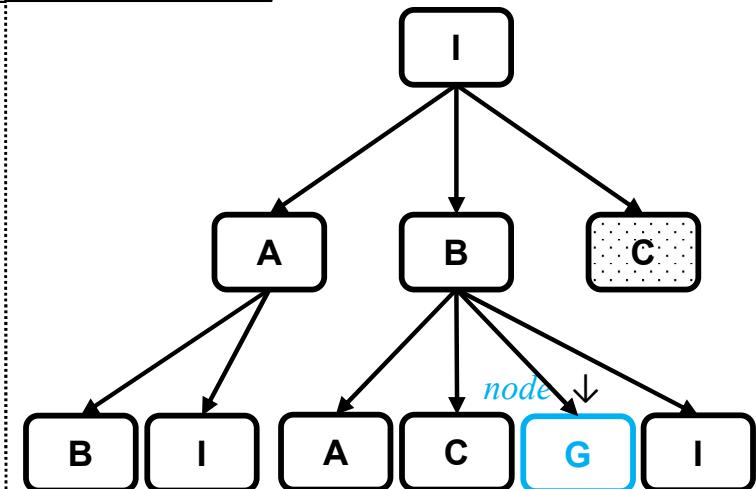
s \leftarrow *child*.STATE

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

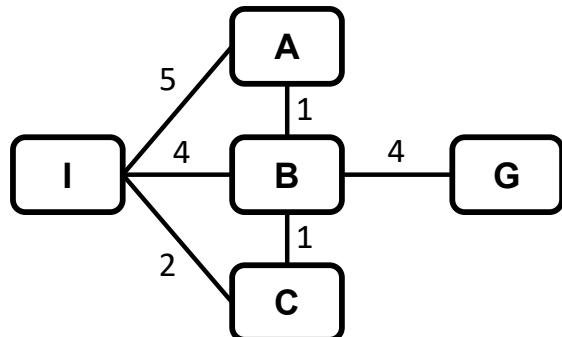
reached[*s*] \leftarrow *child*

 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

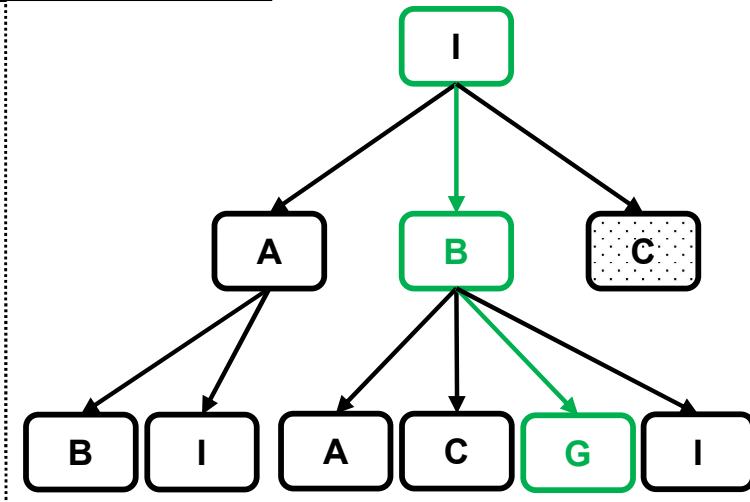
Frontier	Parent	I					
Node	C						
f(Node)	4						

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
  
```



SOLUTION FOUND: I → B → G

