A* Algorithm: Evaluation Function

```
Calculate / obtain:

f(n) = g(State_n) + h(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
                                                     f(n) = g(State_n) + h(State_n)
  node \leftarrow Node(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)
```

Best-First Search: A* Pseudocode

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure

node ← NODE(STATE=problem.INITIAL) f(n) = g(State<sub>n</sub>) + h(State<sub>n</sub>)

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

reached ← a lookup table, with one entry with 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

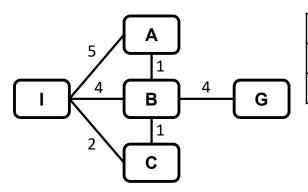
reached[s] ← child

add child to frontier

return failure
```

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



Straight-line distance to Goal state						
State	_	Α	В	C	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent				
Frontier	Node				
ᇁ	f(Node)				
	Darant				
b	Parent				
Reached	Key/State				

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

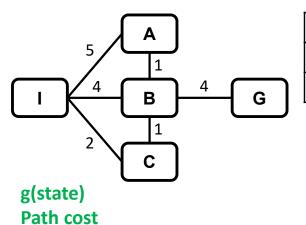
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure

State Space Graph

Algorithm



Straight-line distance to Goal state						
State	-	Α	В	C	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

h(state) Heuristic

J.	Parent				
Frontier	Node				
F	f(Node)				
р	Parent				
Reached	Key/State				
Re	Path cost				

Frontier / Reached

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

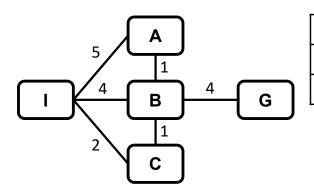
 $s \leftarrow child.STATE$

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



Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

State Space Graph

Frontier / Reached

Algorithm

Search Tree

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

function BEST-FIRST-SEARCH(problem, f)

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

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

while not IS-EMPTY(frontier) do

 $node \leftarrow POP(frontier)$

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

for each child in EXPAND(problem, node) do

 $s \leftarrow child.STATE$

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

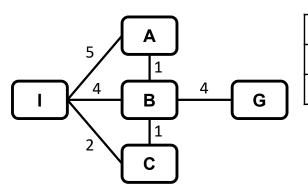
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIA	L	STAT	E:	
GOAL	S	TATE	: G	

er	Parent				
ontie	Node				
Fr	f(Node)				

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

 $s \leftarrow child.STATE$

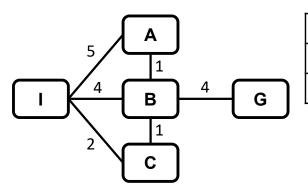
if s is not in reached or child.PATH-COST < reached[s].PATH-COST then

 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure





Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent				
ontier	Node	=			
Fr	f(Node)	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

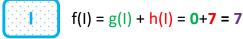
 $s \leftarrow child.STATE$

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

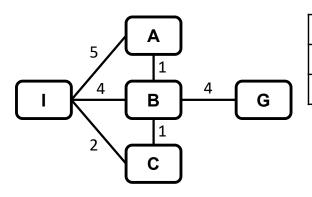
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

e.	Parent				
Frontier	Node	- 1			
Fr	f(Node)	7			
g	Parent				
Reached	Key/State				

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

 $s \leftarrow child.STATE$

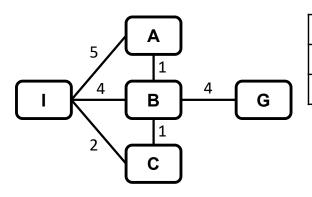
if s is not in reached or child.PATH-COST < reached[s].PATH-COST then

 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure





Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

- S	Parent				
Frontier	Node	I			
ᇁ	f(Node)	7			
p	Parent				
Reached	Key/State	I			
Re	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

 $reached[s] \leftarrow child$

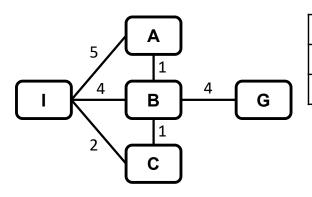
add *child* to *frontier*

return failure



Path cost 0





Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

ē	Parent				
Frontier	Node	ı			
Ţ	f(Node)	7			
<u> </u>	Parent				
Reached	Parent Key/State	 I			

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

 $s \leftarrow child.STATE$

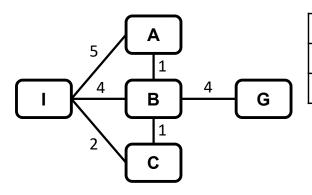
if s is not in reached or child.PATH-COST < reached[s].PATH-COST then

 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure





Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

J.	Parent				
Frontier	Node	I			
Ŧ	f(Node)	7			
b	Parent				
Reached	Parent Key/State	 I			

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

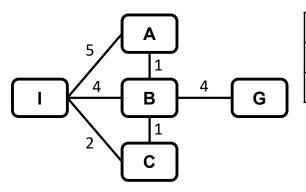
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	_	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent				
Frontier	Node	ı			
Ŧ	f(Node)	7			
p	Parent				
Reached	Key/State	ı			
Re	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

 $s \leftarrow child.STATE$

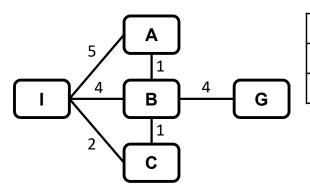
if s is not in reached or child.PATH-COST < reached[s].PATH-COST then

 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure





Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

J.	Parent				
Frontier	Node				
Ŧ	f(Node)				
p	Parent				
Reached	Parent Key/State	 I			

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

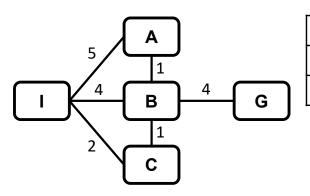
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je Je	Parent				
Frontier	Node				
Ŧ	f(Node)				
l p	Parent				
Reached	Parent Key/State	 I			

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

 $reached[s] \leftarrow child$

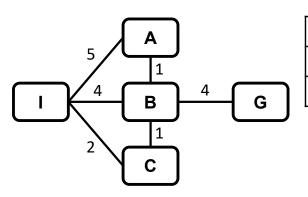
add *child* to *frontier*

return failure



 $node \rightarrow$





Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

J.	Parent				
Frontier	Node				
Ŧ	f(Node)				
p	Parent				
Reached	Parent Key/State	 I			

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

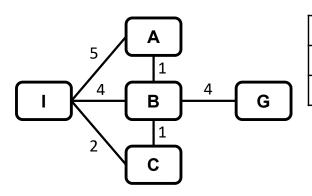
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

J.	Parent				
Frontier	Node				
Ŧ	f(Node)				
p	Parent				
Reached	Parent Key/State	 I			

State Space Graph

Frontier / Reached

Algorithm

Search Tree

 $node \rightarrow$

function BEST-FIRST-SEARCH(problem, f)

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

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

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

while not IS-EMPTY(frontier) do

 $node \leftarrow POP(frontier)$

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

for each child in EXPAND(problem, node) do

 $s \leftarrow child.STATE$

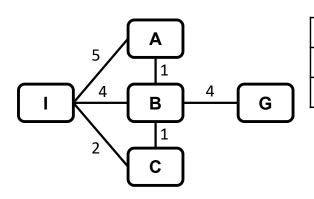
if s is not in reached or child.PATH-COST < reached[s].PATH-COST then

 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure





Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent				
Frontier	Node				
ᇁ	f(Node)				
þ	Parent				
Reached	Parent Key/State	 I			

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

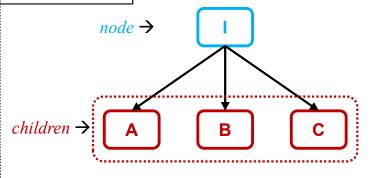
 $s \leftarrow child.STATE$

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

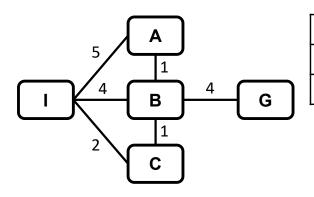
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	_	Α	В	C	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

7	Parent				
Frontier	Node				
Fr	f(Node)				
p	Parent				
Reached	Parent Key/State	 I			

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

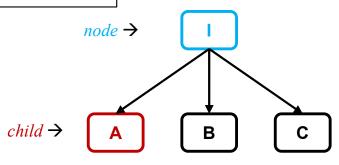
 $s \leftarrow child.STATE$

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

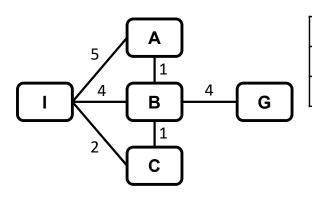
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

J.	Parent				
Frontier	Node				
Ŧ	f(Node)				
p	Parent				
Reached	Parent Key/State	 I			

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

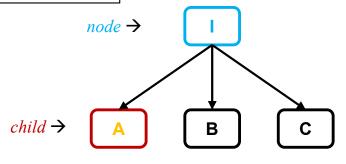
 $s \leftarrow child$.STATE

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

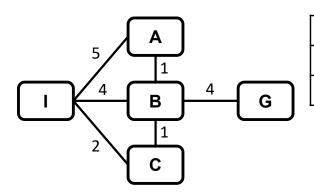
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

7	Parent				
Frontier	Node				
Fr	f(Node)				
p	Parent				
Reached	Parent Key/State	 I			

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

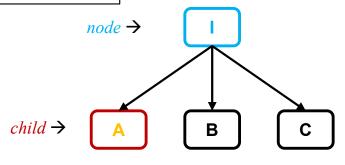
 $s \leftarrow child.STATE$

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

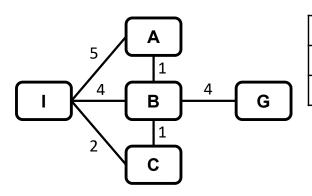
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

<u>-</u>	Parent				
Frontier	Node				
F	f(Node)				
p	Parent				
Reached	Parent Key/State	 I			

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

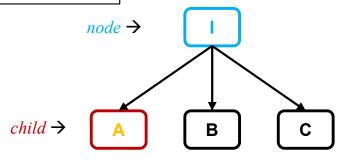
 $s \leftarrow child.STATE$

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

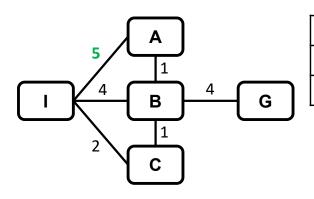
 $reached[s] \leftarrow child$

add child to frontier

return failure







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

-e-	Parent					
Frontier	Node					
Fr	f(Node)					
þ	Parent		1			
Reached	Parent Key/State	 I	I A			

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

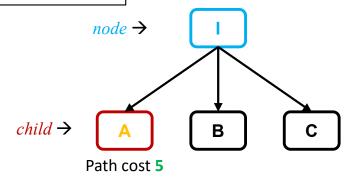
 $s \leftarrow child.STATE$

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

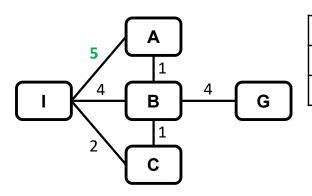
 $reached[s] \leftarrow child$

add child to frontier

return failure







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

S.	Parent	I				
Frontier	Node	A				
Ţ	f(Node)	7				
p	Parent		I			
Reached	Key/State	ı	Α			
a	1		l		l	

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

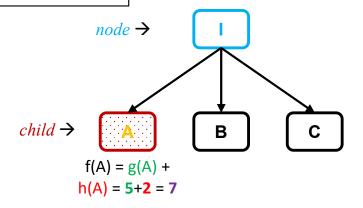
 $s \leftarrow child.STATE$

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

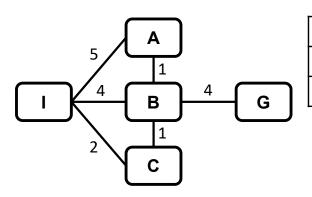
 $reached[s] \leftarrow child$

add child to frontier

return failure







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

S.	Parent	I				
Frontier	Node	Α				
규	f(Node)	7				
p	Parent		I			
Reached	Key/State	I	Α			
Re	Path cost	0	5			

State Space Graph

Frontier / Reached

Algorithm

Search Tree

function BEST-FIRST-SEARCH(problem, f)

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

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

while not IS-EMPTY(frontier) do

 $node \leftarrow POP(frontier)$

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

for each child in EXPAND(problem, node) do

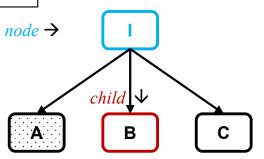
 $s \leftarrow child.STATE$

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

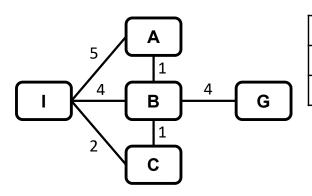
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

S.	Parent	I				
Frontier	Node	Α				
Ţ	f(Node)	7				
p	Parent		I			
Reached	Key/State	ı	Α			
Re	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

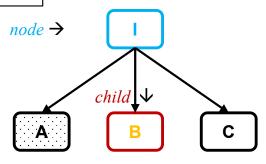
$$s \leftarrow child$$
.STATE

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

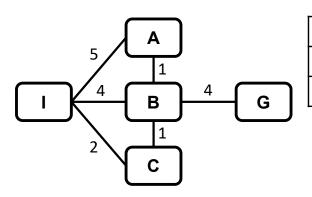
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

S.	Parent	I				
Frontier	Node	Α				
Ţ	f(Node)	7				
p	Parent		I			
Reached	Key/State	ı	Α			
Re	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

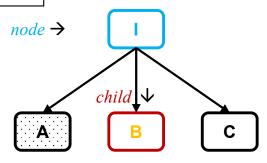
 $s \leftarrow child$.STATE

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

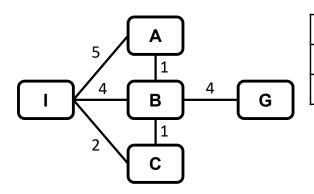
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

J.	Parent	I				
Frontier	Node	Α				
Ŧ	f(Node)	7				
	Davant					
b	Parent		l			
Reached	Key/State	 I	A			

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

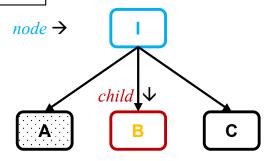
 $s \leftarrow child.STATE$

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

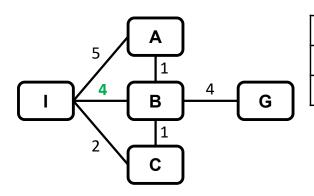
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent	I				
Frontier	Node	Α				
Fr	f(Node)	7				
p	Parent		I	1		
Reached	Parent Key/State	 I	I A	I B		

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

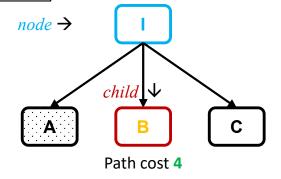
 $s \leftarrow child.STATE$

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

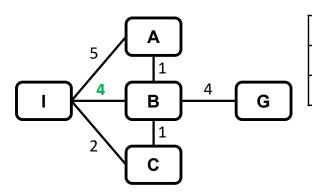
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	Ι	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	-1	I			
Frontier	Node	В	Α			
규	f(Node)	7	7			
p	Parent		I	I		
Reached	Key/State	I	Α	В		
Re	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

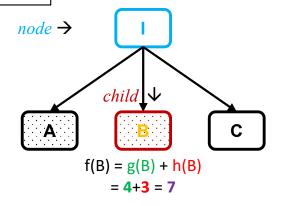
$$s \leftarrow child.STATE$$

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

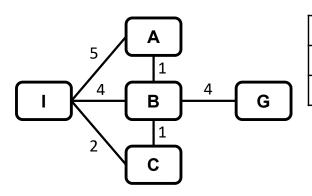
 $reached[s] \leftarrow child$

add child to frontier

return failure







Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I			
Frontier	Node	В	Α			
ᇁ	f(Node)	7	7			
	ı					
b	Parent		I	I		
Reached	Parent Key/State	 I	I A	I B		

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

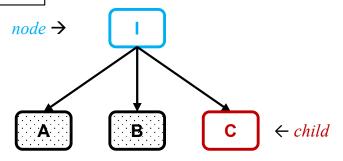
 $s \leftarrow child.STATE$

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

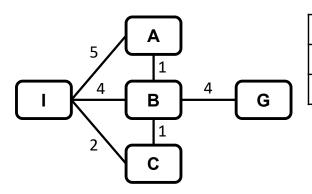
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state					
State	-	Α	В	С	G
h(State)	7	2	3	4	0

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I			
Frontier	Node	В	Α			
ᇁ	f(Node)	7	7			
l g	Parent		I	I		
Reached	Parent Key/State	 I	I A	I B		

State Space Graph

Frontier / Reached

Algorithm

Search Tree

function BEST-FIRST-SEARCH(problem, f)

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

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

while not IS-EMPTY(frontier) do

 $node \leftarrow POP(frontier)$

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

for each child in EXPAND(problem, node) do

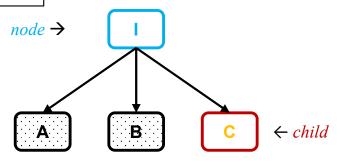
 $s \leftarrow child$.STATE

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

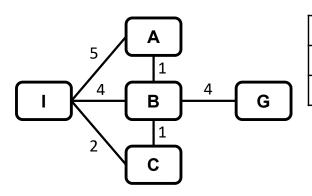
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je.	Parent	I	I			
Frontier	Node	В	Α			
Ţ	f(Node)	7	7			
b	Parent		I	I		
Reached	Parent Key/State	 I	I A	I B		

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

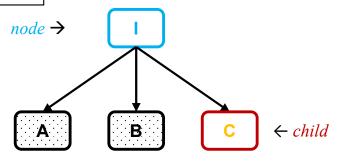
 $s \leftarrow child$.STATE

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

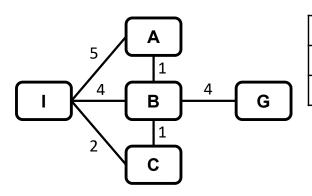
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

J.	Parent	I	I			
Frontier	Node	В	Α			
Ţ	f(Node)	7	7			
p	Parent		I	I		
Reached	Key/State	ı	Α	В		
Re	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

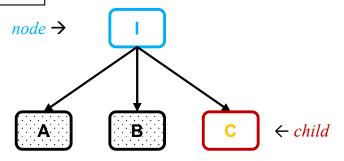
 $s \leftarrow child.STATE$

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

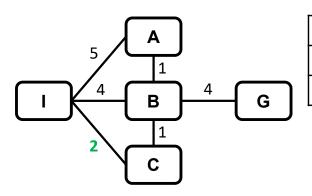
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

-G	Parent	I	I				
Frontier	Node	В	Α				
Ŧ	f(Node)	7	7				
þ	Parent		I	I	-1		
Reached	Parent Key/State	 I	I A	I B	I C		

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

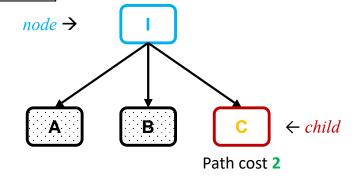
 $s \leftarrow child$.STATE

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

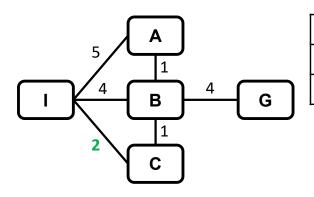
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je.	Parent	-1	I	Ι			
Frontier	Node	С	В	Α			
ᇁ	f(Node)	6	7	7			
p	Parent		I	I	I		
Reached	Parent Key/State		I A	В	l C		

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

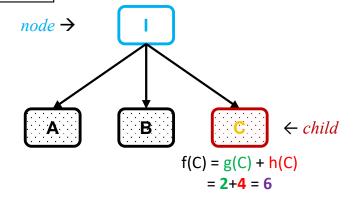
 $s \leftarrow child.STATE$

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

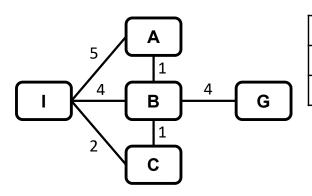
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I	I			
Frontier	Node	С	В	Α			
Ţ.	f(Node)	6	7	7			
p	Parent		I	I	I		
Reached	Key/State	ı	Α	В	С		
Re	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

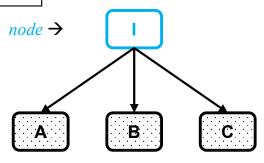
 $s \leftarrow child.STATE$

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

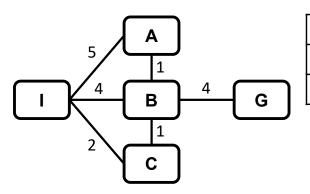
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent	I	I	I			
Frontier	Node	С	В	Α			
Fr	f(Node)	6	7	7			
pa	Parent		I	I	I		
Reached	Key/State	I	Α	В	С		
Re	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

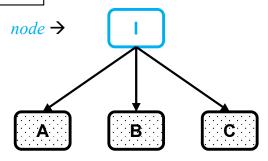
 $s \leftarrow child.STATE$

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

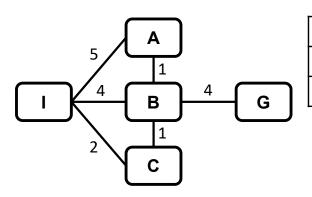
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent	I	I				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
þ	Parent		I	I	I		
Reached	Parent Key/State	 I	I A	I B	l C		

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

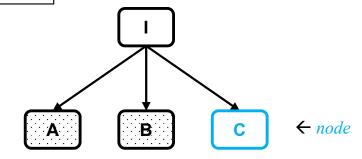
 $s \leftarrow child.STATE$

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

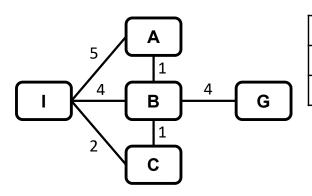
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

-e-	Parent	I	I				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
pa	Parent		I	I	I		
Reached	Parent Key/State	 I	I A	I B	l C		

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

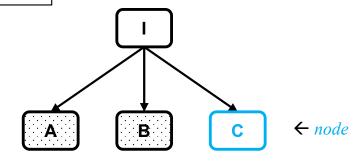
 $s \leftarrow child.STATE$

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

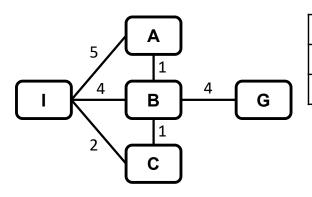
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I				
Frontier	Node	В	Α				
규	f(Node)	7	7				
p	Parent		I	I	I		
Reached	Parent Key/State	 I	I A	I B	l C		

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

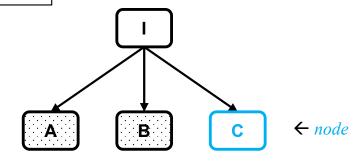
 $s \leftarrow child$.STATE

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

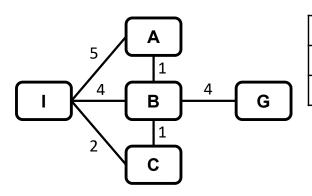
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I				
Frontier	Node	В	Α				
규	f(Node)	7	7				
p	Parent		I	I	I		
Reached	Parent Key/State	 I	I A	I B	l C		

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

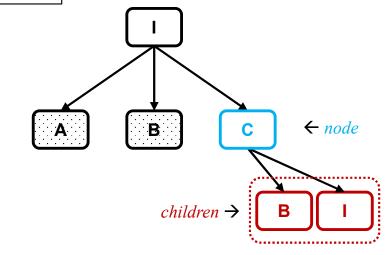
 $s \leftarrow child.STATE$

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

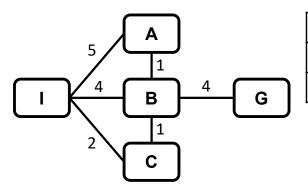
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

S.	Parent	I	I				
Frontier	Node	В	Α				
Ţ.	f(Node)	7	7				
p	Parent		I	I	I		
Reached	Key/State	ı	Α	В	С		
Re	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

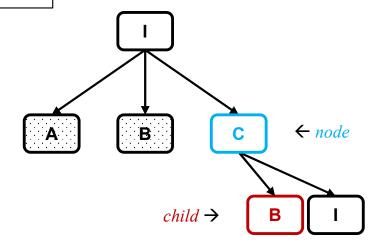
 $s \leftarrow child.STATE$

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

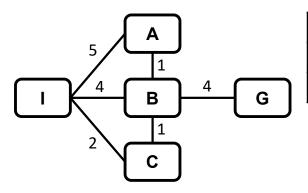
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

Je.	Parent	I	I				
Frontier	Node	В	Α				
Ţ	f(Node)	7	7				
p	Parent		I	I	I		
Reached	Parent Key/State	 I	I A	I B	l C		

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

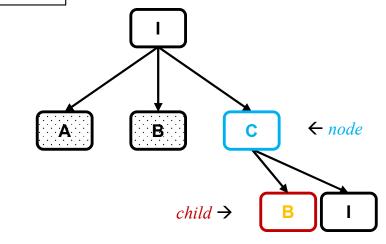
 $s \leftarrow child$.STATE

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

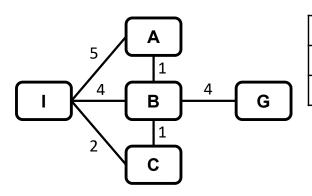
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	I	I				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
p	Parent		I	I	I		
Reached	Key/State	ı	Α	В	С		
Re	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

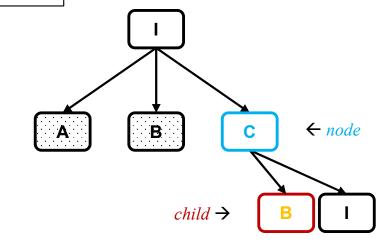
 $s \leftarrow child.STATE$

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

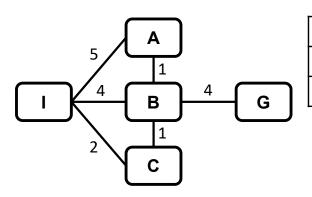
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

-S-	Parent	I	I				
Frontier	Node	В	Α				
Ţ	f(Node)	7	7				
p	Parent		I	I	I		
Reached	Parent Key/State		I A	В	l C		

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

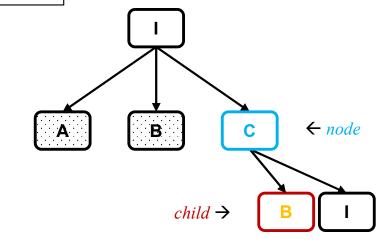
if s is not

(ached or child.PATH-COST < reached[s].PATH-COST then

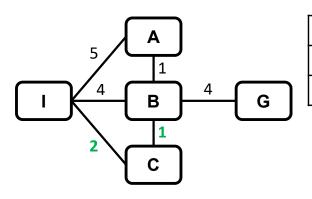
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

Je.	Parent	I	I				
Frontier	Node	В	Α				
표	f(Node)	7	7				
l p	Parent		I	- 1	I		
Reached	Parent Key/State	 I	I A	В	l C		

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

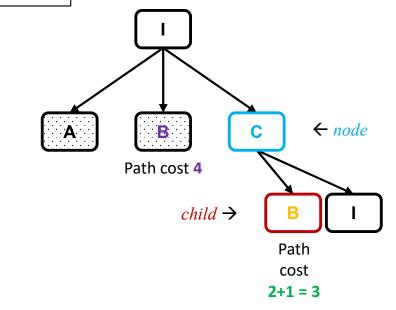
 $s \leftarrow child.STATE$

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

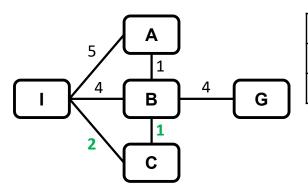
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

S.	Parent	I	I				
Frontier	Node	В	Α				
Ţ.	f(Node)	7	7				
p	Parent		I	-1	I		
Reached	Key/State	ı	Α	В	С		
Re	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

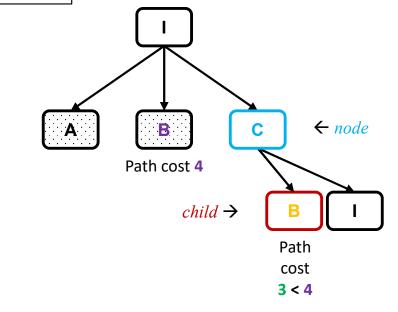
 $s \leftarrow child$.STATE

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

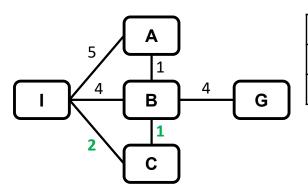
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

e.	Parent	I	I				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
р	Parent		I	-1	I		
Reached	Key/State	I	Α	В	С		
Re	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

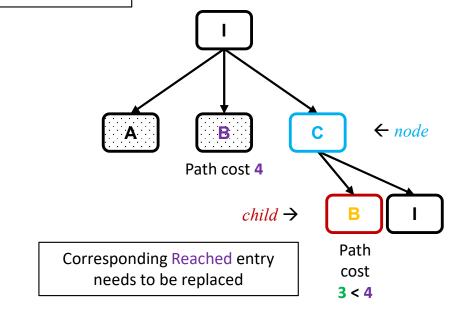
 $s \leftarrow child$.STATE

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

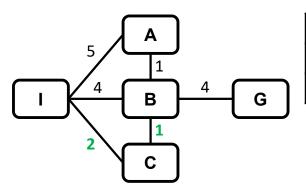
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	I	I				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

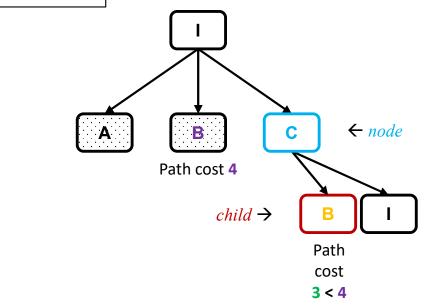
 $s \leftarrow child$.STATE

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

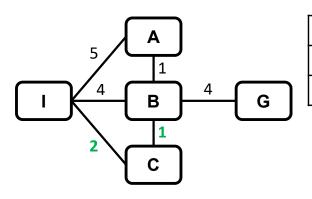
 $reached[s] \leftarrow child$

add child to frontier

return failure







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

er	Parent	С	I	I			
Frontier	Node	В	В	Α			
Fr	f(Node)	6	7	7			
ъ							
p	Parent		I	С	I		
Reached	Parent Key/State	 I	I A	C B	C		

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

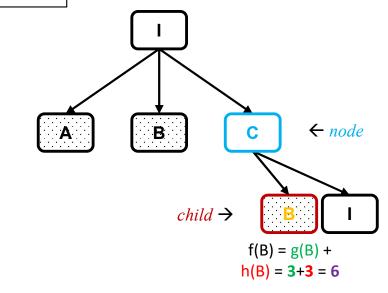
 $s \leftarrow child.STATE$

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

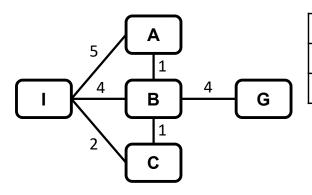
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

INITIAL STATE: I GOAL STATE: G

Je Je	Parent	С	I	I			
Frontier	Node	В	В	Α			
Ŧ	f(Node)	6	7	7			
p	Parent		I	С	I		
Reached	Key/State	I	Α	В	С		
Re	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

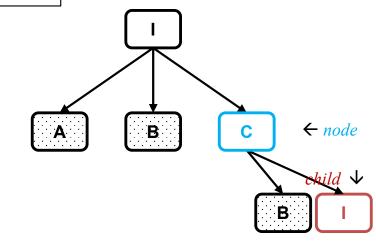
 $s \leftarrow child.STATE$

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

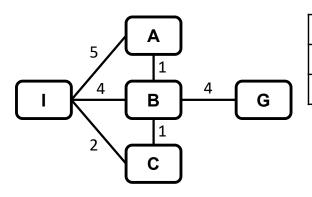
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

INITIAL STATE: I GOAL STATE: G

J.	Parent	С	I	ı			
Frontier	Node	В	В	Α			
Ţ	f(Node)	6	7	7			
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

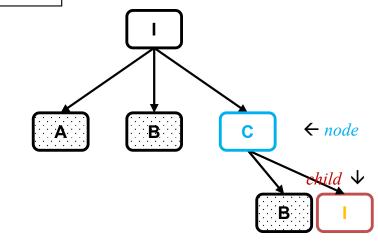
 $s \leftarrow child.STATE$

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

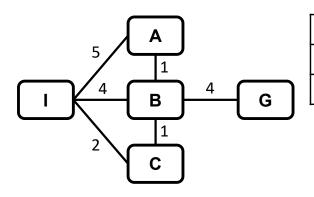
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

Je.	Parent	С	I	I			
Frontier	Node	В	В	Α			
ᇁ	f(Node)	6	7	7			
p	Parent		I	С	I		
Reached	Key/State	I	Α	В	С		
Re	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

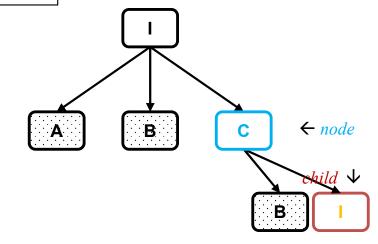
 $s \leftarrow child.STATE$

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

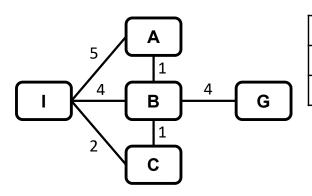
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent	С	Ι	Ι			
Frontier	Node	В	В	Α			
Fr	f(Node)	6	7	7			
þ	Parent		I	С	I		
Reached	Parent Key/State	 I	I A	C B	l C		

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

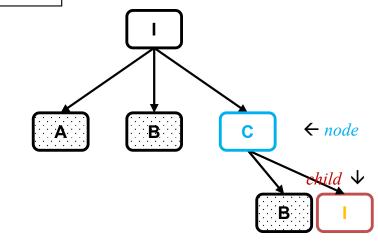
if s is not

(ached or child.PATH-COST < reached[s].PATH-COST then

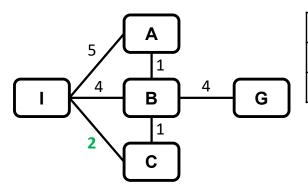
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	С	I	ı			
Frontier	Node	В	В	Α			
Ţ.	f(Node)	6	7	7			
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

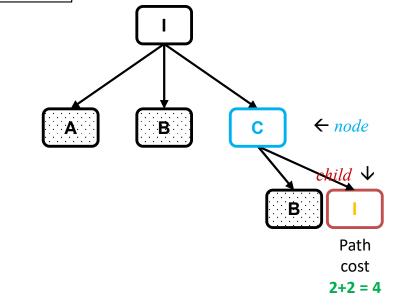
 $s \leftarrow child$.STATE

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

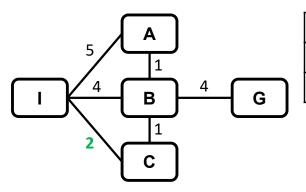
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	_	Α	В	C	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	С	I	ı			
Frontier	Node	В	В	Α			
Ţ.	f(Node)	6	7	7			
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

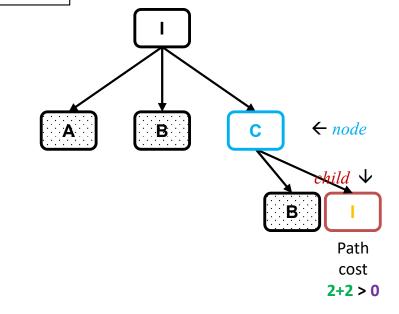
 $s \leftarrow child$.STATE

if s is not cached or child.PATH-COST

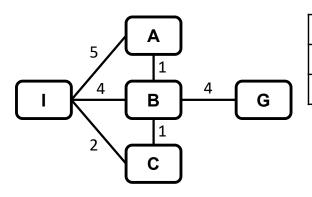
(ached[s].PATH-COST then

reache child add child frontier

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je.	Parent	С	I	I			
Frontier	Node	В	В	Α			
Ŧ	f(Node)	6	7	7			
p	Parent		I	С	I		
Reached	Key/State	I	Α	В	С		
Re	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

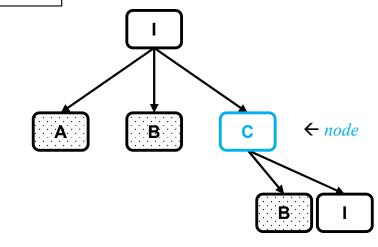
 $s \leftarrow child.STATE$

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

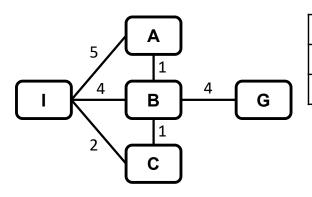
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je.	Parent	С	I	I			
Frontier	Node	В	В	Α			
Ŧ	f(Node)	6	7	7			
p	Parent		I	С	I		
Reached	Parent Key/State	 I	I A	C B	l C		

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

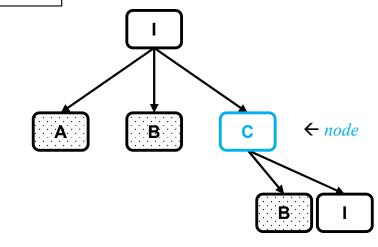
 $s \leftarrow child.STATE$

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

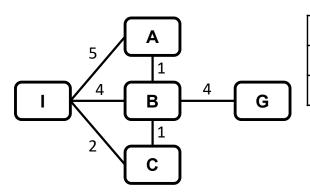
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	ı	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je.	Parent	I	I				
Frontier	Node	В	Α				
Ţ	f(Node)	7	7				
b	Parent		I	С	I		
Reached	Parent Key/State	 I	I A	C B	l C		

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

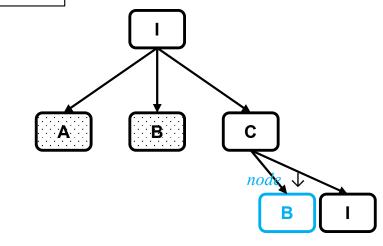
 $s \leftarrow child.STATE$

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

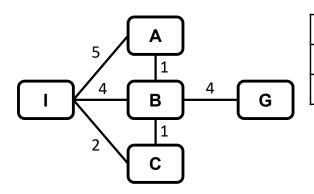
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I				
Frontier	Node	В	Α				
Ţ.	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

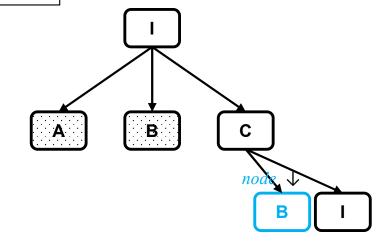
 $s \leftarrow child$.STATE

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

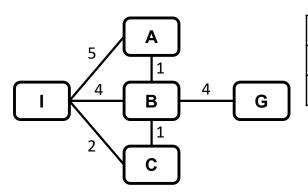
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	Ι	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I				
Frontier	Node	В	Α				
ᇁ	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Parent Key/State	 I	I A	C B	l C		

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

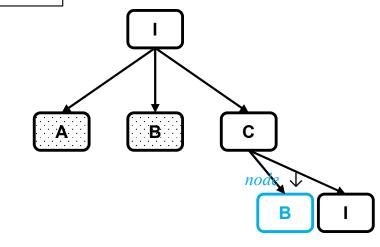
 $s \leftarrow child$.STATE

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

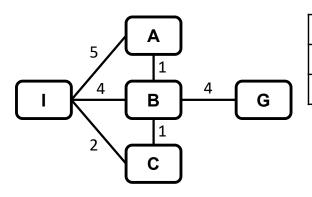
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent	I	Ι				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Key/State	I	Α	В	С		
Re	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

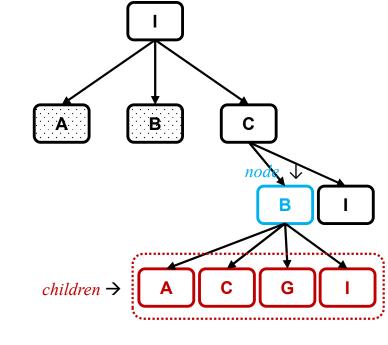
 $s \leftarrow child.STATE$

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

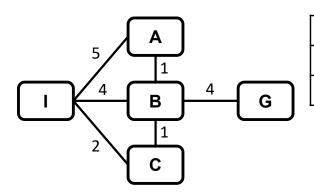
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

S.	Parent	I	I				
Frontier	Node	В	Α				
Ţ.	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

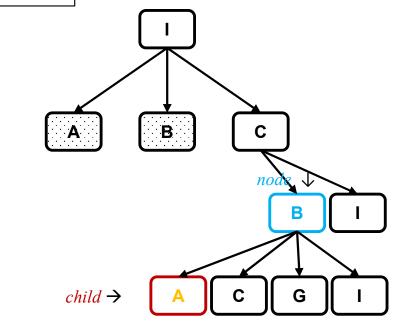
 $s \leftarrow child.STATE$

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

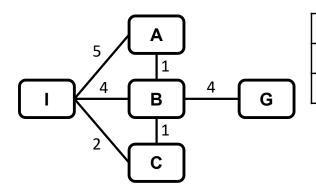
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	C	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je.	Parent	I	I				
Frontier	Node	В	Α				
Ţ	f(Node)	7	7				
p	Parent		ı	С	ı		
Reached	Parent Key/State	 I	I A	C B	I C		

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

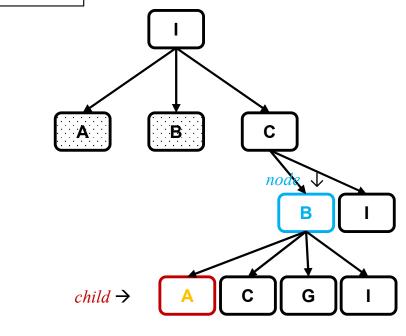
 $s \leftarrow child$.STATE

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

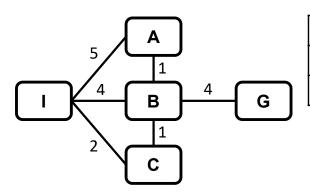
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	_	Α	В	C	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

Je.	Parent	I	I				
Frontier	Node	В	Α				
Ţ	f(Node)	7	7				
b	Parent		I	С	I		
Reached	Parent Key/State	 I	I A	C B	l C		

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

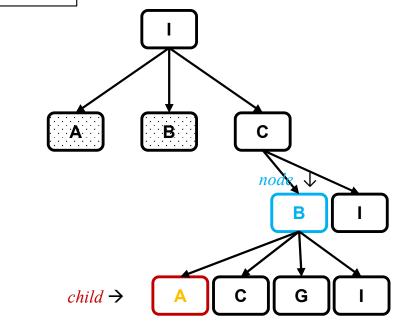
 $s \leftarrow child$.STATE

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

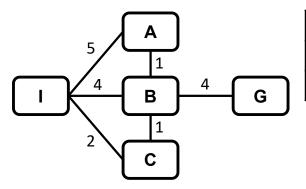
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	I	I				
Frontier	Node	В	Α				
Ţ	f(Node)	7	7				
b	Parent		I	С	I		
Reached	Parent Key/State	 I	I A	C B	l C		

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

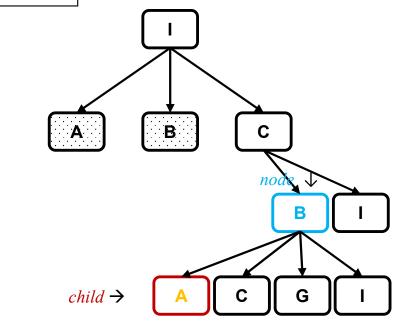
 $s \leftarrow child$.STATE

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

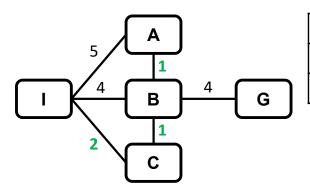
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	I	I				
Frontier	Node	В	Α				
Ţ.	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

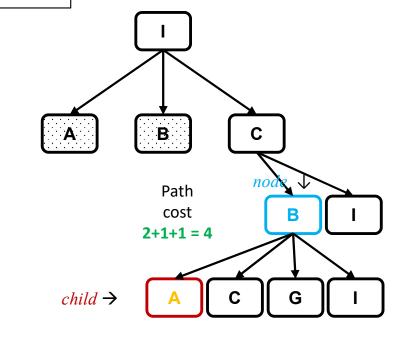
 $s \leftarrow child$.STATE

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

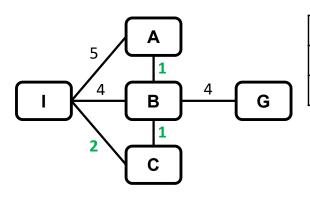
 $reached[s] \leftarrow child$

add child to frontier

return failure







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

er	Parent	I	I				
Frontier	Node	В	Α				
Ţ.	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

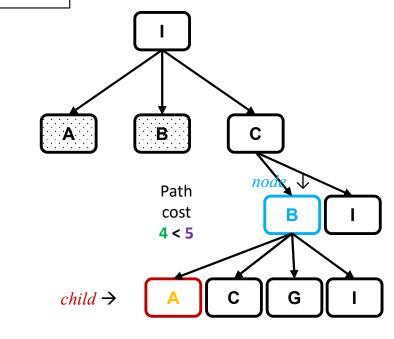
 $s \leftarrow child$.STATE

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

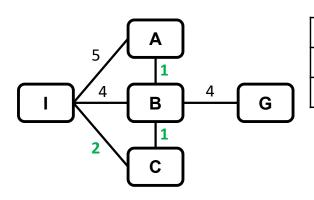
 $reached[s] \leftarrow child$

add child to frontier

return failure







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

er	Parent	I	I				
Frontier	Node	В	Α				
Ţ.	f(Node)	7	7				
p	Parent		I	С	I		
Reached	Key/State	ı	Α	В	С		
Re	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

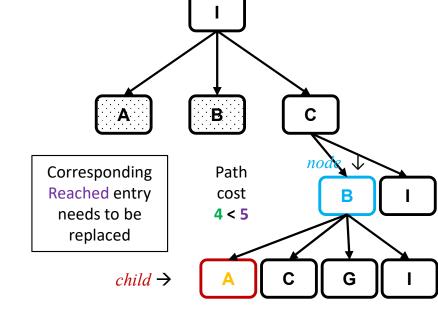
 $s \leftarrow child$.STATE

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

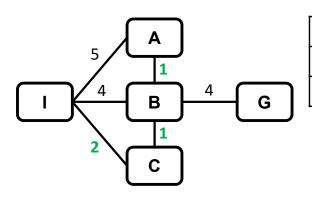
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

Je.	Parent	I	Ι				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
p	Parent		В	С	I		
Reached	Key/State	ı	Α	В	С		
Re	Path cost	0	4	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

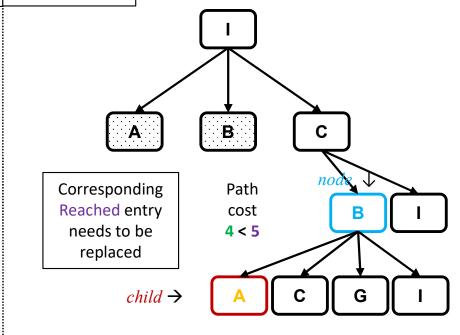
 $s \leftarrow child$.STATE

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

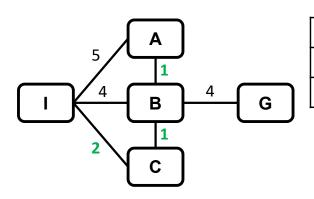
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	Ι	Ι			
Frontier	Node	A	В	Α			
Ţ	f(Node)	6	7	7			
р	Parent		В	С	I		
Reached	Key/State	I	Α	В	С		
Re	Path cost	0	4	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

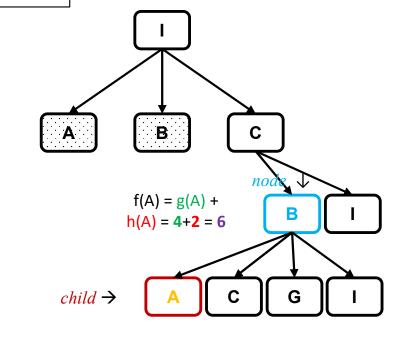
 $s \leftarrow child$.STATE

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

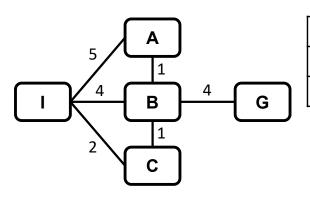
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	ı	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

Je.	Parent	В	I	I			
Frontier	Node	Α	В	Α			
Ŧ	f(Node)	6	7	7			
p	Parent		В	С	I		
Reached	Key/State	I	Α	В	С		
Re	Path cost	0	4	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

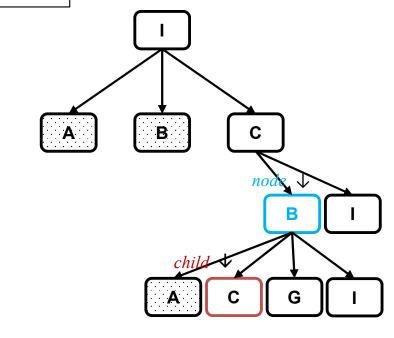
 $s \leftarrow child.STATE$

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

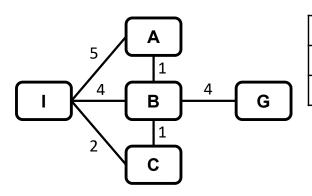
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	-	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	Ι				
Frontier	Node	Α	В				
Fr	f(Node)	6	7				
$\overline{}$							
D D	Parent		В	С	Ι		
Reached	Parent Key/State	 I	B A	C B	С		

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

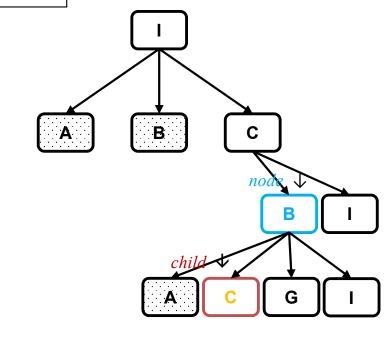
 $s \leftarrow child$.STATE

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

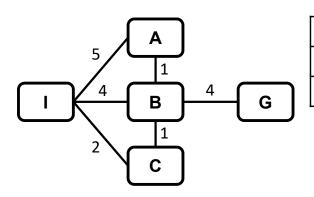
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent	В	I	I			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
p	Parent		В	С	I		
Reached	Key/State	ı	Α	В	С		
Re	Path cost	0	4	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

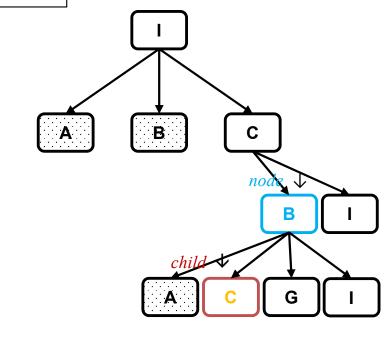
 $s \leftarrow child$.STATE

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

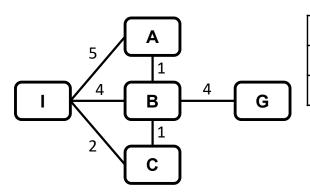
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	ı	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	Ι	Ι			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
þ	Parent		В	С	I		
Reached	Parent Key/State	 I	B A	C B	C		

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

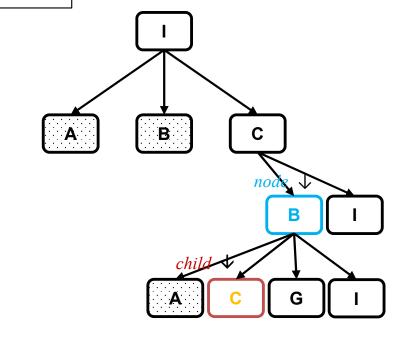
 $s \leftarrow child$.STATE

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

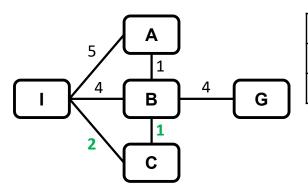
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	1	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

S.	Parent	В	I	Ι			
Frontier	Node	Α	В	Α			
Ŧ	f(Node)	6	7	7			
g	Parent		В	С	I		
Reached	Parent Key/State	 I	B A	C B	C		

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

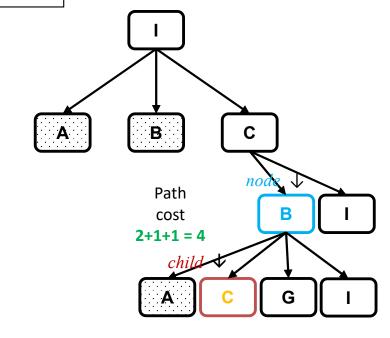
if s is not sached or child.PATH-COSD

(ached[s].PATH-COST then

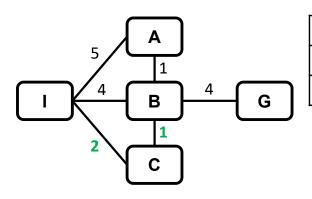
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	ı	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	Ι	Ι			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
b	Parent		В	С	1		
Reached	Parent Key/State	 I	B A	C B	l C		

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

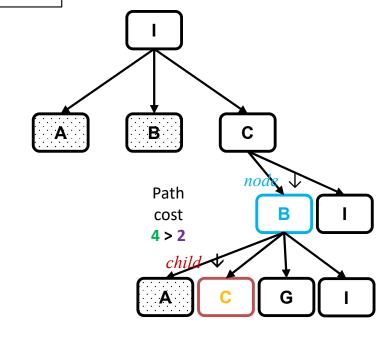
if s is not sached or child.PATH-COST

(ached[s].PATH-COST then

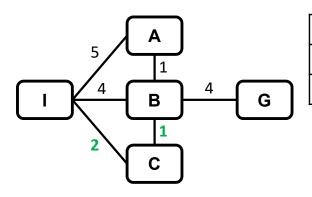
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	I	Ι			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
р	Parent		В	С	I		
Reached	Key/State	I	Α	В	С		

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

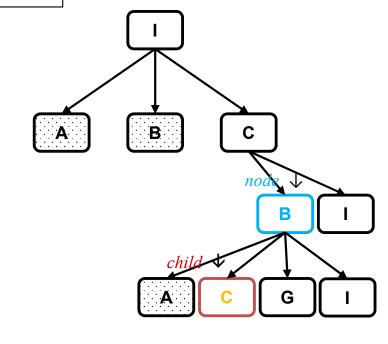
 $s \leftarrow child$.STATE

if s is not eached or child.PATH-COSD

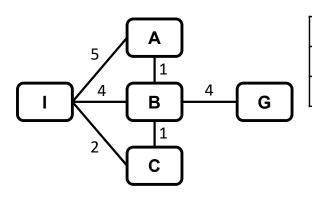
⟨ached[s].PATH-COST then

reache child add child frontier

return failure







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

J.	Parent	В	I	ı			
Frontier	Node	Α	В	Α			
Ţ	f(Node)	6	7	7			
p	Parent		В	С	ļ		
Reached	Key/State	ı	Α	В	С		
Re	Path cost	0	4	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

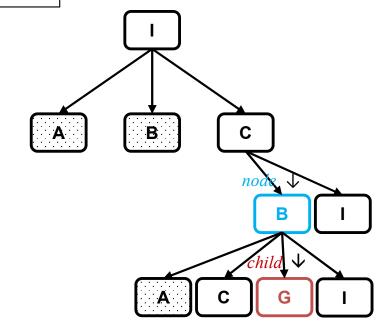
 $s \leftarrow child.STATE$

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

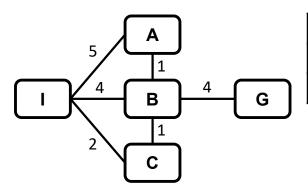
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

Je.	Parent	В	I	I			
Frontier	Node	Α	В	Α			
Ŧ	f(Node)	6	7	7			
p	Parent		В	С	I		
Reached	Key/State	I	Α	В	С		
Re	Path cost	0	4	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

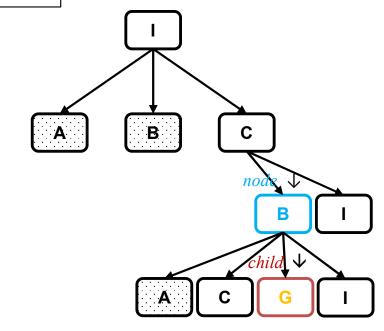
 $s \leftarrow child$.STATE

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

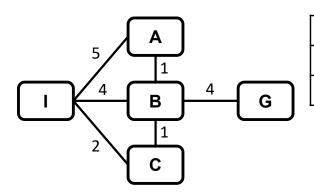
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

-e-	Parent	В	I	I			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
	_						
g	Parent		В	С	1		
Reached	Parent Key/State	 I	B A	C B	C		

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

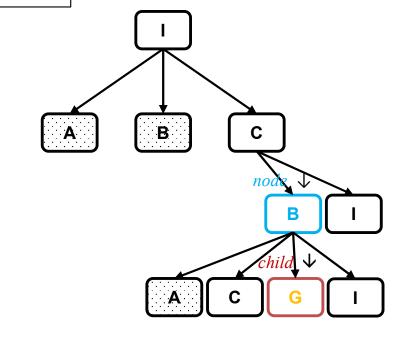
 $s \leftarrow child$.STATE

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

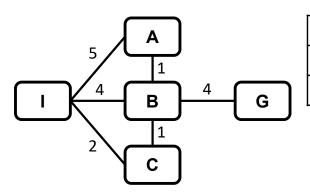
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	Ι	Ι			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
þ	Parent		В	С	I		
Reached	Parent Key/State	 I	B A	C B	C		

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

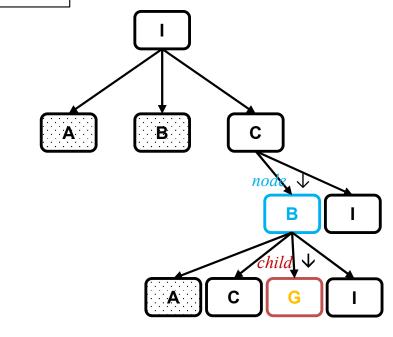
 $s \leftarrow child.STATE$

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

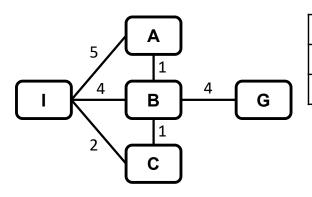
 $reached[s] \leftarrow child$

add child to frontier

return failure







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

er	Parent	В	I	I			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
p	Parent		В	С	I		
Reached	Key/State	I	Α	В	С		
Re	Path cost	0	4	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

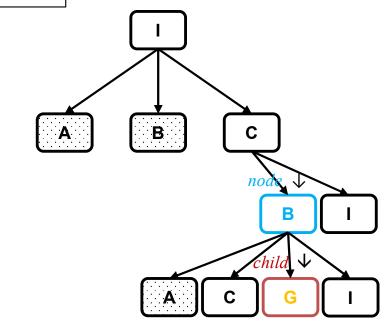
 $s \leftarrow child.STATE$

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

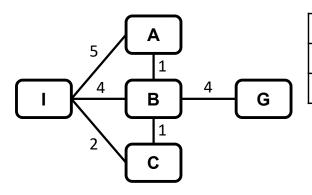
 $reached[s] \leftarrow child$

add child to frontier

return failure







Straight-line distance to Goal state							
State	-	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	I	I			
Frontier	Node	Α	В	Α			
Fr	f(Node)	6	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	I	Α	В	С	G	
Re	Path cost	0	4	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

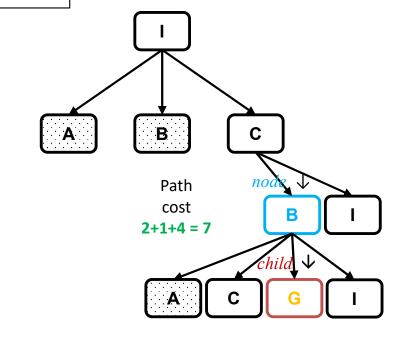
 $s \leftarrow child.STATE$

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

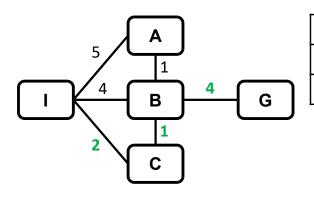
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	_	Α	В	C	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	В	I	I	В	
Frontier	Node	Α	G	В	Α	G	
Fr	f(Node)	6	7	7	7	7	
p	Parent		В	С	I	В	
Reached	Key/State	I	Α	В	С	G	
Re	Path cost	0	4	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

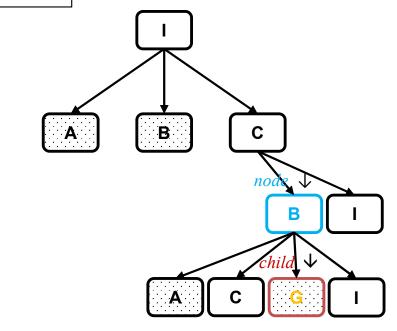
 $s \leftarrow child.STATE$

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

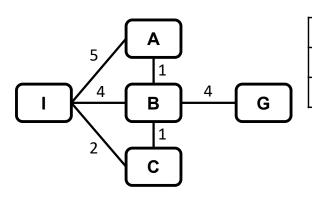
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	-	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	В	I	I		
Frontier	Node	Α	G	В	Α		
Fr	f(Node)	6	7	7	7		
þ	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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

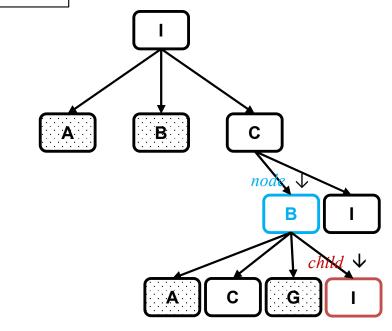
 $s \leftarrow child.STATE$

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

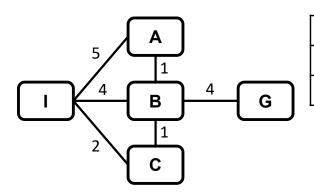
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	-	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	В	I	I		
Frontier	Node	Α	G	В	Α		
Fr	f(Node)	6	7	7	7		
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

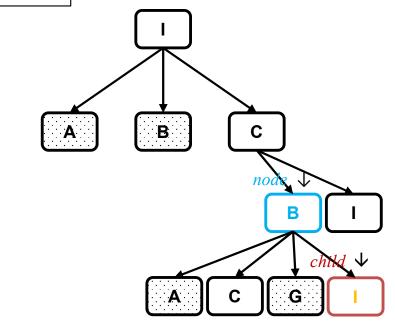
 $s \leftarrow child$.STATE

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

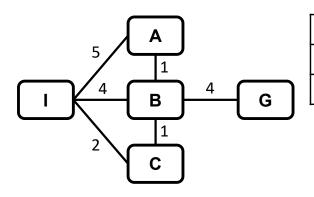
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	_	Α	В	C	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	В	I	I		
Frontier	Node	Α	G	В	Α		
Fr	f(Node)	6	7	7	7		
þ	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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

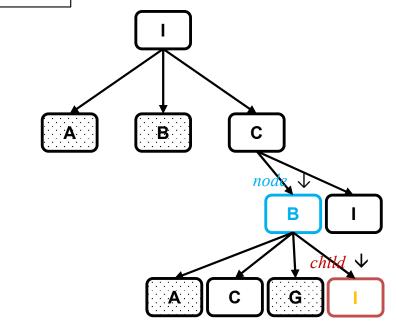
 $s \leftarrow child$.STATE

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

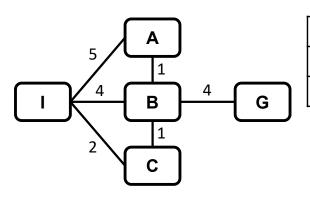
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







Straight-line distance to Goal state							
State	ı	Α	В	С	G		
h(State)	7	2	3	4	0		

INITIAL STATE: I GOAL STATE: G

er	Parent	В	В	Ι	-		
Frontier	Node	Α	G	В	Α		
Fr	f(Node)	6	7	7	7		
b	Parent		В	С	I	В	
Reached	Parent Key/State		B A	C B	C	B G	

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

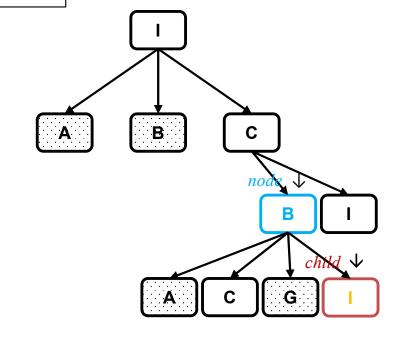
 $s \leftarrow child$.STATE

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

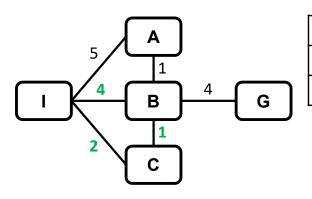
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	В	I	I		
Frontier	Node	Α	G	В	Α		
Fr	f(Node)	6	7	7	7		
þ	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	C	B G	

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

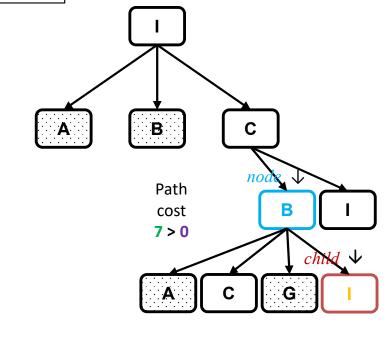
if s is not cached or child.PATH-COSD

ached[s].PATH-COST then

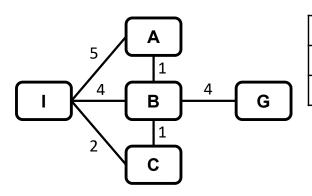
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

INITIAL STATE: I GOAL STATE: G

S.	Parent	В	В	I	I		
Frontier	Node	Α	G	В	Α		
Ŧ	f(Node)	6	7	7	7		
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

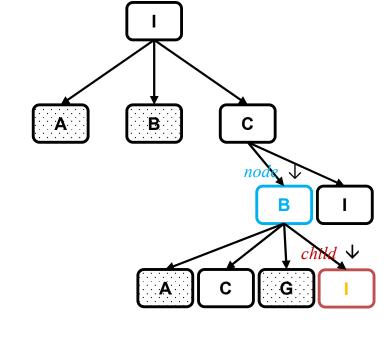
 $s \leftarrow child.STATE$

ached or *child*.PATH-COST

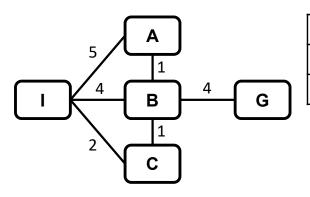
(ached[s].PATH-COST then

reache child add chid frontier

return failure







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

- G	Parent	В	В	I	I		
Frontier	Node	Α	G	В	Α		
구	f(Node)	6	7	7	7		
p	Parent		В	С	I	В	
Reached	Key/State	I	Α	В	С	G	
Re	Path cost	0	4	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

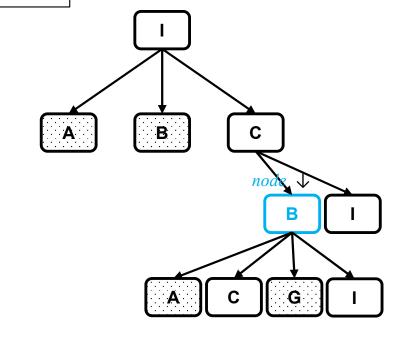
 $s \leftarrow child$.STATE

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

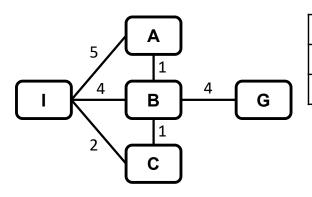
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	В	I	I		
Frontier	Node	Α	G	В	Α		
Fr	f(Node)	6	7	7	7		
þ	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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

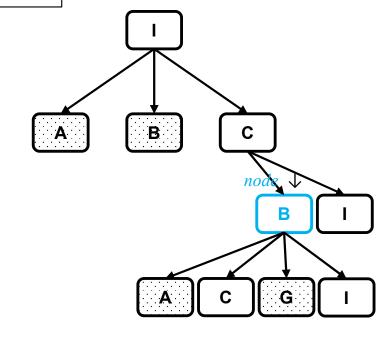
 $s \leftarrow child.STATE$

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

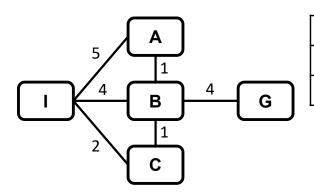
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

J.O.	Parent	В	I	I			
Frontier	Node	G	В	Α			
F	f(Node)	7	7	7			
p	Parent		В	С	ļ	В	
Reached	Key/State	I	Α	В	С	G	
Re	Path cost	0	4	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

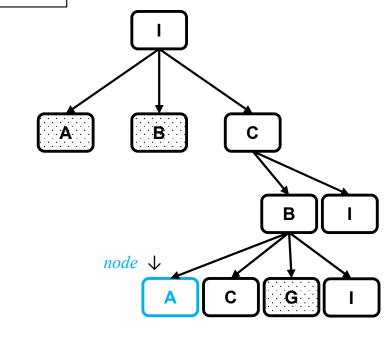
 $s \leftarrow child.STATE$

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

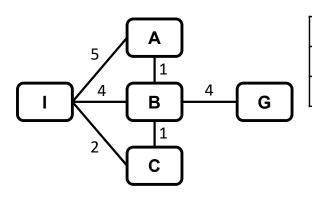
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

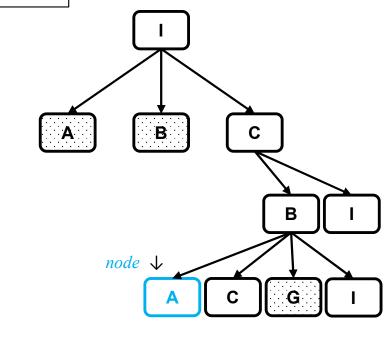
 $s \leftarrow child$.STATE

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

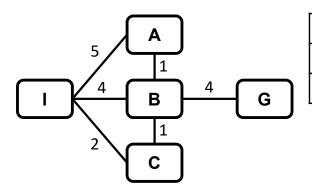
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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 FALSE

for each child in EXPAND(problem, node) do

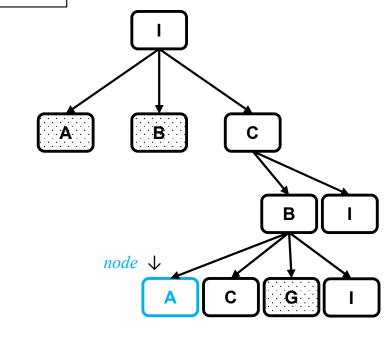
 $s \leftarrow child$.STATE

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

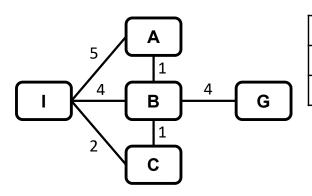
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	Ι	Ι			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
						_	
b	Parent		В	С	I	В	
Reached	Rarent Key/State	 I	A	В	C	B G	

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

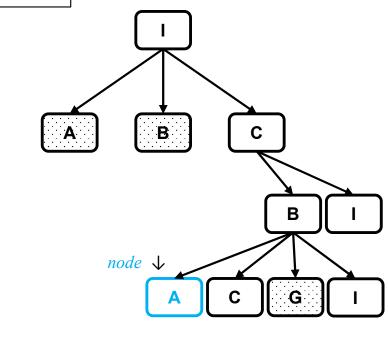
 $s \leftarrow child$.STATE

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

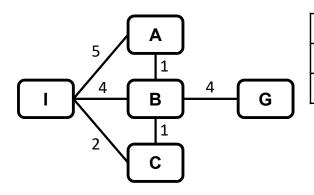
 $reached[s] \leftarrow child$

add *child* to *frontier*

return failure







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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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*

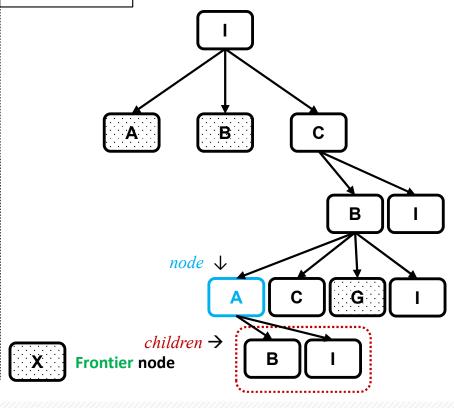
for each child in EXPAND(problem, node) do

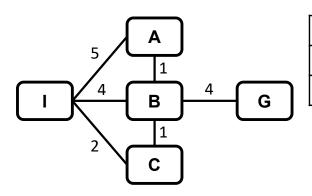
 $s \leftarrow child.STATE$

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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*

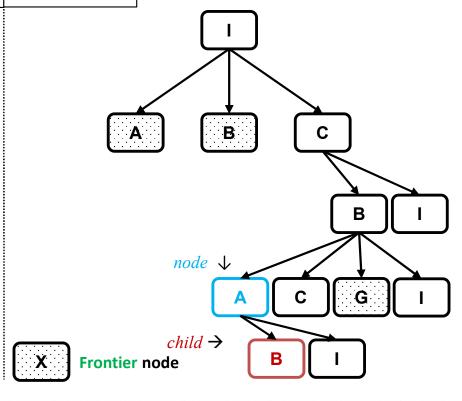
for each child in EXPAND(problem, node) do

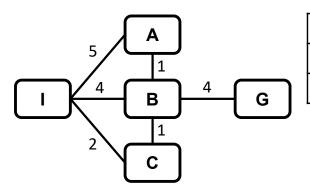
 $s \leftarrow child.STATE$

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

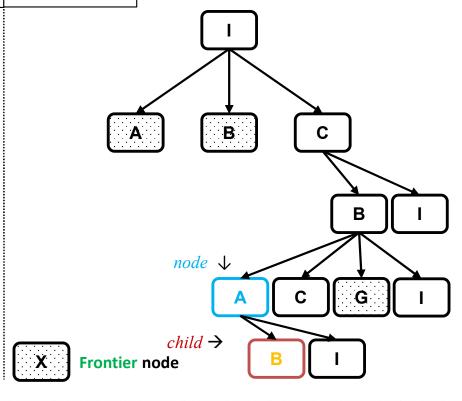
for each child in EXPAND(problem, node) do

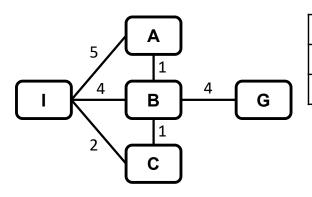
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

J.O.	Parent	В	I	I			
Frontier	Node	G	В	Α			
Ę	f(Node)	7	7	7			
p	Parent		В	С	ļ	В	
Reached	Key/State	I	Α	В	С	G	
Re	Path cost	0	4	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

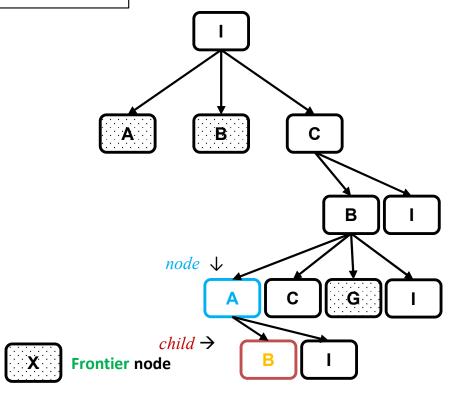
for each child in EXPAND(problem, node) do

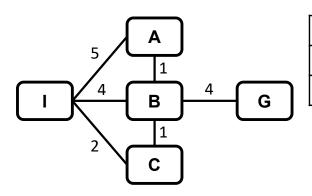
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

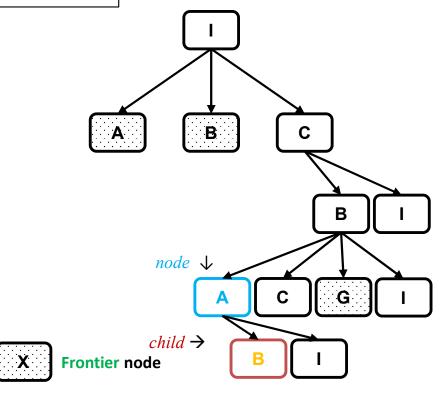
for each child in EXPAND(problem, node) do

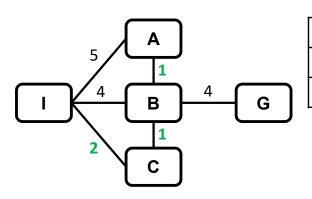
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	Ι	I			
Frontier	Node	G	В	Α			
Ţ	f(Node)	7	7	7			
p	Parent		В	С	ļ	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

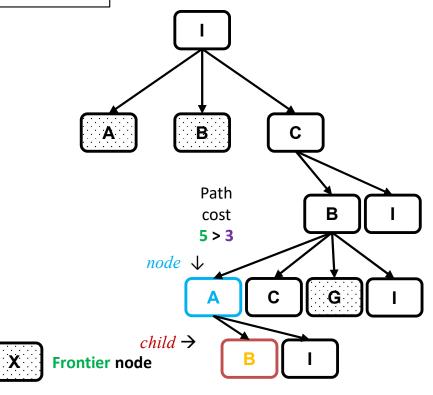
 $s \leftarrow child$.STATE

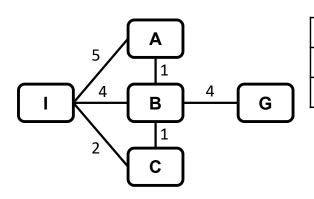
if s is not sached or child.PATH-COST

(ached[s].PATH-COST then

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

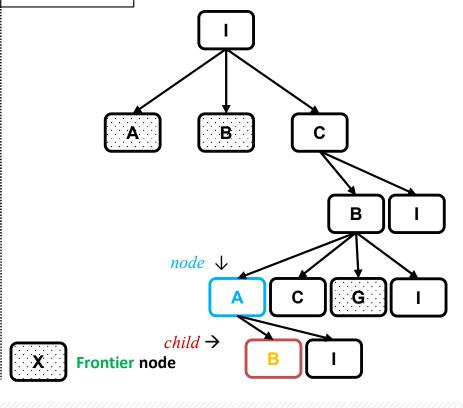
for each child in EXPAND(problem, node) do

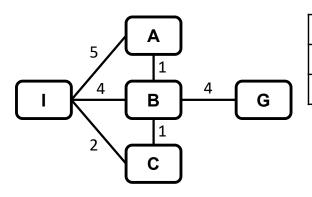
 $s \leftarrow child$.STATE

if s is not sached or child.PATH-COST

(ached[s].PATH-COST then

reache child add child frontier





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	I	Α	В	С	G	
Re	Path cost	0	4	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*

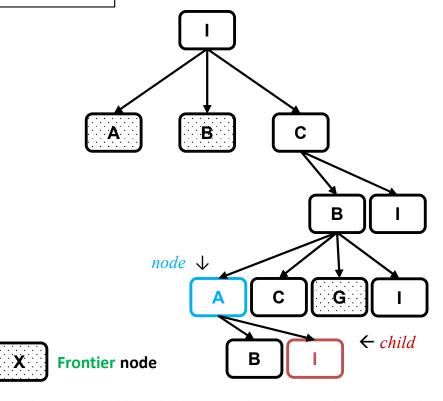
for each child in EXPAND(problem, node) do

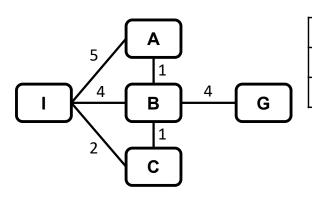
 $s \leftarrow child.STATE$

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
þ	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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

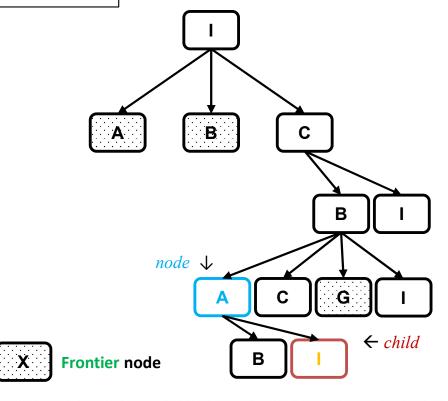
for each child in EXPAND(problem, node) do

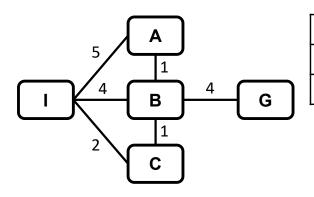
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

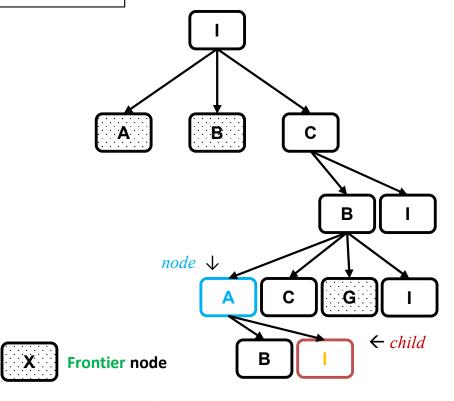
for each child in EXPAND(problem, node) do

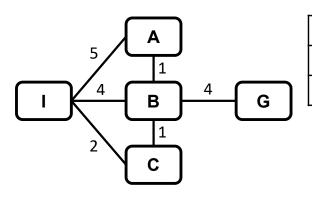
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

- G	Parent	В	I	I			
Frontier	Node	G	В	Α			
구	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
l &	Path cost	0	4	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

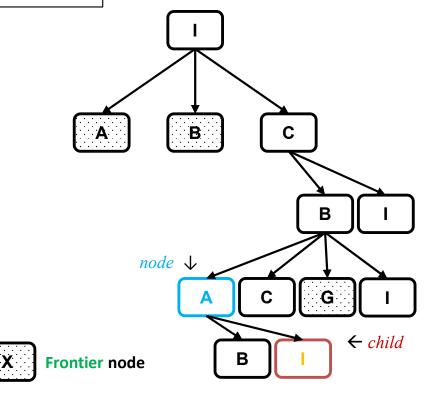
for each child in EXPAND(problem, node) do

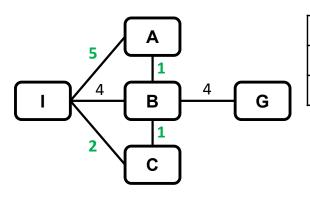
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	Ι			
Frontier	Node	G	В	Α			
ᇁ	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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

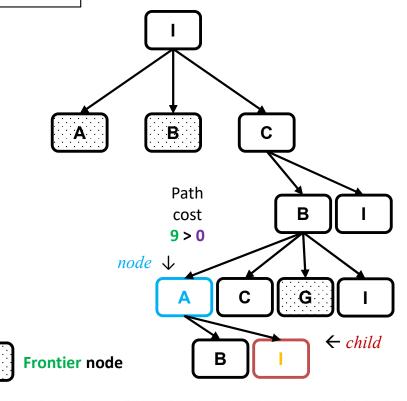
 $s \leftarrow child$.STATE

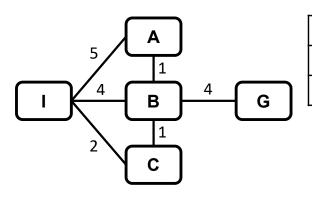
if s is not sached or child.PATH-COST

(ached[s].PATH-COST then

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

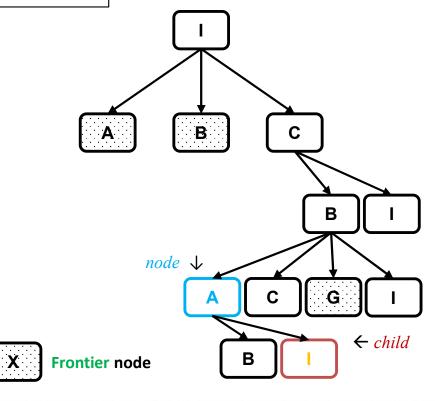
for each child in EXPAND(problem, node) do

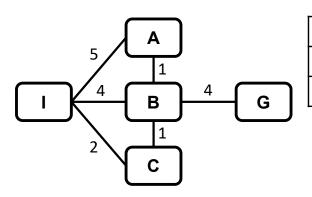
 $s \leftarrow child$.STATE

if s is not eached or child.PATH-COSD

(ached[s].PATH-COST then

reache child add child frontier





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

INITIAL STATE: I GOAL STATE: G

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
þ	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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

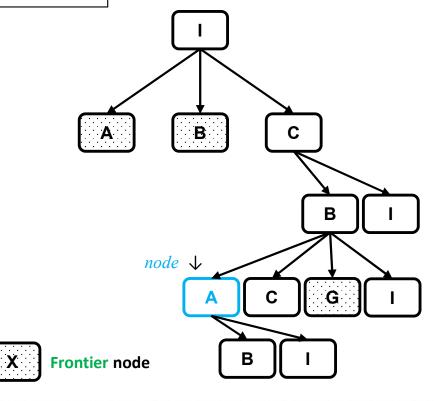
for each child in EXPAND(problem, node) do

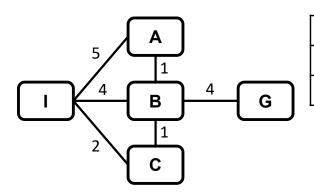
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	В	I	I			
Frontier	Node	G	В	Α			
Fr	f(Node)	7	7	7			
p	Parent		В	С	I	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

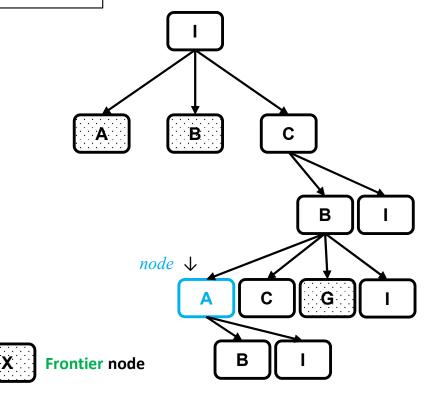
for each child in EXPAND(problem, node) do

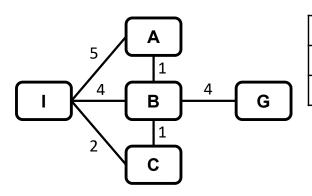
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





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

er	Parent	I	I				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
p	Parent		В	С	ļ	В	
Reached	Key/State	ı	Α	В	С	G	
Re	Path cost	0	4	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

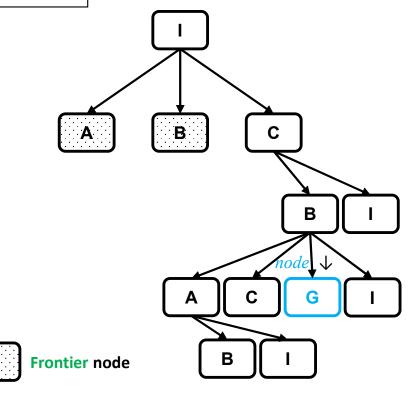
for each child in EXPAND(problem, node) do

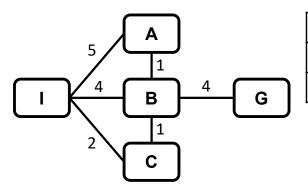
 $s \leftarrow child.STATE$

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





Straight-line distance to Goal state						
State	1	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

er	Parent	Ι	Ι				
Frontier	Node	В	Α				
Fr	f(Node)	7	7				
b	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	C	B G	

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

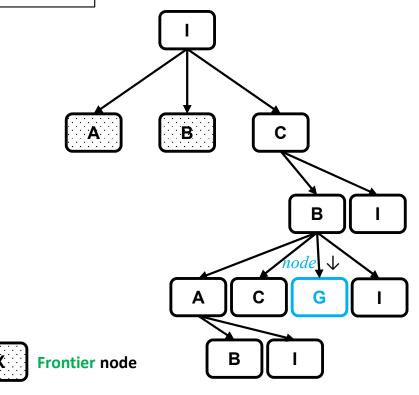
for each child in EXPAND(problem, node) do

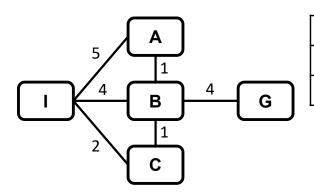
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

e.	Parent	I	I				
Frontier	Node	В	Α				
ᇁ	f(Node)	7	7				
b	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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!

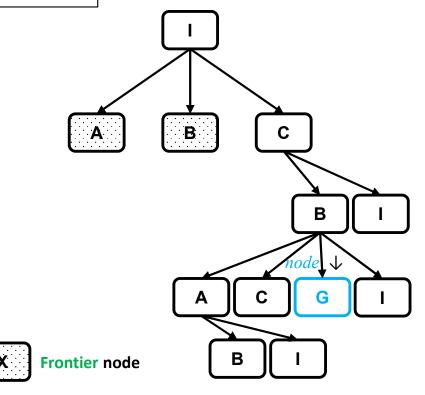
for each child in EXPAND(problem, node) do

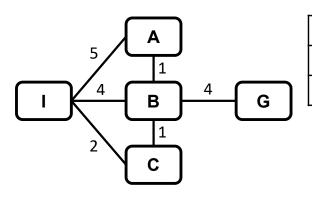
 $s \leftarrow child$.STATE

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

 $reached[s] \leftarrow child$

add *child* to *frontier*





Straight-line distance to Goal state						
State	-	Α	В	С	G	
h(State)	7	2	3	4	0	

INITIAL STATE: I GOAL STATE: G

e.	Parent	I	I				
Frontier	Node	В	Α				
ᇁ	f(Node)	7	7				
b	Parent		В	С	I	В	
Reached	Parent Key/State	 I	B A	C B	l C	B G	

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*

