A* Algorithm: Evaluation Function

Calculate / obtain:

$$f(n) = \mathbf{g(State_n)} + \mathbf{h(State_n)}$$

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
    node ← NODE(STATE=problem.INITIAL)
    frontier ← a priority queue ordered by 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

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

f(n) = g(State$_n$) + h(State$_n$)

Best-First Search: A* Pseudocode

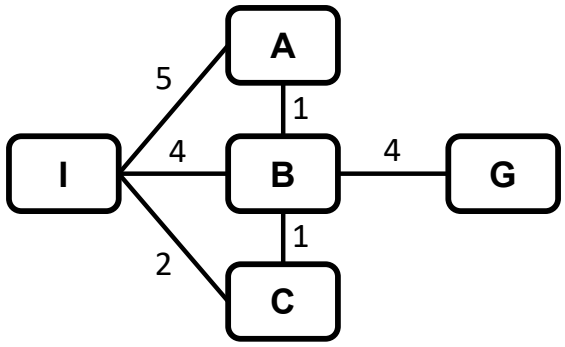
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
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

f(n) = $g(State_n)$ + $h(State_n)$

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

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

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

Reached								
	Parent							
	Key/State							
	Path cost							

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

s ← *child*.STATE

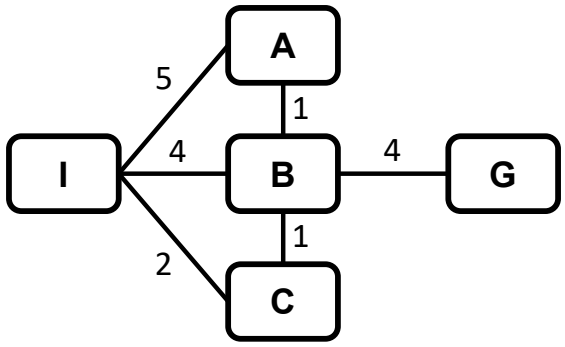
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] ← *child*

add *child* to *frontier*

return *failure*

A* Search: Example



g(state)
Path cost

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

INITIAL STATE: I
GOAL STATE: G

h(state)
Heuristic

Frontier								
	Parent							
	Node							
	f(Node)							

Reached								
	Parent							
	Key/State							
	Path cost							

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

      s ← *child*.STATE

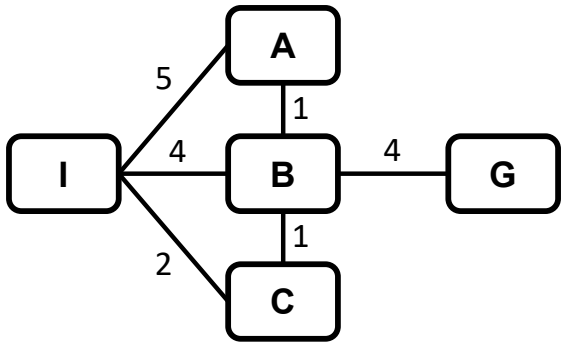
      if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

s ← *child*.STATE

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

reached[*s*] ← *child*

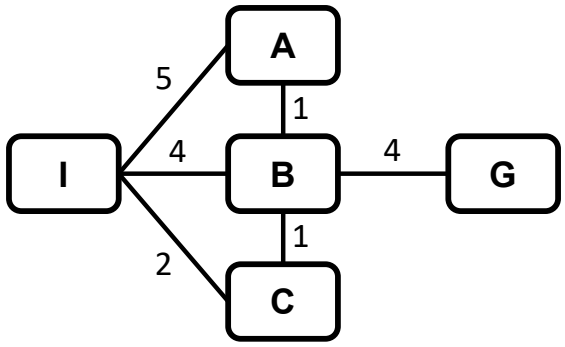
add *child* to *frontier*

return *failure*

node → I

X **Frontier node**

A* Search: Example



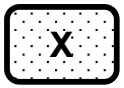
Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent							
	Node							
	f(Node)							

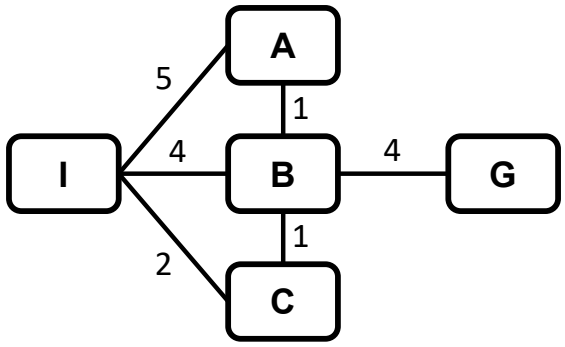
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



Frontier node

A* Search: Example



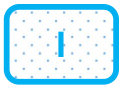
Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

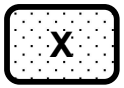
Frontier								
	Parent	----						
	Node	I						
	f(Node)	7						

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

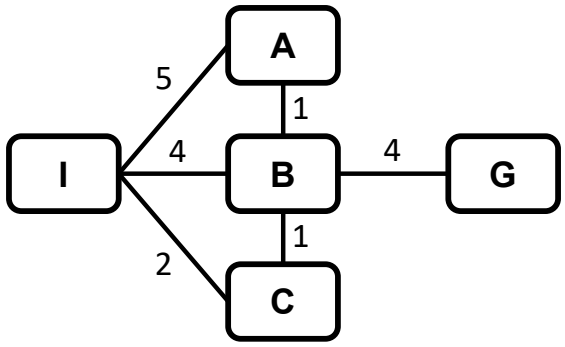


f(I) = g(I) + h(I) = **0**+**7** = **7**



Frontier node

A* Search: Example



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

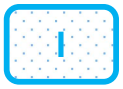
INITIAL STATE: I
GOAL STATE: G

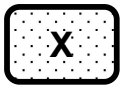
Frontier								
	Parent	----						
	Node	I						
	f(Node)	7						

Reached								
	Parent							
	Key/State							
	Path cost							

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

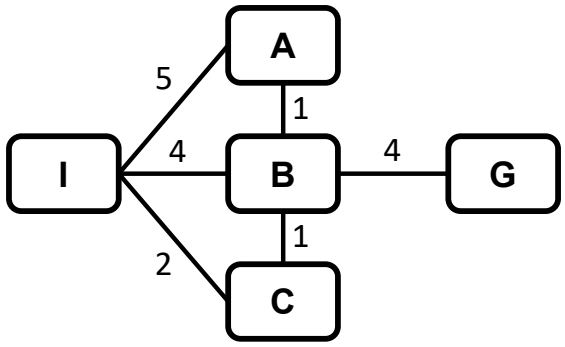
```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE)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





Frontier node

A* Search: Example



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

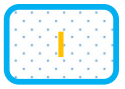
INITIAL STATE: I
GOAL STATE: G

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

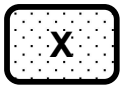
Reached								
	Parent	----						
	Key/State	I						
	Path cost	0						

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

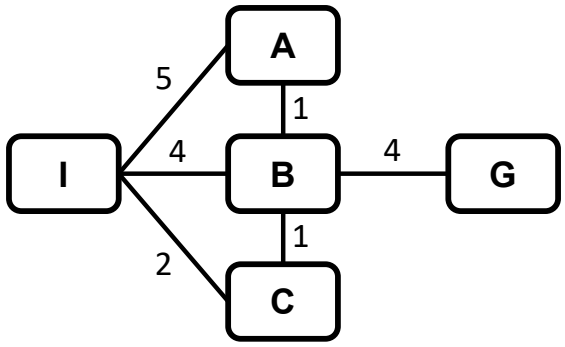


Path cost **0**



Frontier node

A* Search: Example



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

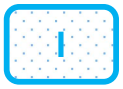
INITIAL STATE: I
GOAL STATE: G

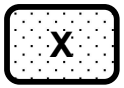
Frontier								
	Parent	----						
	Node	I						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

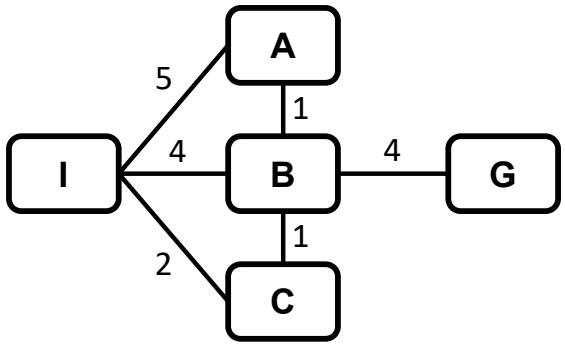
```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE)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





Frontier node

A* Search: Example



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

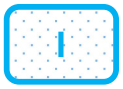
INITIAL STATE: I
GOAL STATE: G

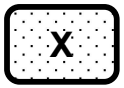
Frontier								
	Parent	----						
	Node	I						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

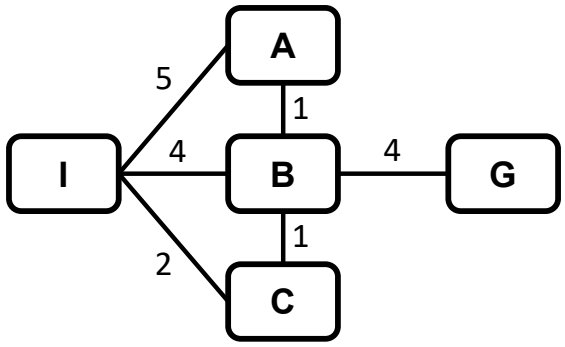
```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE)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





Frontier node

A* Search: Example



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

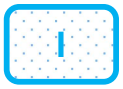
INITIAL STATE: I
GOAL STATE: G

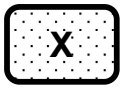
Frontier								
	Parent	----						
	Node	I						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

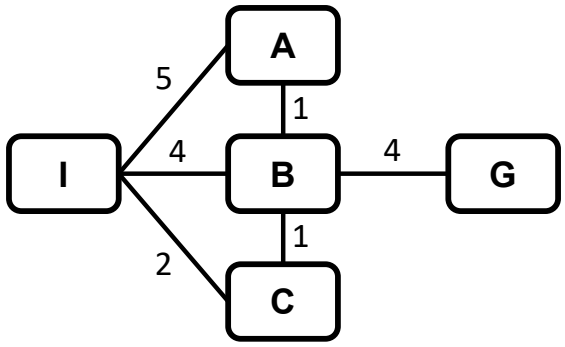
```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do          NOT EMPTY!
    node ← POP(frontier)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





Frontier node

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

s ← *child*.STATE

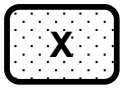
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] ← *child*

add *child* to *frontier*

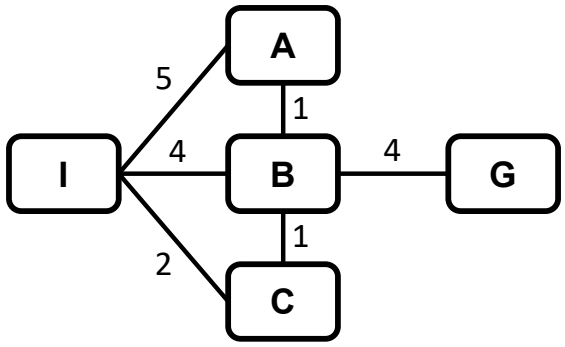
return *failure*

node →



Frontier node

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

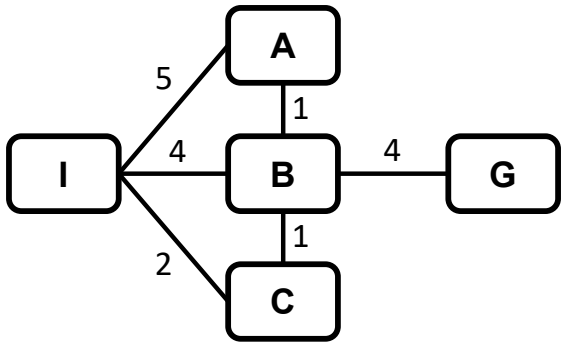
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node → I

X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

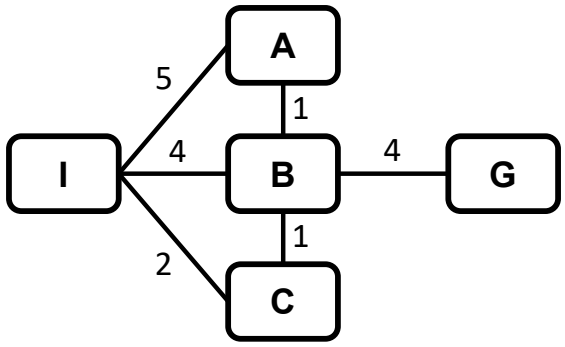
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node → I

X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

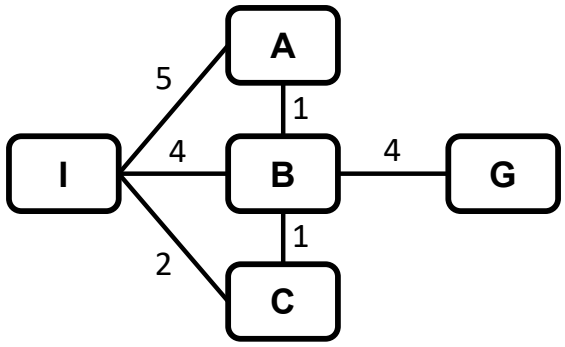
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node → I

X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

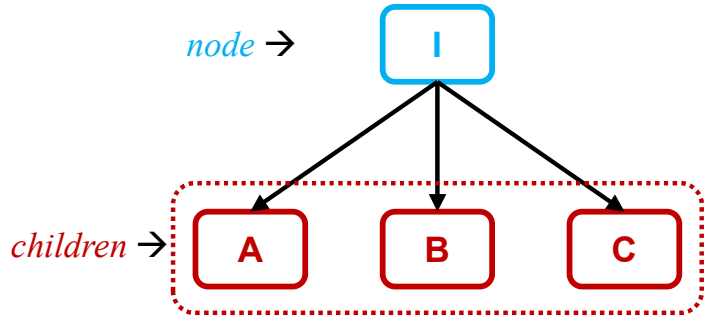
s ← *child*.STATE

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

reached[*s*] ← *child*

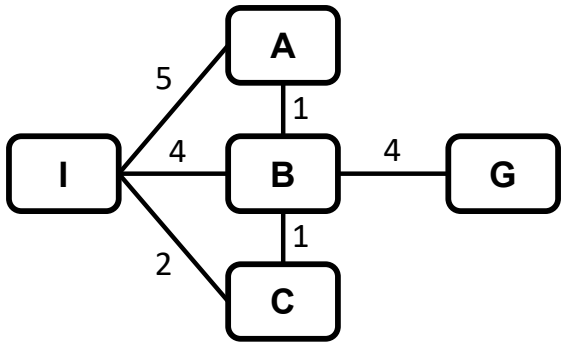
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

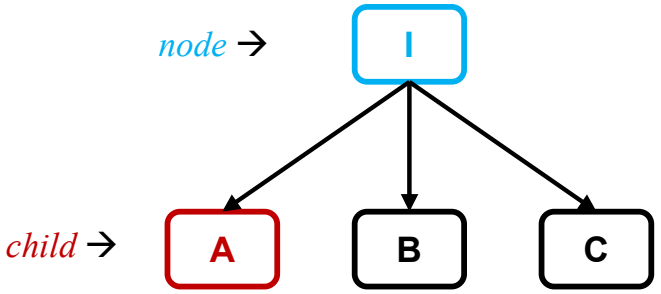
s ← *child*.STATE

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

reached[*s*] ← *child*

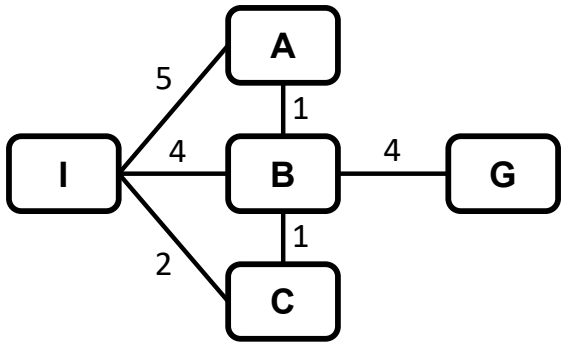
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

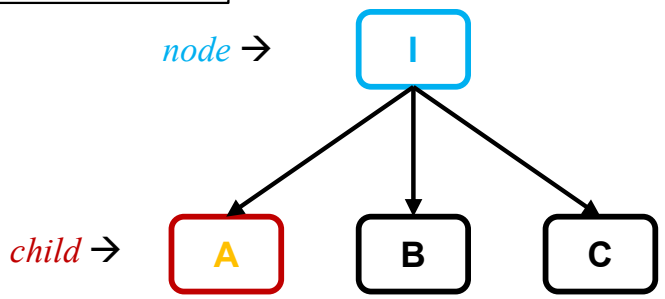
s ← *child*.STATE

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

reached[*s*] ← *child*

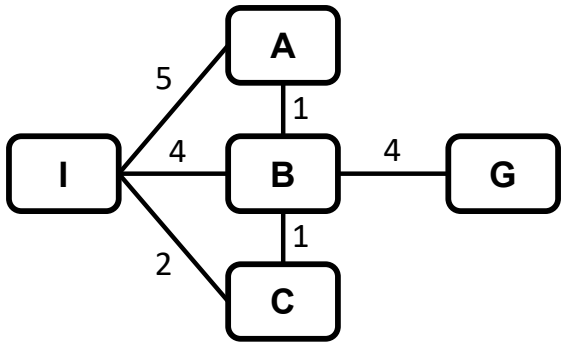
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

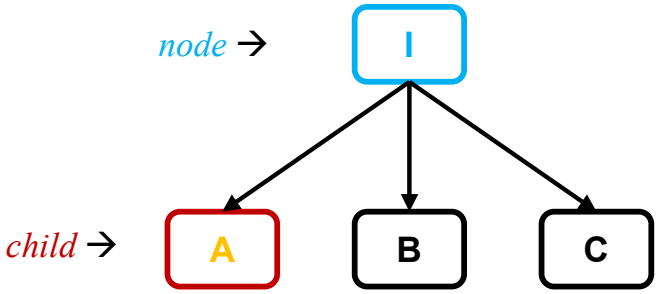
      s ← *child*.STATE

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

        reached[*s*] ← *child*

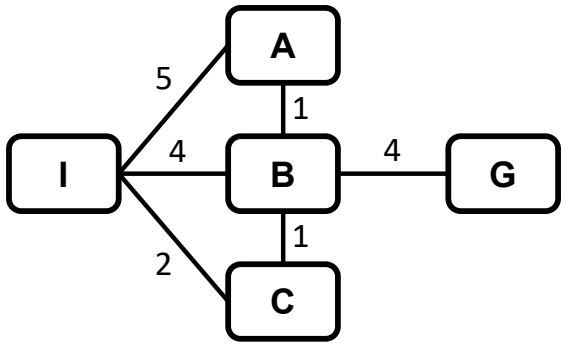
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

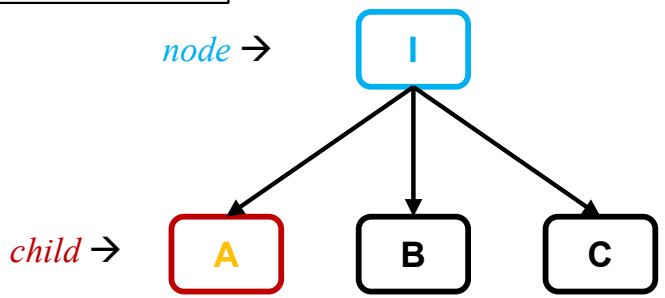
s ← *child*.STATE

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

reached[*s*] ← *child*

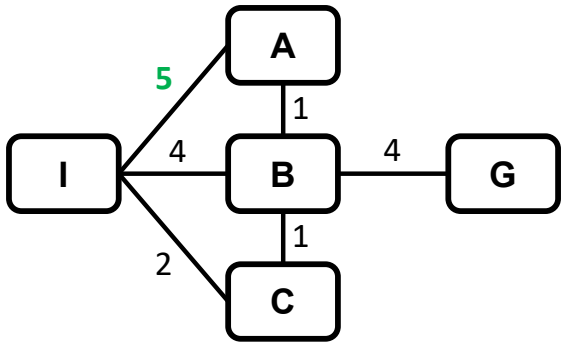
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent							
	Node							
	f(Node)							

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

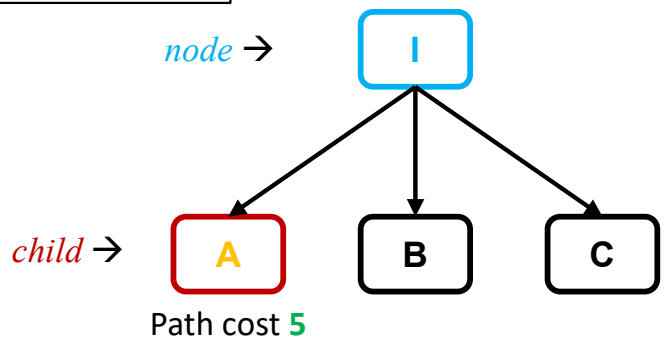
      s ← *child*.STATE

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

        reached[*s*] ← *child*

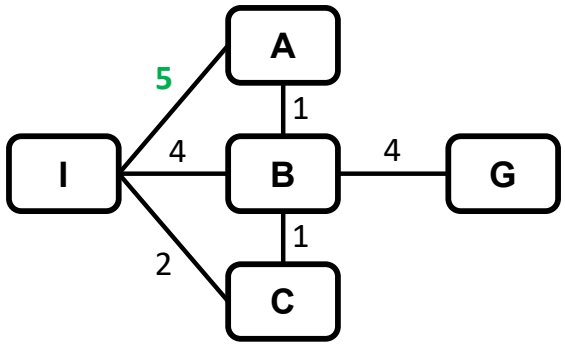
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I						
	Node	A						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

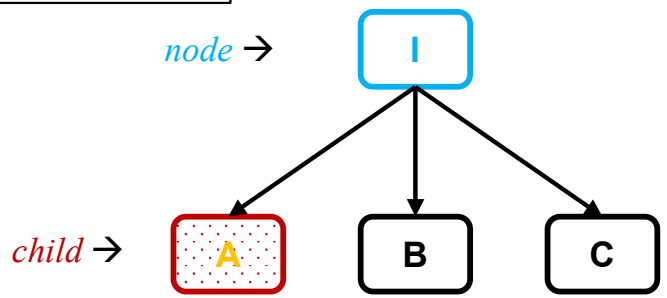
s ← *child*.STATE

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

reached[*s*] ← *child*

add *child* to *frontier*

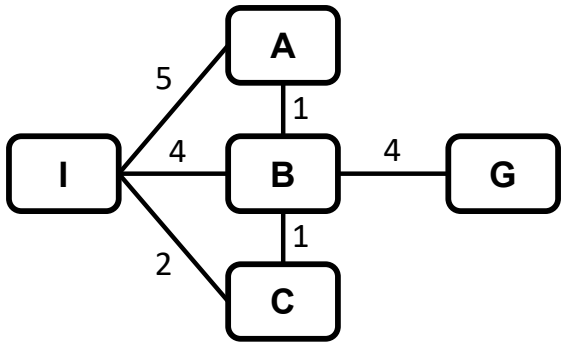
return *failure*



f(A) = g(A) +
h(A) = 5+2 = 7

X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I						
	Node	A						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

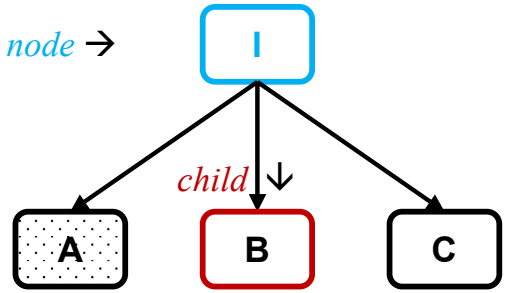
s ← *child*.STATE

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

reached[*s*] ← *child*

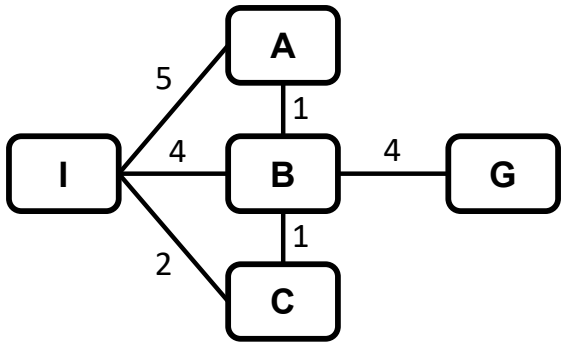
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I						
	Node	A						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

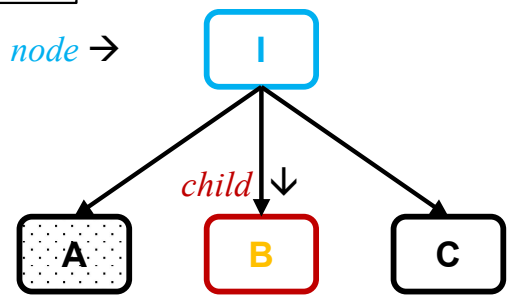
      s ← *child*.STATE

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

        reached[*s*] ← *child*

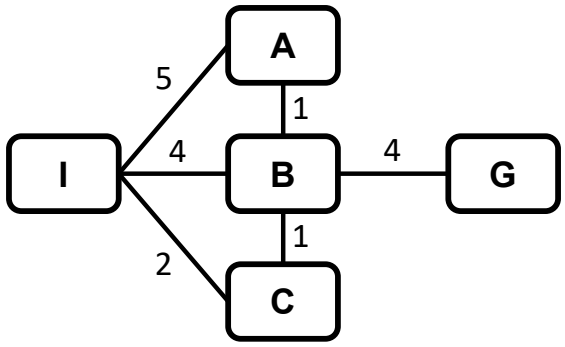
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I						
	Node	A						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

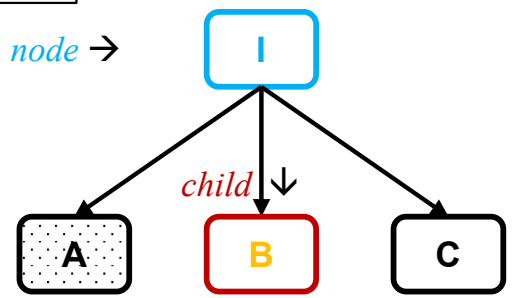
      s ← *child*.STATE

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

        reached[*s*] ← *child*

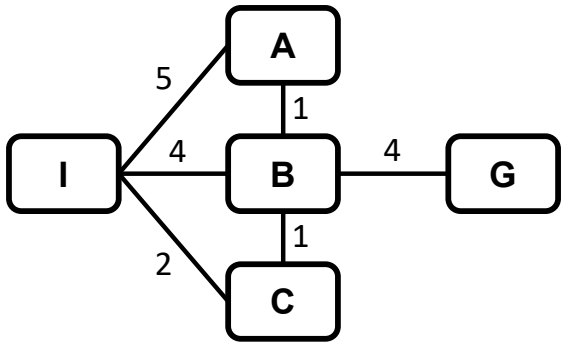
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I						
	Node	A						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

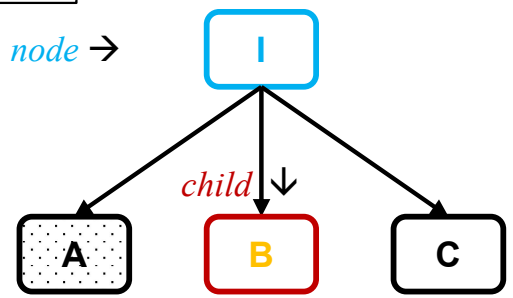
s ← *child*.STATE

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

reached[*s*] ← *child*

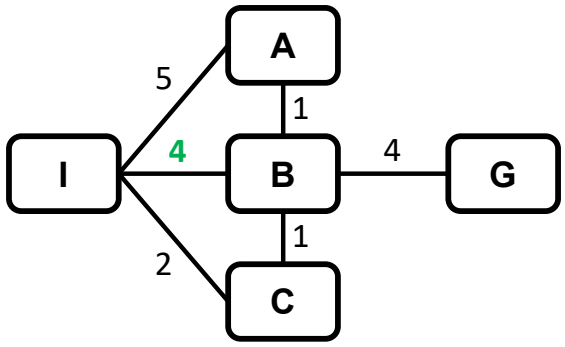
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I						
	Node	A						
	f(Node)	7						

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

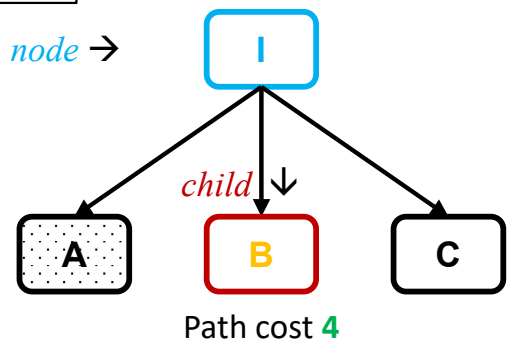
s ← *child*.STATE

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

reached[*s*] ← *child*

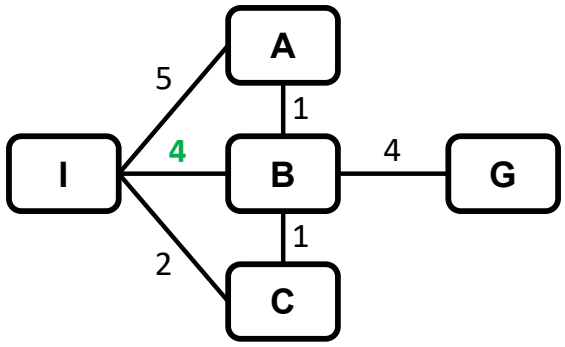
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

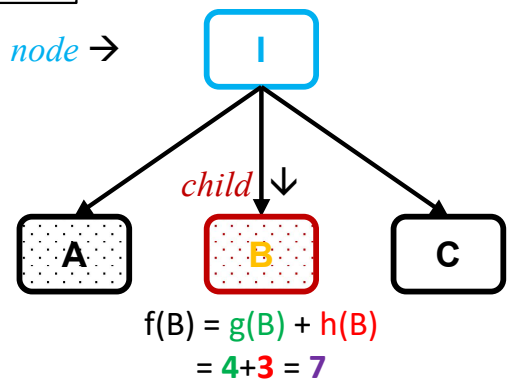
INITIAL STATE: I
GOAL STATE: G

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

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

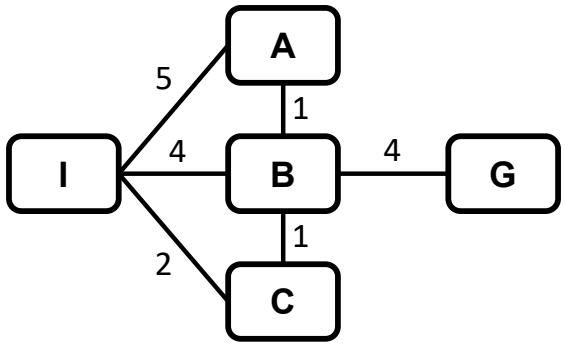
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

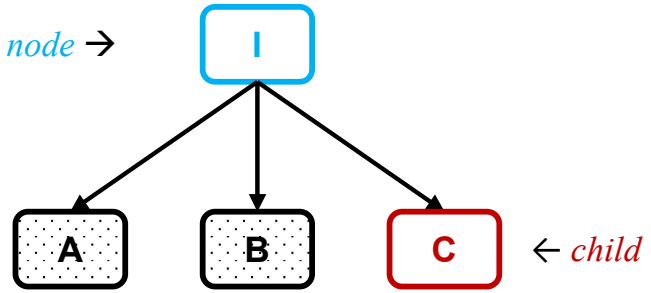
INITIAL STATE: I
GOAL STATE: G

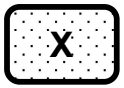
Frontier								
	Parent	I	I					
	Node	B	A					
	f(Node)	7	7					

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

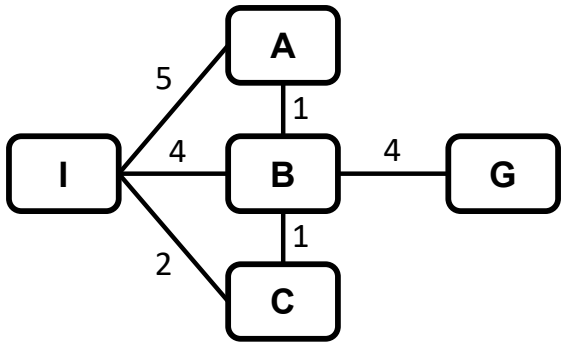
```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE)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





Frontier node

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

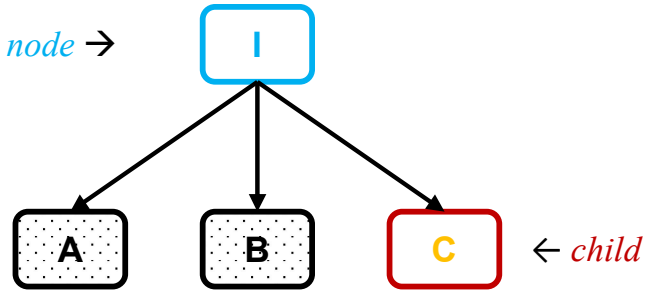
s ← *child*.STATE

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

reached[*s*] ← *child*

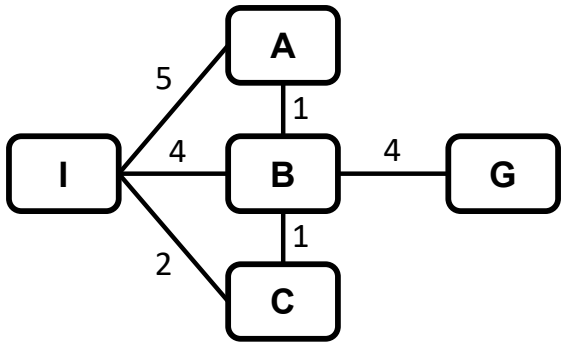
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

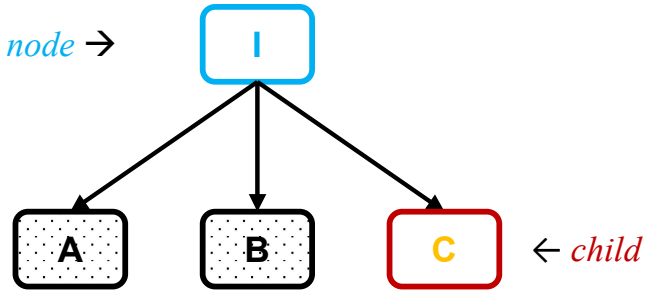
s ← *child*.STATE

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

reached[*s*] ← *child*

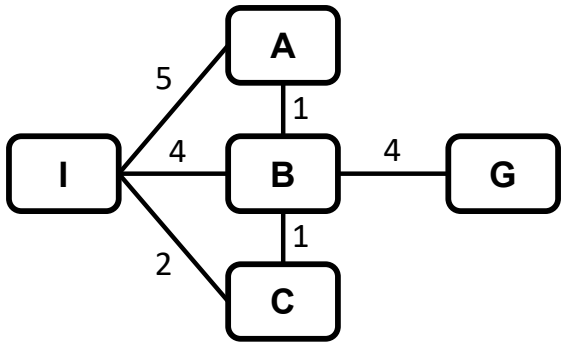
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

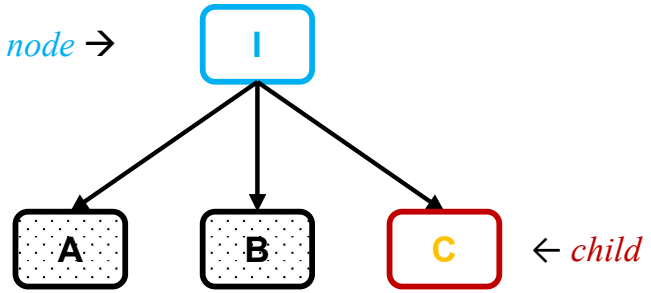
s ← *child*.STATE

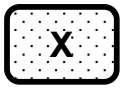
if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] ← *child*

add *child* to *frontier*

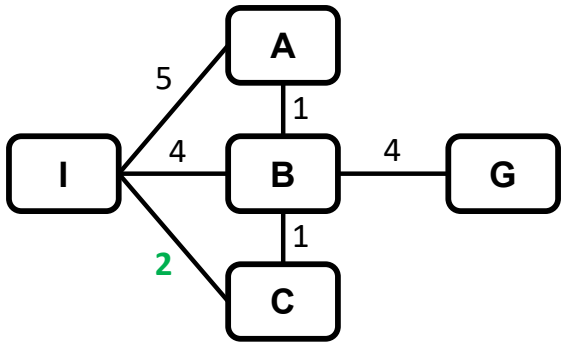
return *failure*





Frontier node

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

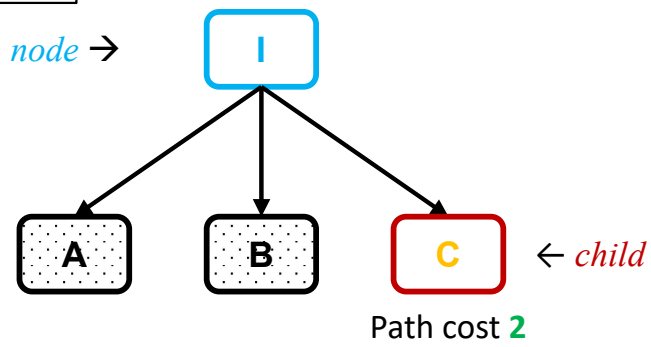
s ← *child*.STATE

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

reached[*s*] ← *child*

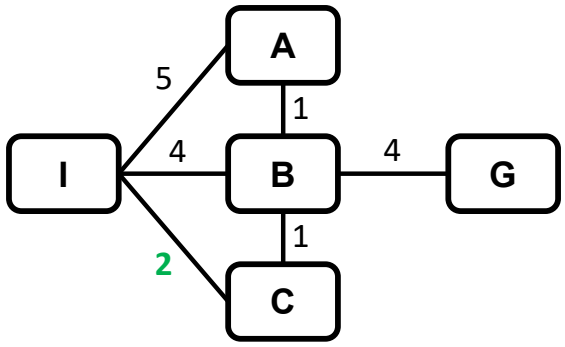
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

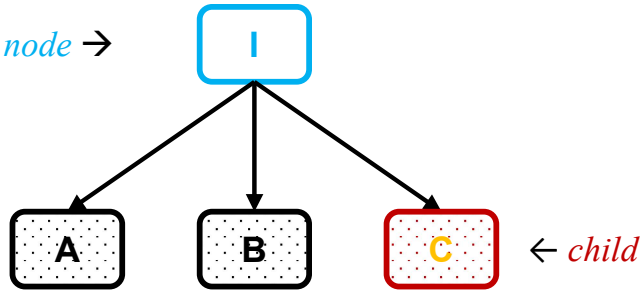
INITIAL STATE: I
GOAL STATE: G

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

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

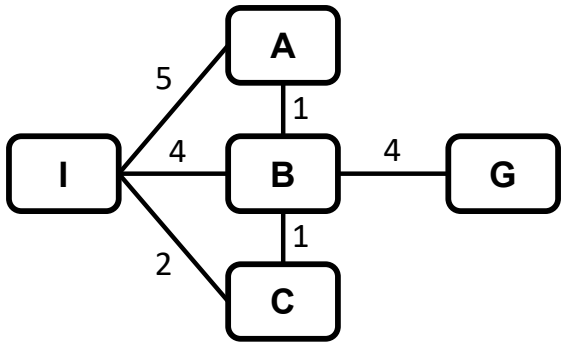
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

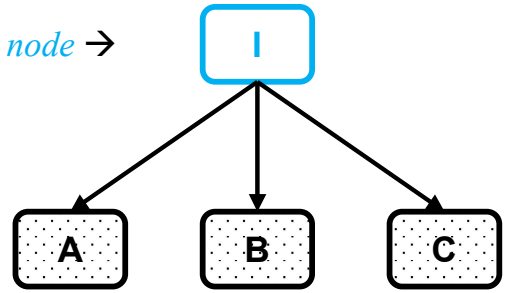
INITIAL STATE: I
GOAL STATE: G

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

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

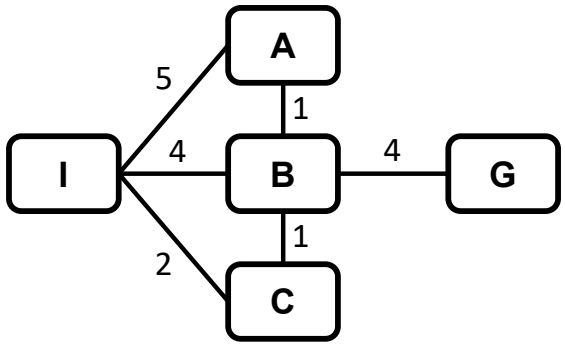
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

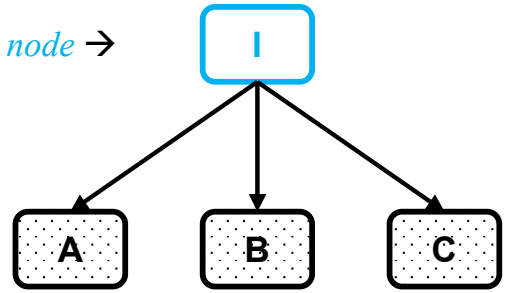
INITIAL STATE: I
GOAL STATE: G

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

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

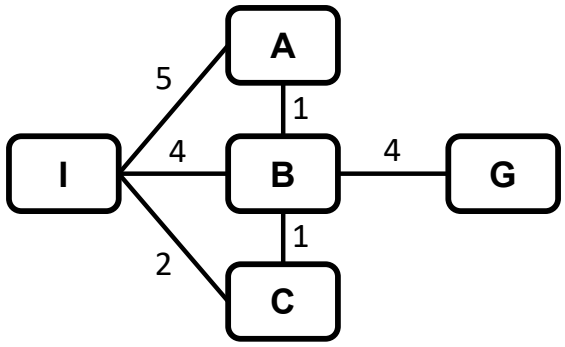
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

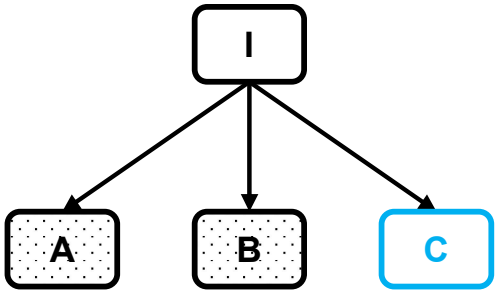
INITIAL STATE: I
GOAL STATE: G

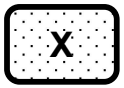
Frontier								
	Parent	I	I					
	Node	B	A					
	f(Node)	7	7					

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

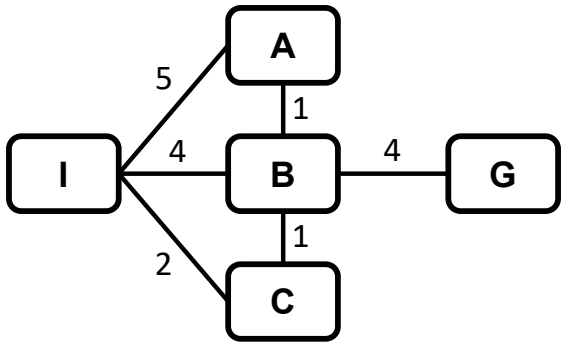
```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE)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





Frontier node

A* Search: Example



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

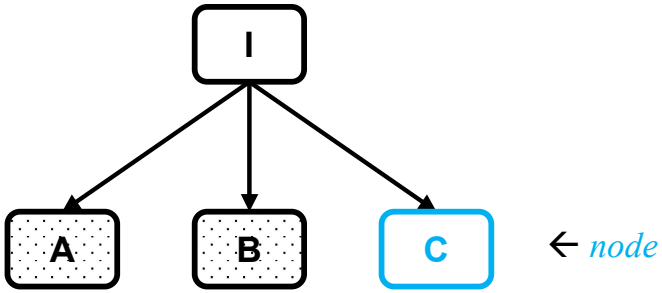
INITIAL STATE: I
GOAL STATE: G

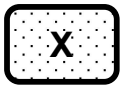
Frontier								
	Parent	I	I					
	Node	B	A					
	f(Node)	7	7					

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

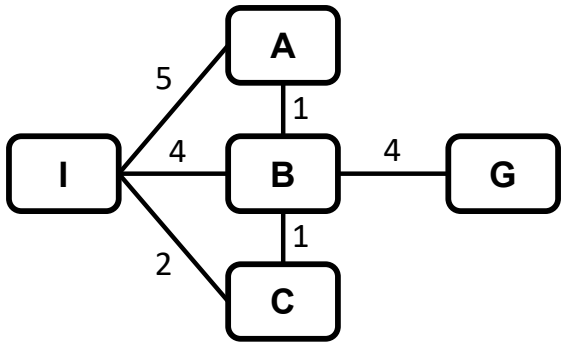
```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE)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





Frontier node

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

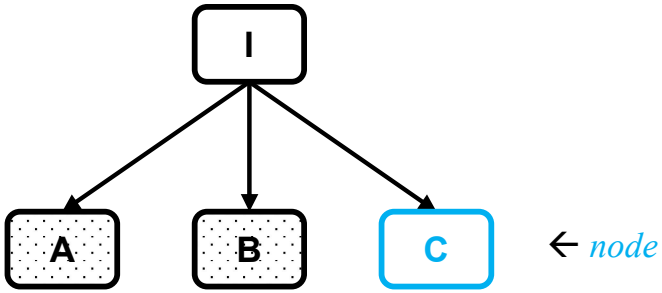
s ← *child*.STATE

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

reached[*s*] ← *child*

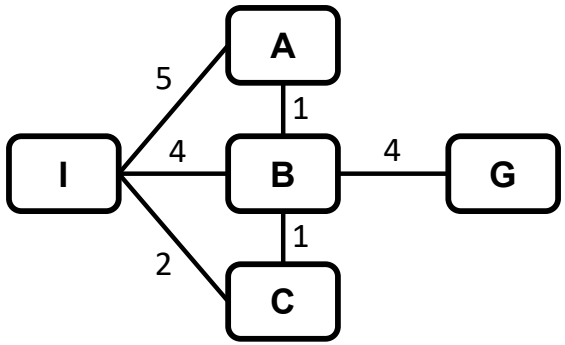
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

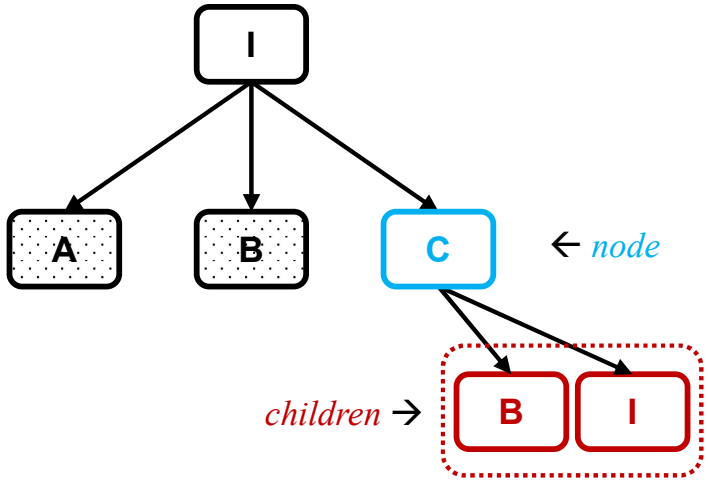
      s ← *child*.STATE

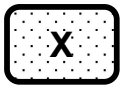
      if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

        reached[*s*] ← *child*

        add *child* to *frontier*

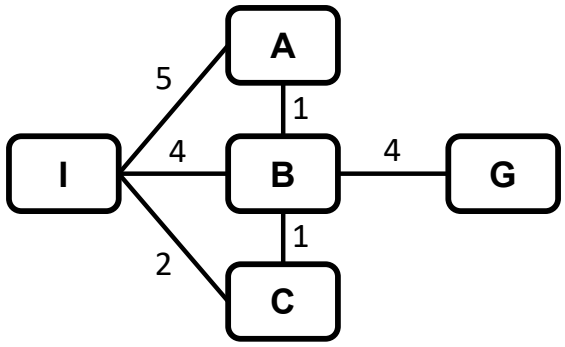
return *failure*





Frontier node

A* Search: Example



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

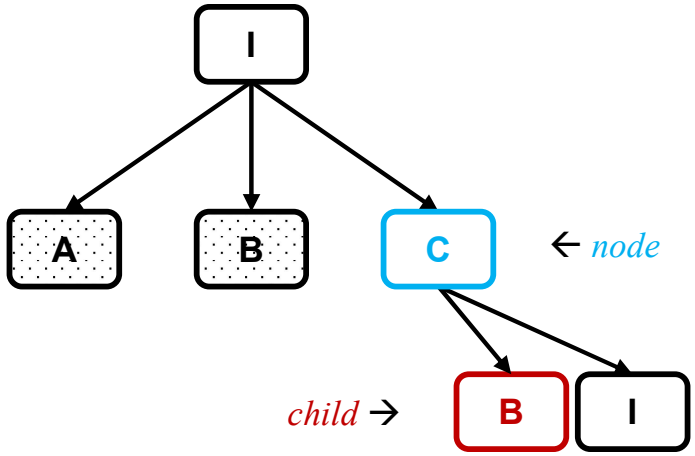
INITIAL STATE: I
GOAL STATE: G

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

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

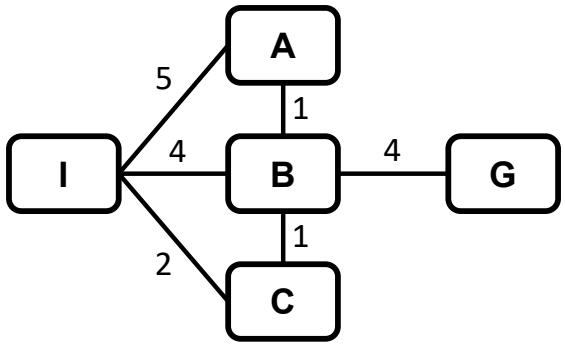
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

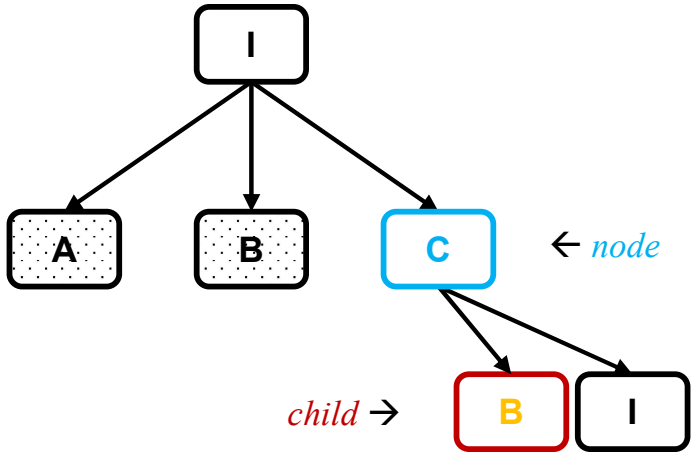
s ← *child*.STATE

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

reached[*s*] ← *child*

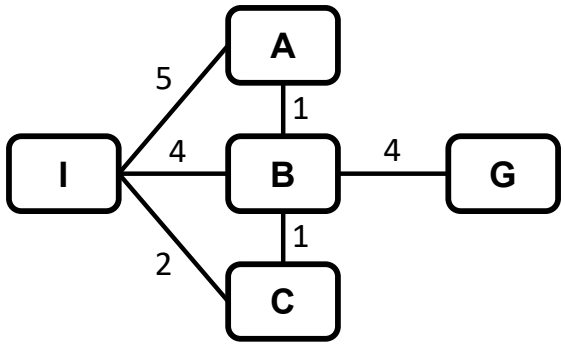
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

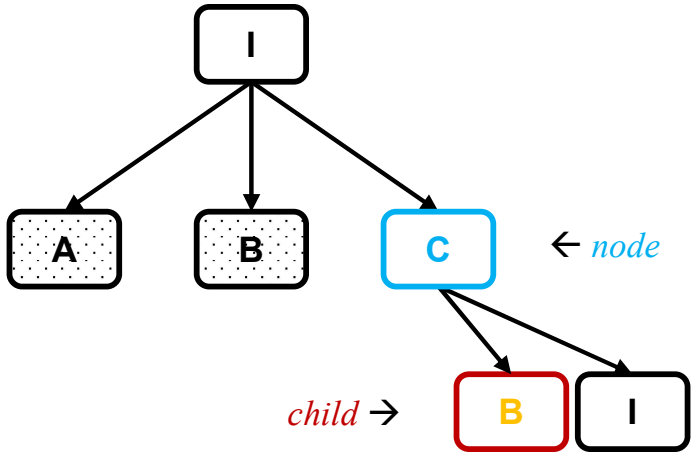
      s ← *child*.STATE

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

        reached[*s*] ← *child*

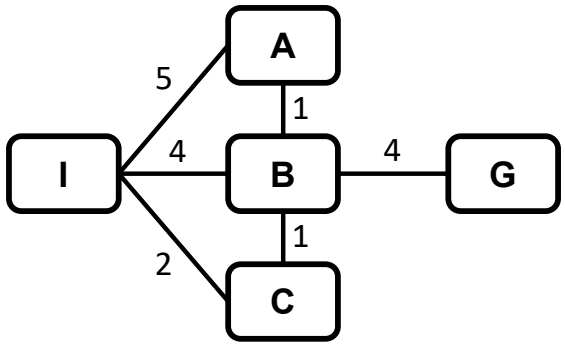
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

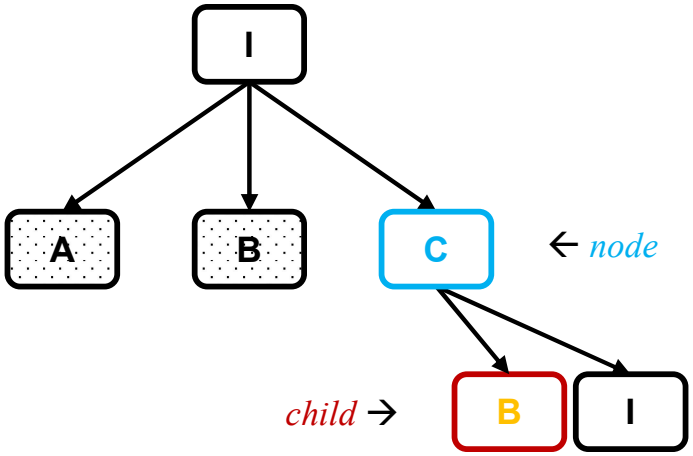
s ← *child*.STATE

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

reached[*s*] ← *child*

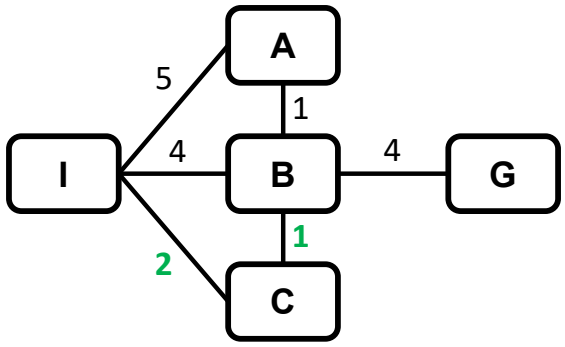
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

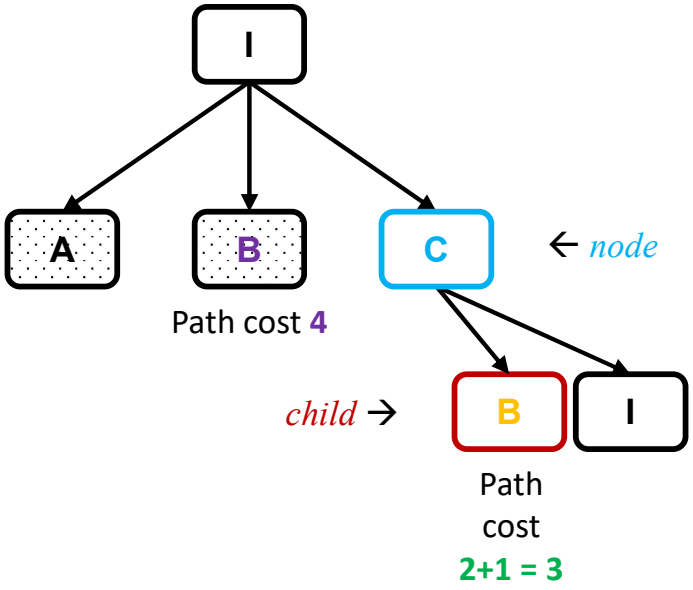
s ← *child*.STATE

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

reached[*s*] ← *child*

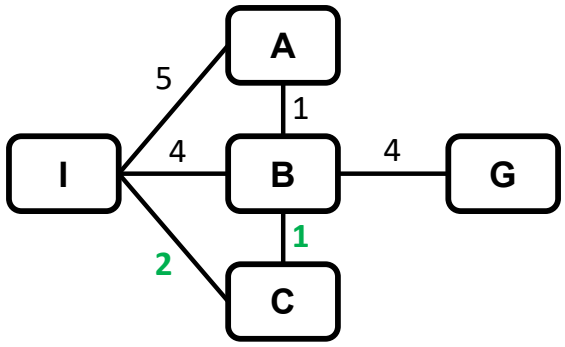
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

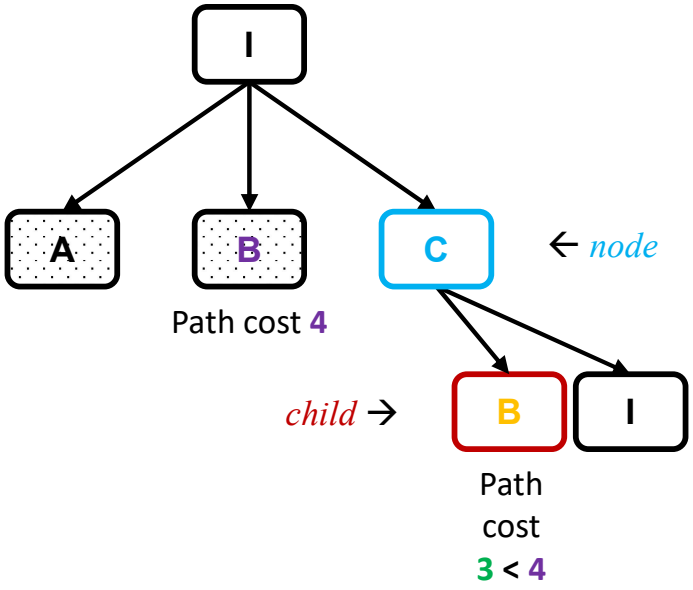
INITIAL STATE: I
GOAL STATE: G

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

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

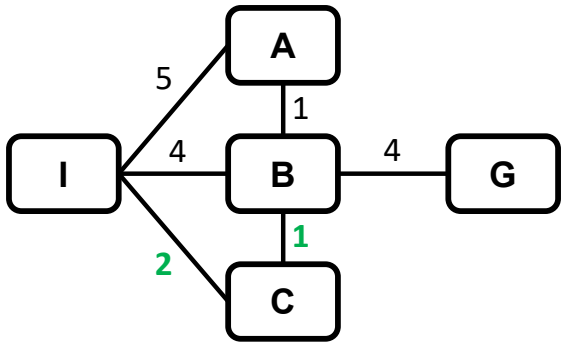
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

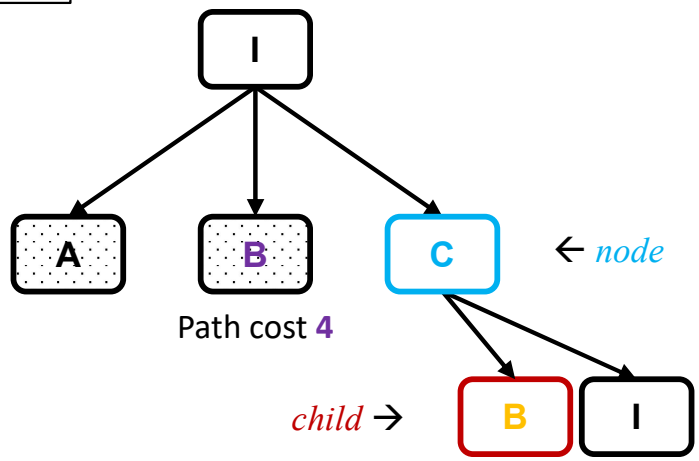
s ← *child*.STATE

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

reached[*s*] ← *child*

add *child* to *frontier*

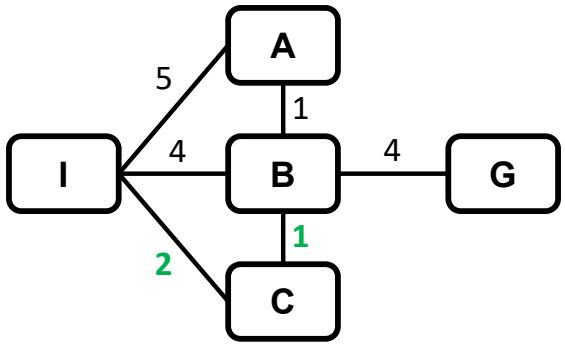
return *failure*



Corresponding Reached entry needs to be replaced

X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

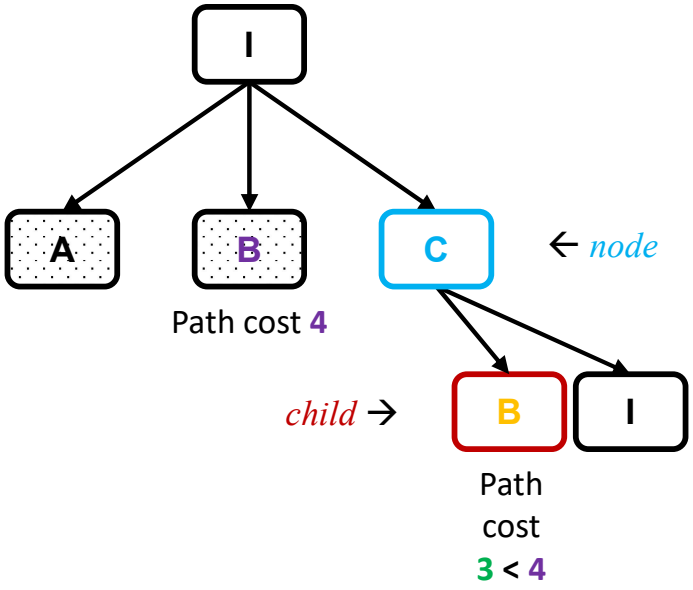
s ← *child*.STATE

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

reached[*s*] ← *child*

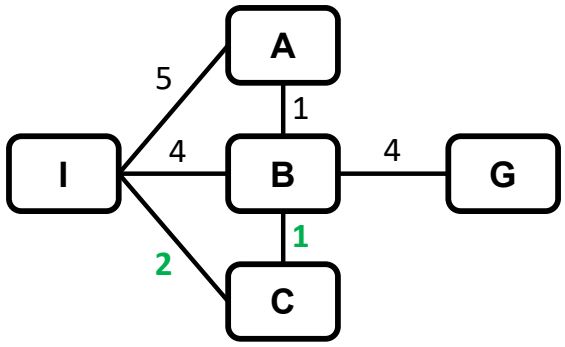
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

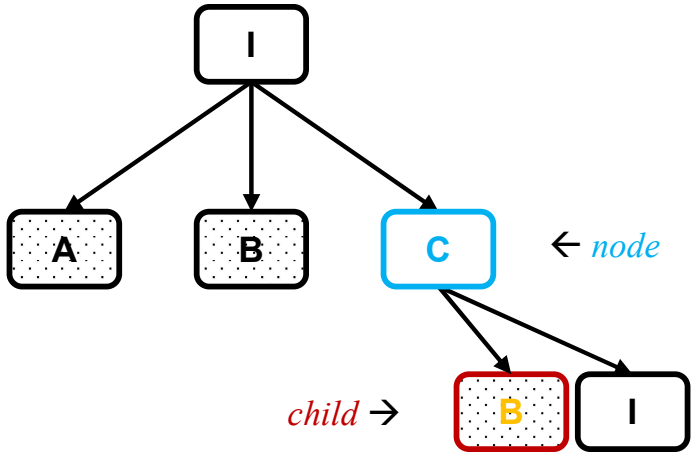
INITIAL STATE: I
GOAL STATE: G

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

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

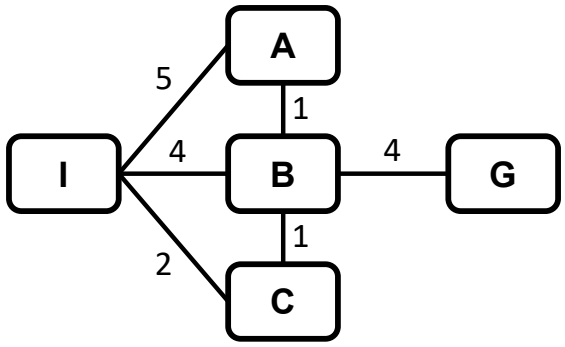
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

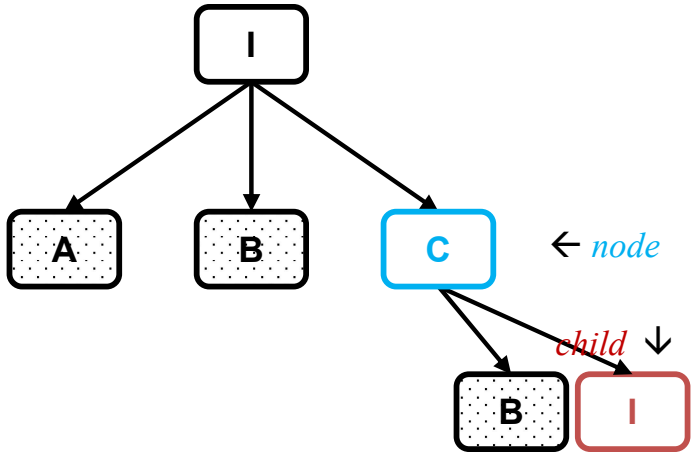
INITIAL STATE: I
GOAL STATE: G

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

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

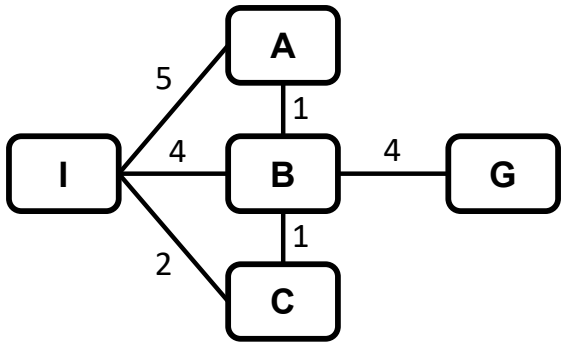
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

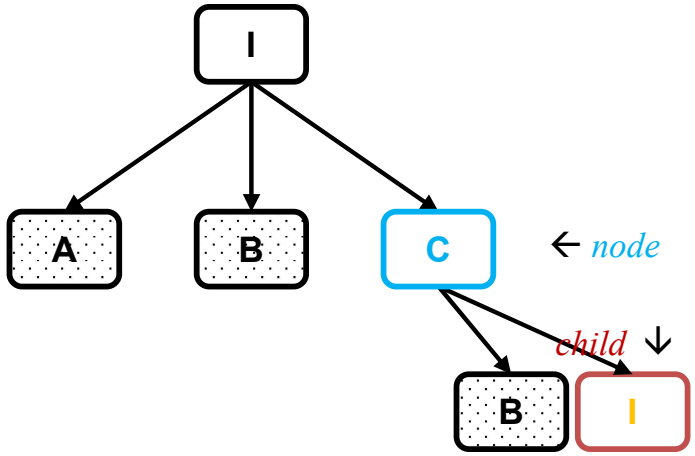
s ← *child*.STATE

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

reached[*s*] ← *child*

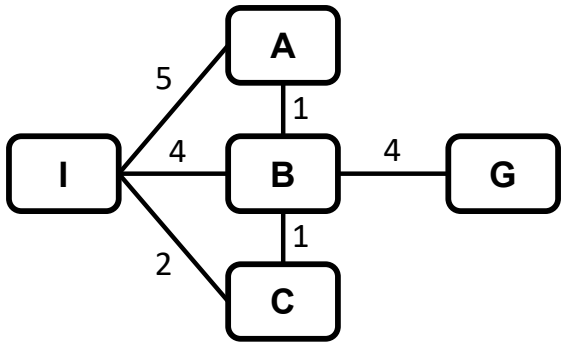
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

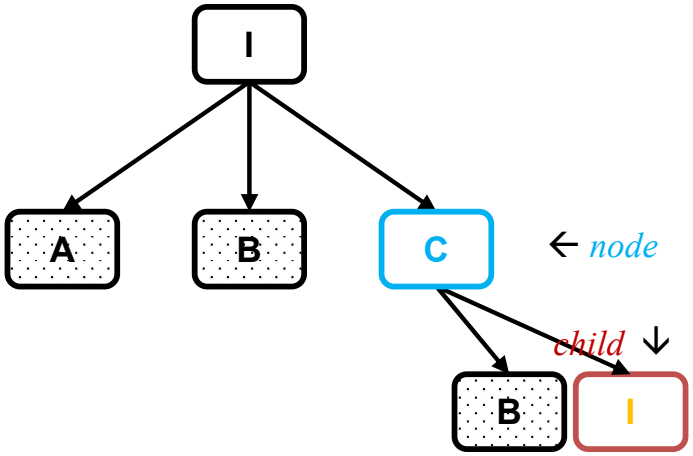
INITIAL STATE: I
GOAL STATE: G

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

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

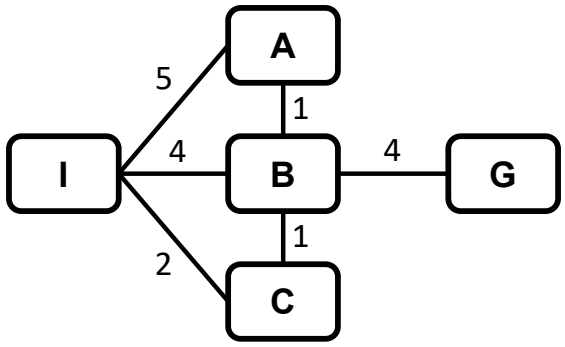
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

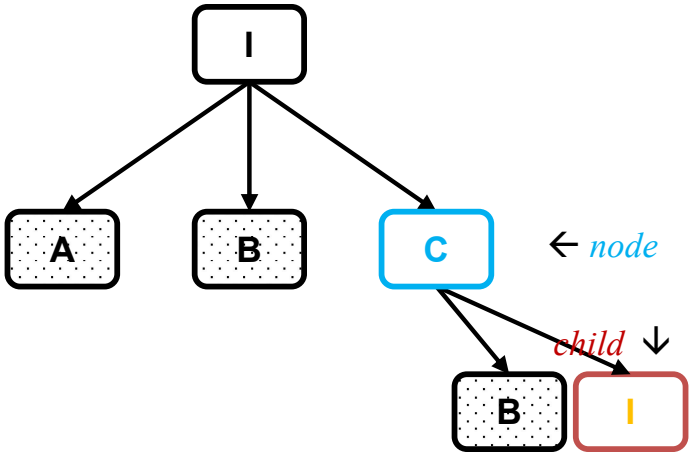
INITIAL STATE: I
GOAL STATE: G

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

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

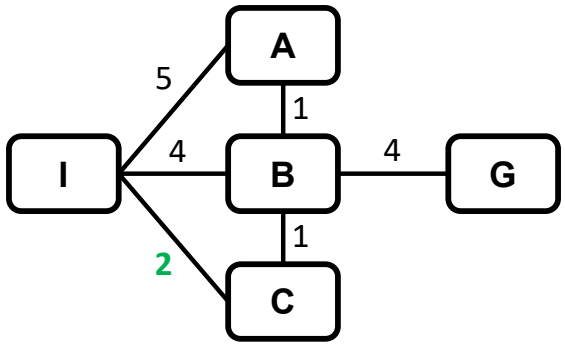
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

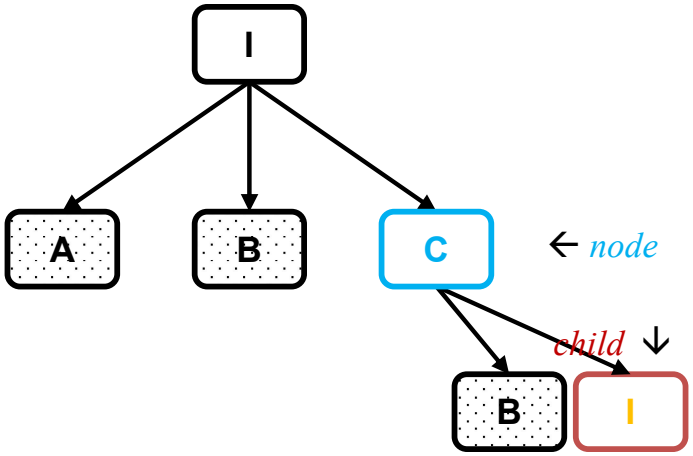
INITIAL STATE: I
GOAL STATE: G

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

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

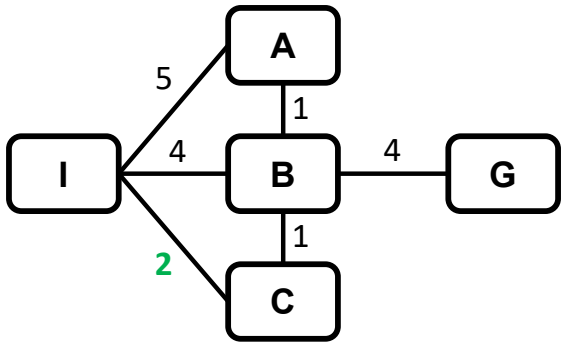
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

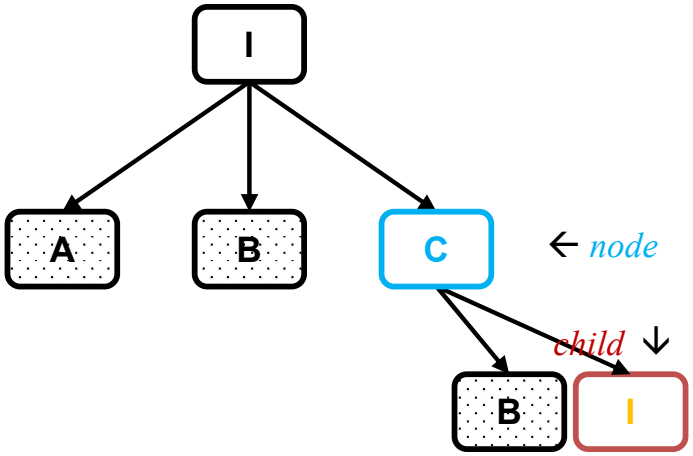
s ← *child*.STATE

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

reached[*s*] ← *child*

add *child* to *frontier*

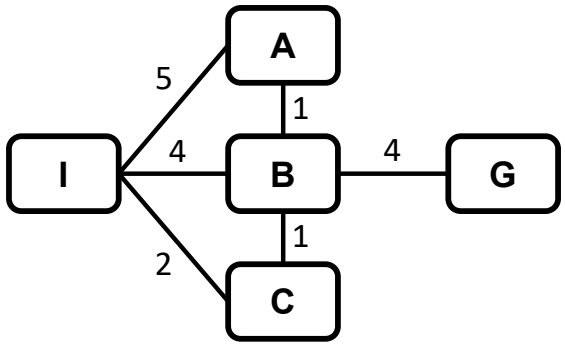
return *failure*



Path cost
2+2 > 0

X **Frontier node**

A* Search: Example



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

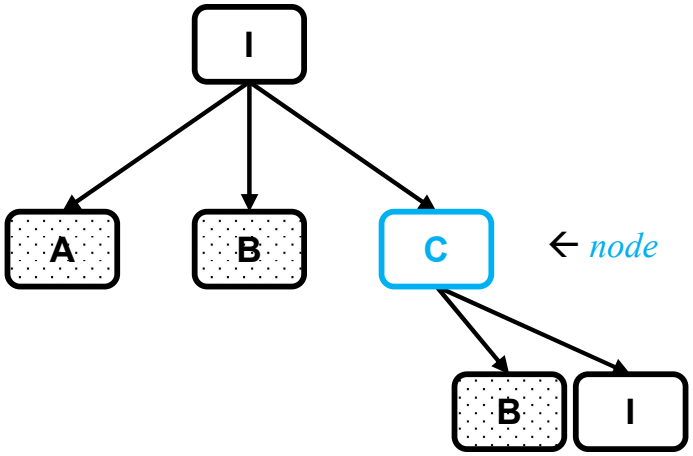
INITIAL STATE: I
GOAL STATE: G

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

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

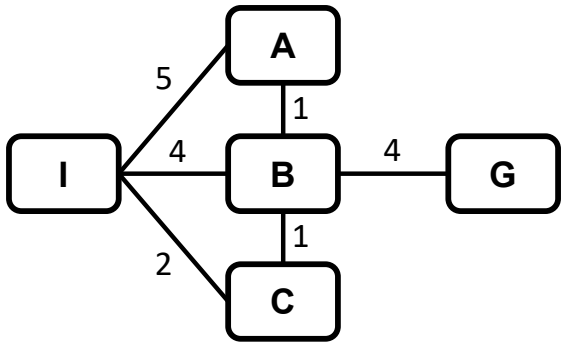
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

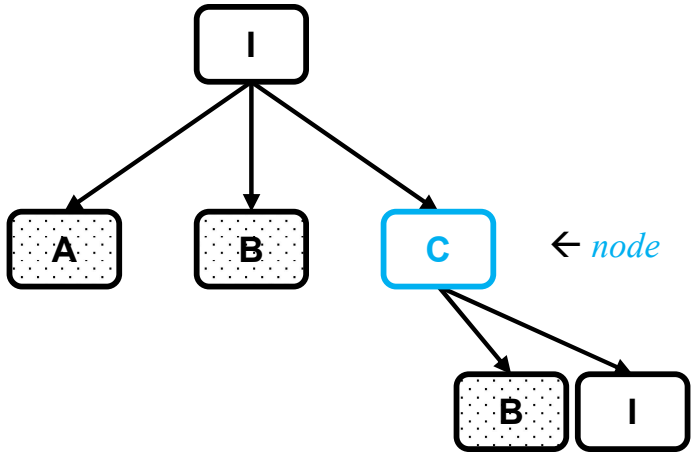
s ← *child*.STATE

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

reached[*s*] ← *child*

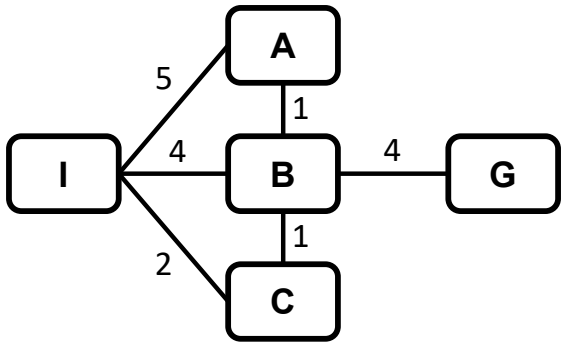
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

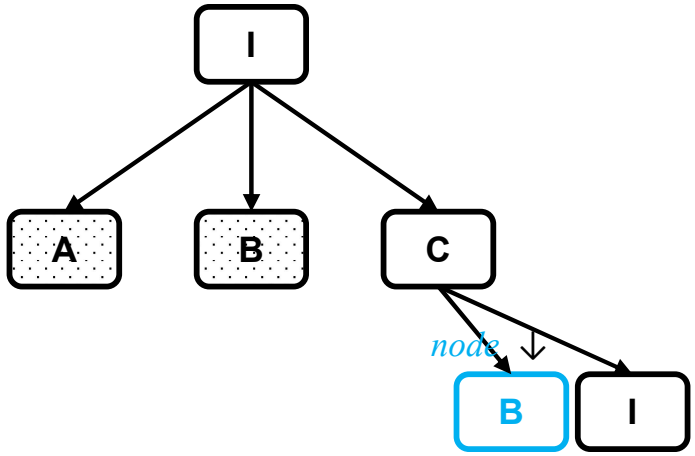
INITIAL STATE: I
GOAL STATE: G

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

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

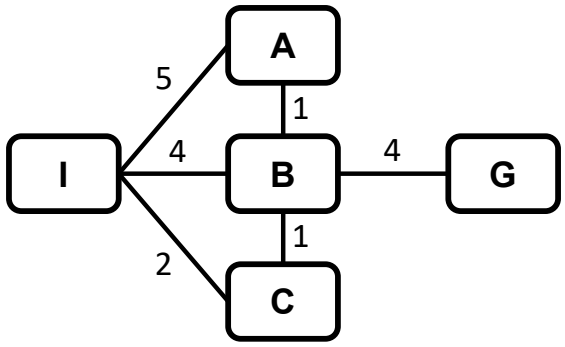
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

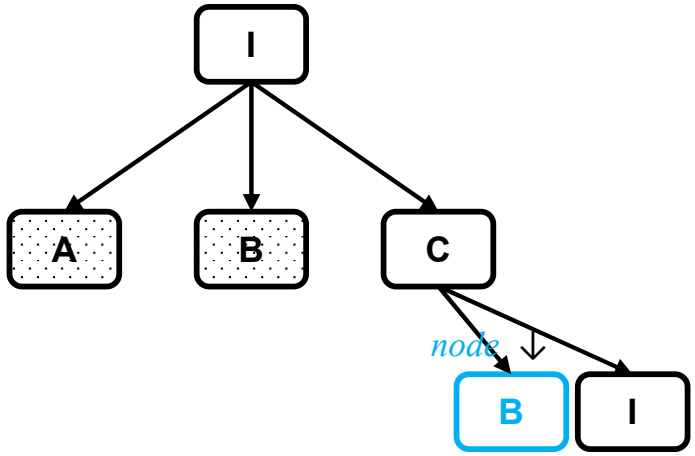
INITIAL STATE: I
GOAL STATE: G

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

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

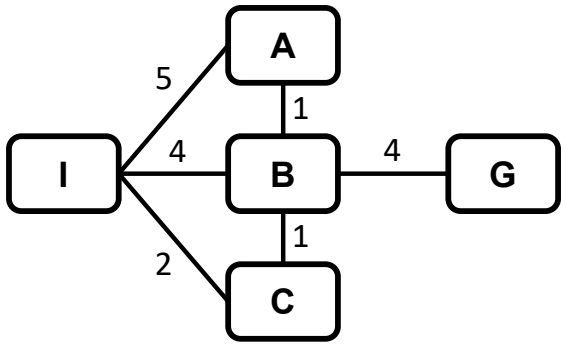
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

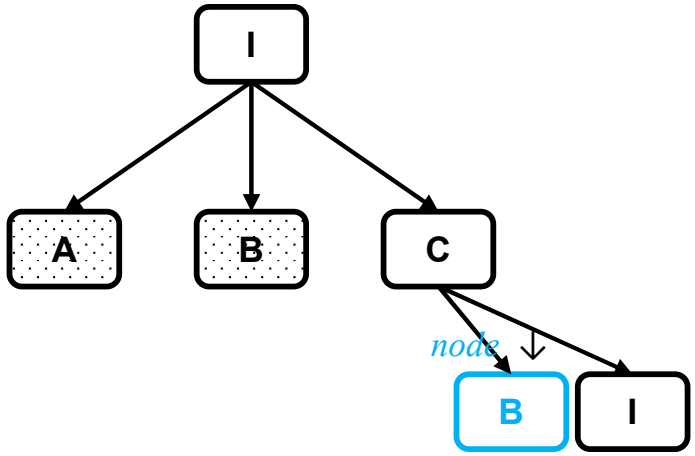
s ← *child*.STATE

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

reached[*s*] ← *child*

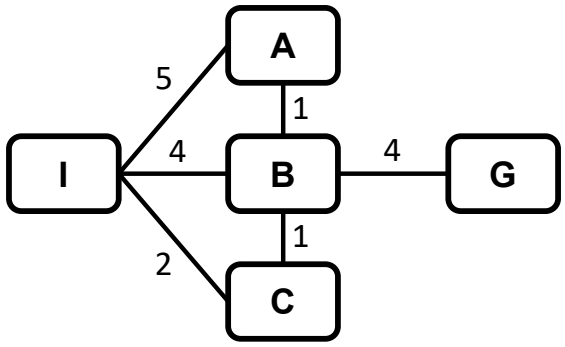
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

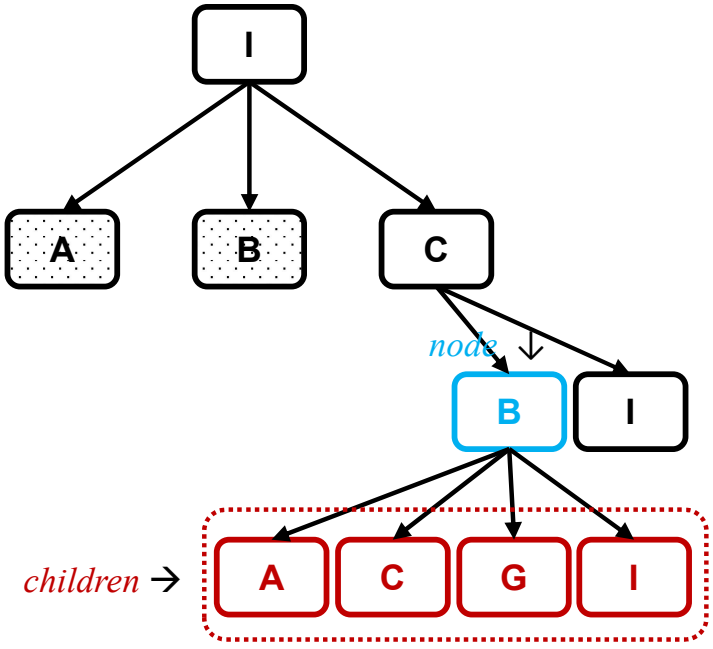
INITIAL STATE: I
GOAL STATE: G

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

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

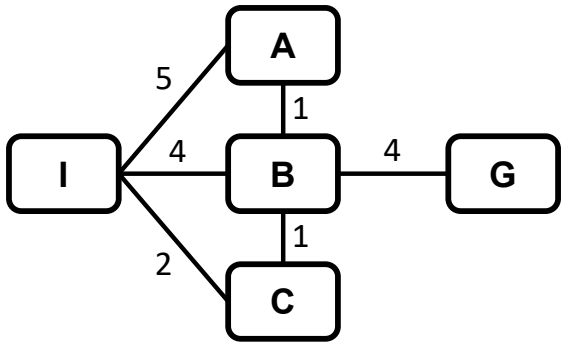
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

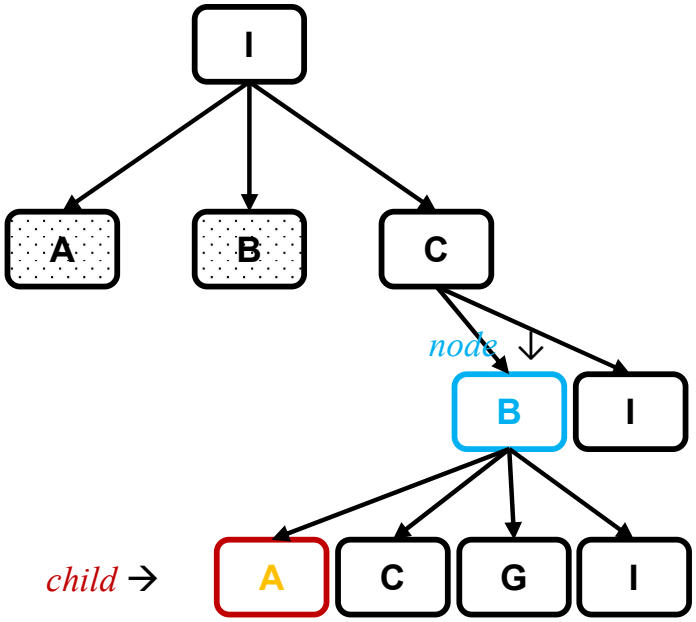
INITIAL STATE: I
GOAL STATE: G

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

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

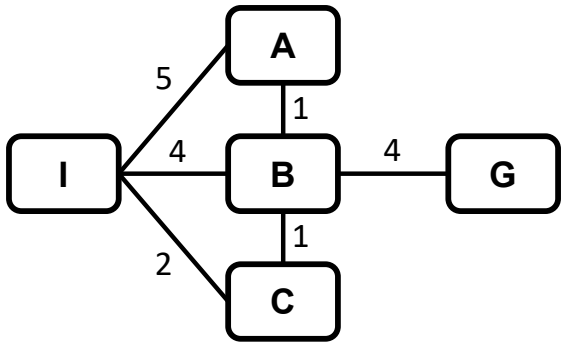
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

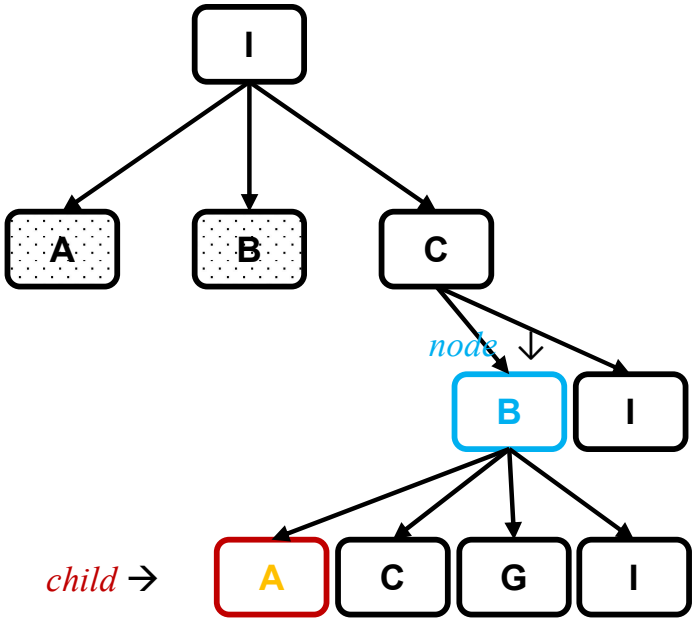
INITIAL STATE: I
GOAL STATE: G

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

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

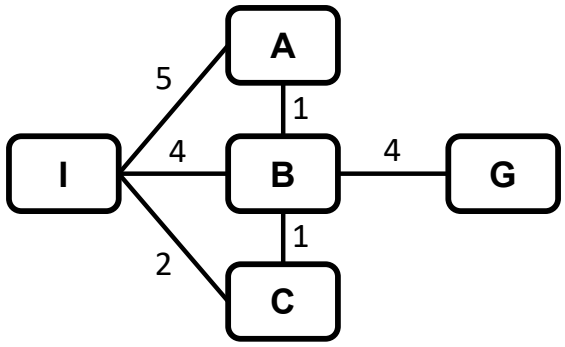
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

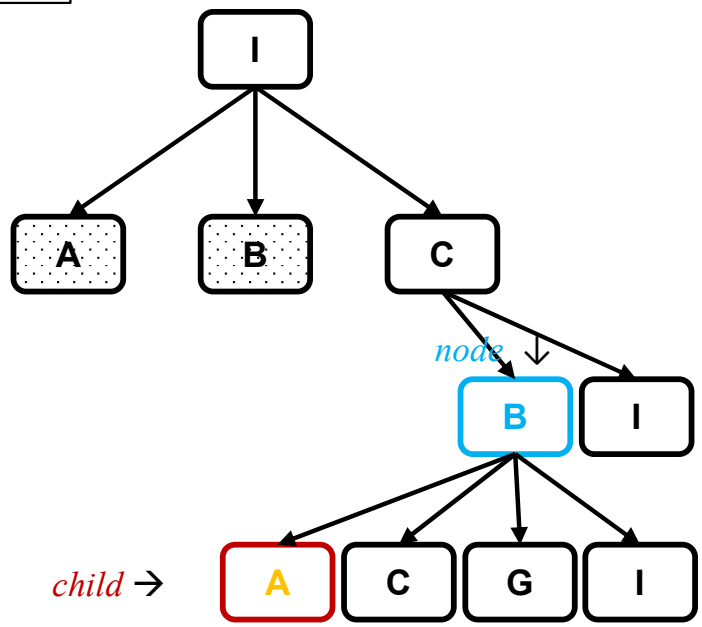
s ← *child*.STATE

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

reached[*s*] ← *child*

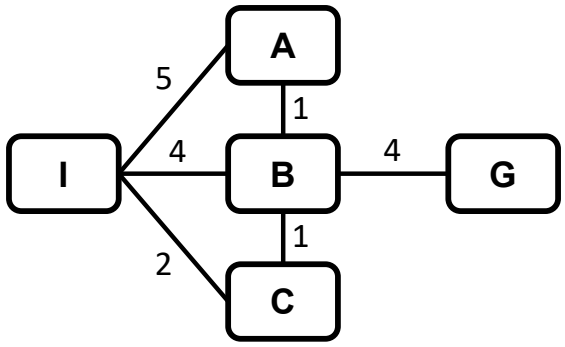
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

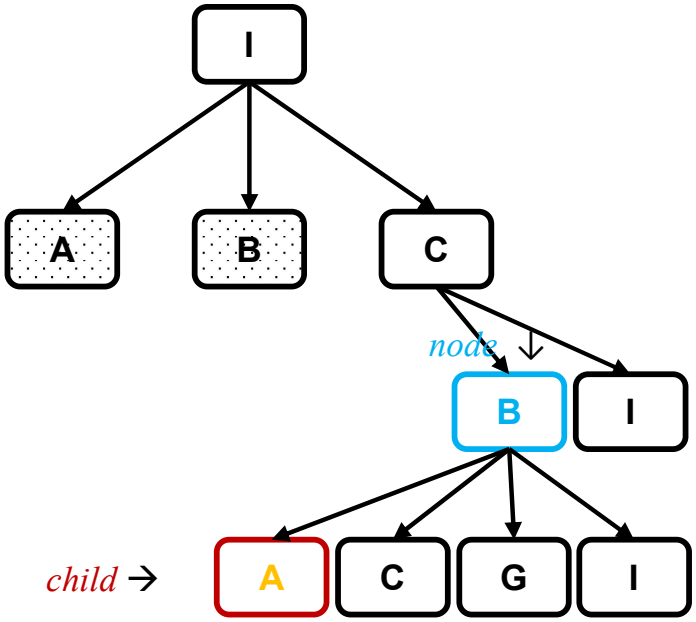
s ← *child*.STATE

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

reached[*s*] ← *child*

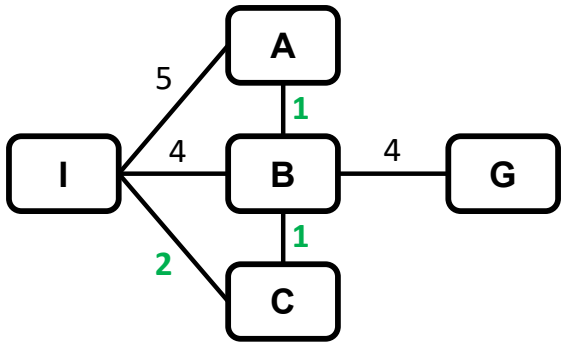
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

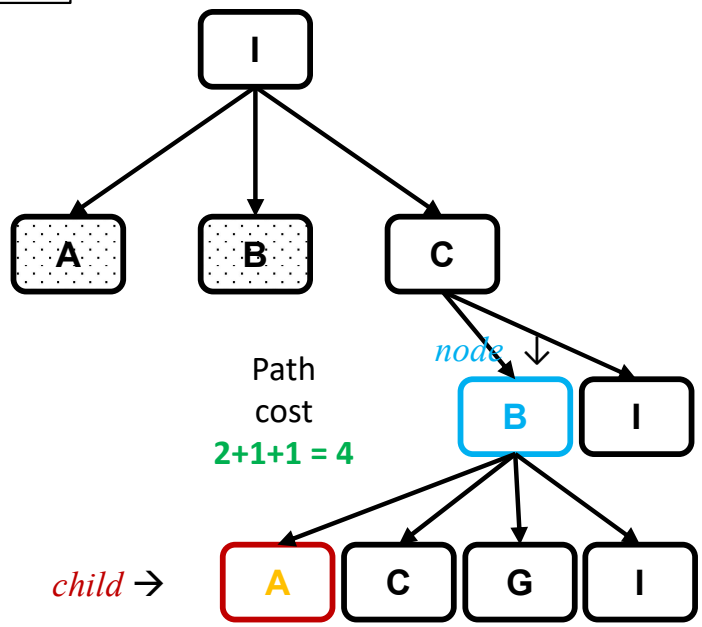
s ← *child*.STATE

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

reached[*s*] ← *child*

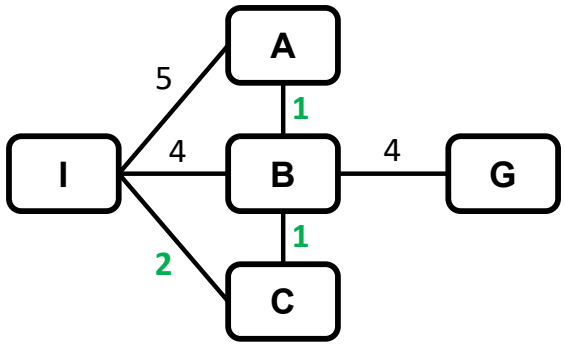
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

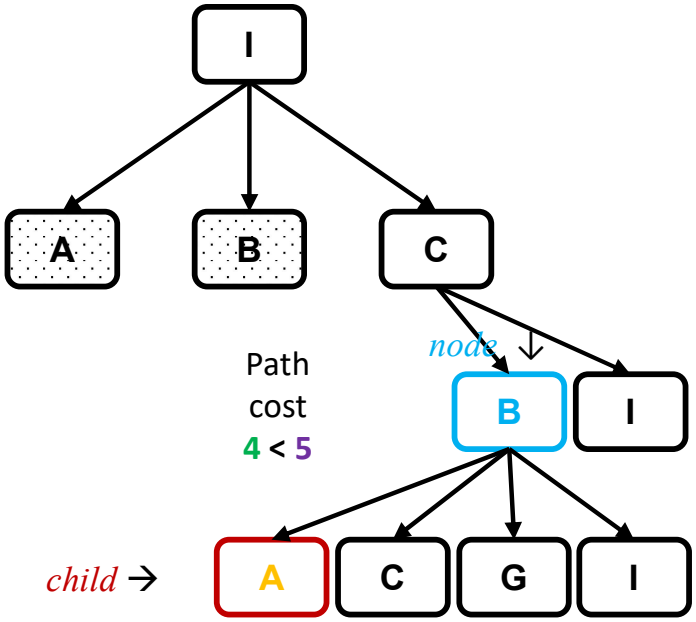
s ← *child*.STATE

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

reached[*s*] ← *child*

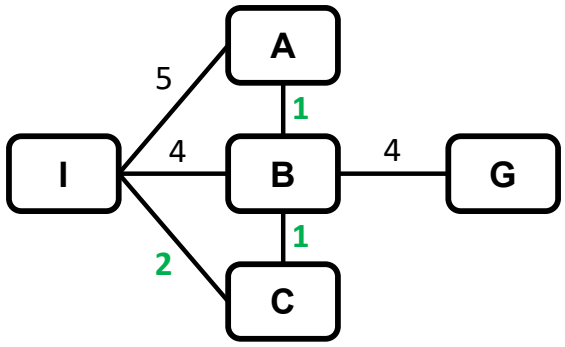
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

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

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

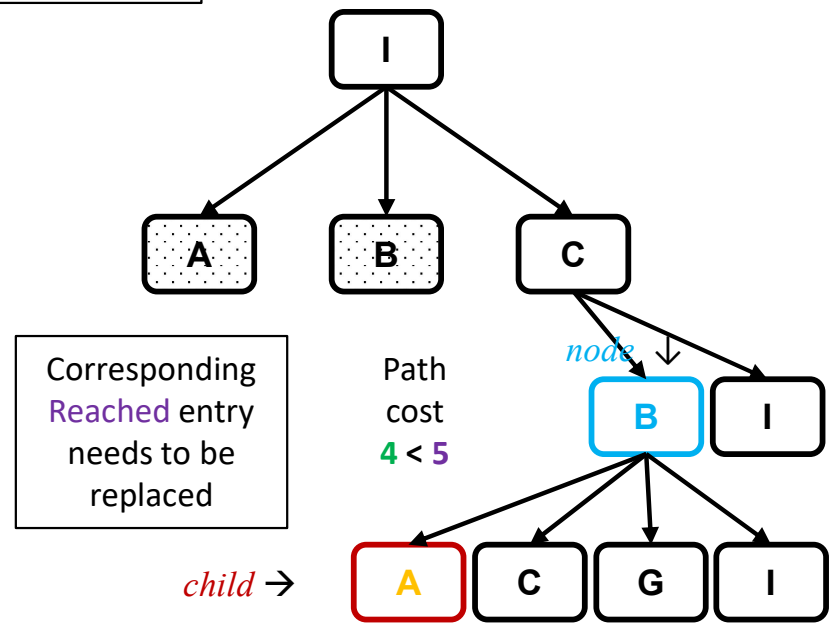
s ← *child*.STATE

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

reached[*s*] ← *child*

add *child* to *frontier*

return *failure*

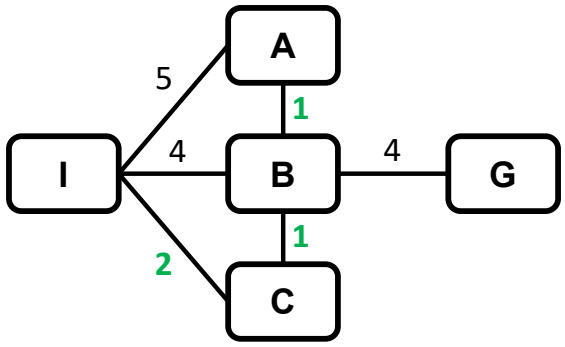


Corresponding Reached entry needs to be replaced

Path cost 4 < 5

X **Frontier node**

A* Search: Example



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

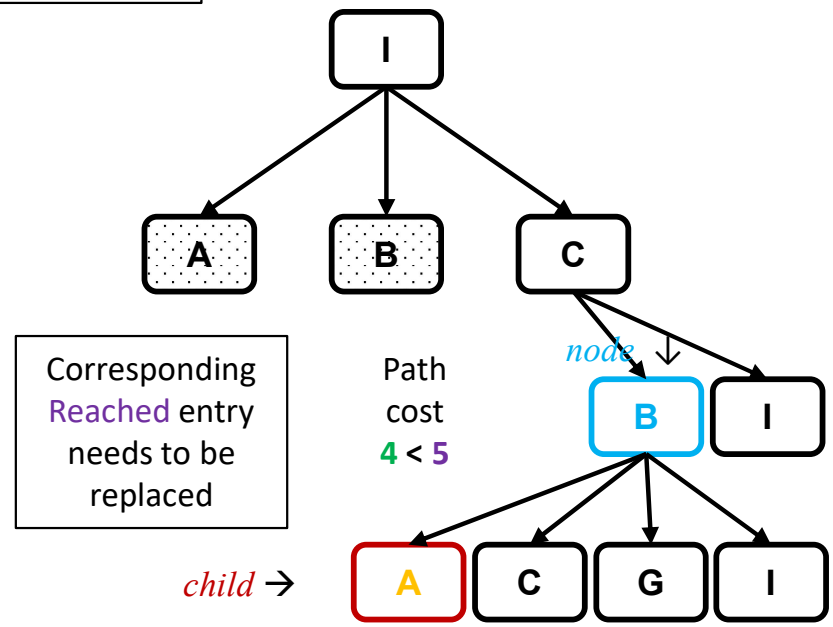
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

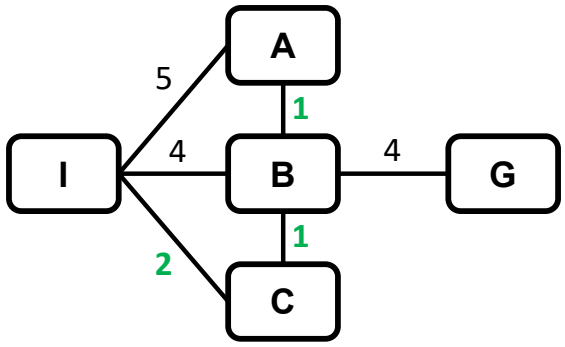


Corresponding Reached entry needs to be replaced

Path cost **4 < 5**

X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

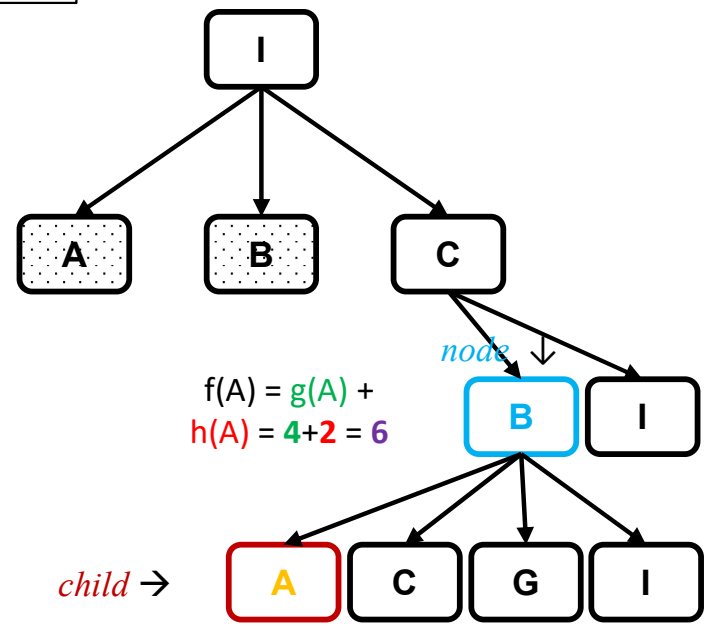
s ← *child*.STATE

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

reached[*s*] ← *child*

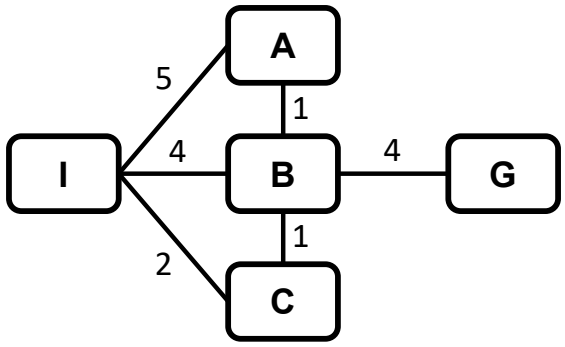
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

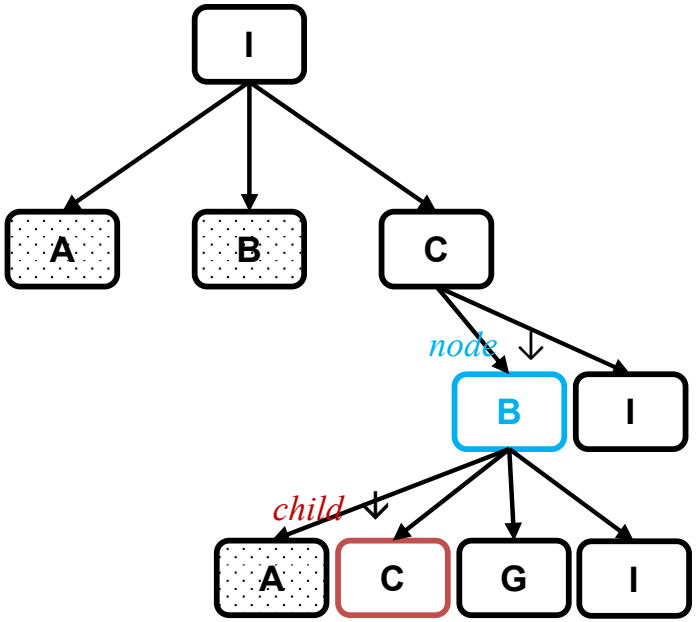
      s ← *child*.STATE

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

        reached[*s*] ← *child*

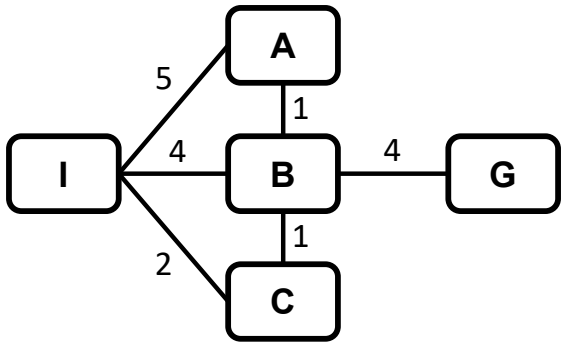
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

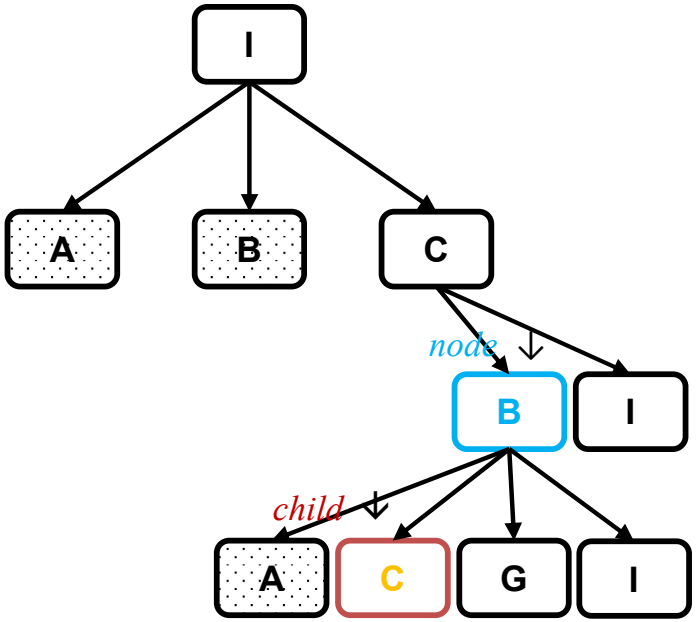
s ← *child*.STATE

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

reached[*s*] ← *child*

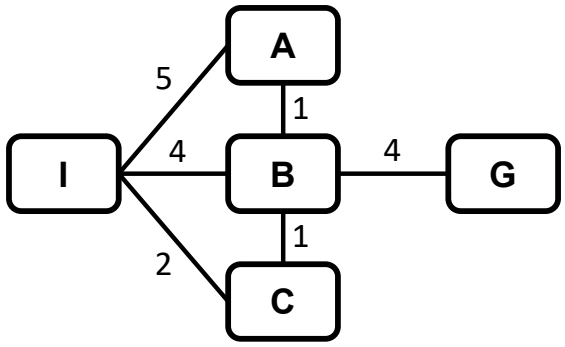
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

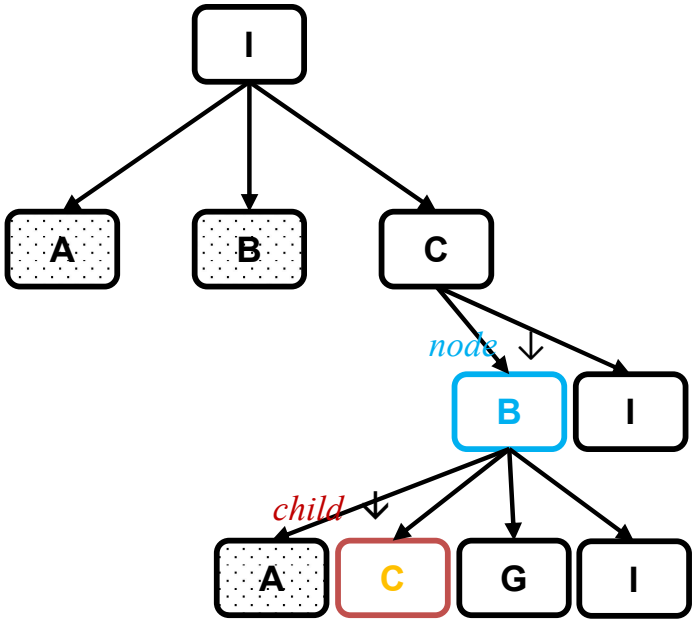
s ← *child*.STATE

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

reached[*s*] ← *child*

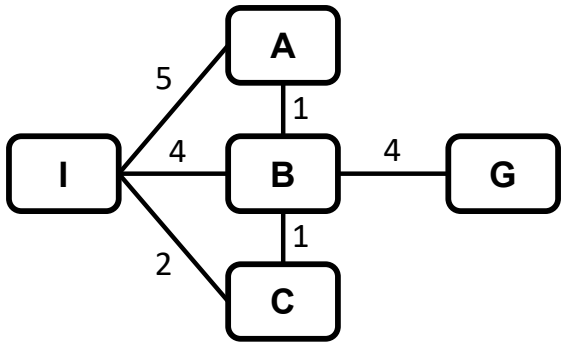
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

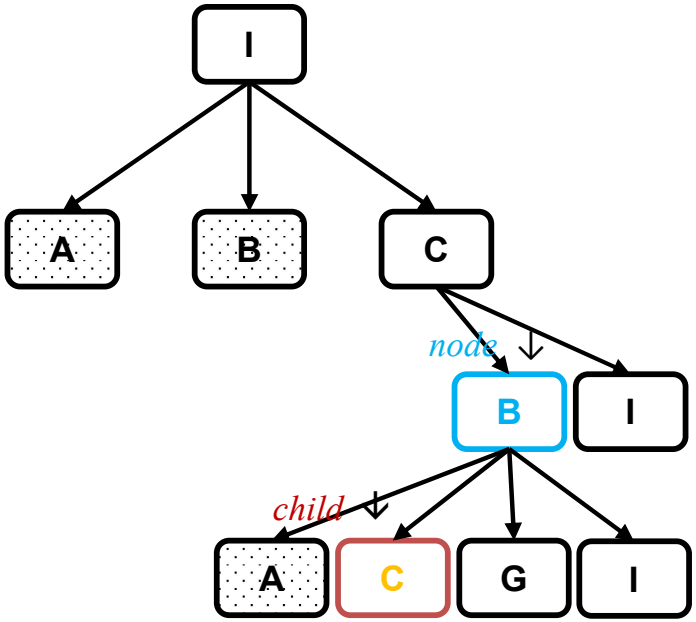
s ← *child*.STATE

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

reached[*s*] ← *child*

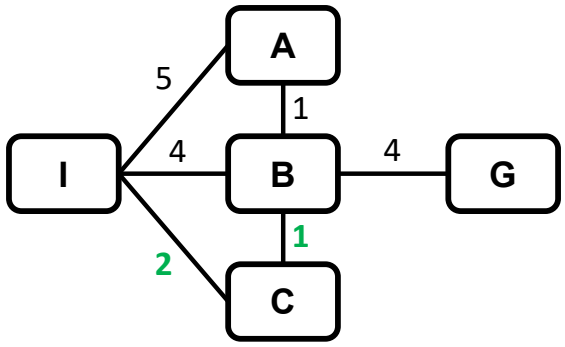
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I				
	Node	A	B	A				
	f(Node)	6	7	7				
Reached	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	**2**			

State Space **Graph** | Frontier / Reached
Algorithm | **Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

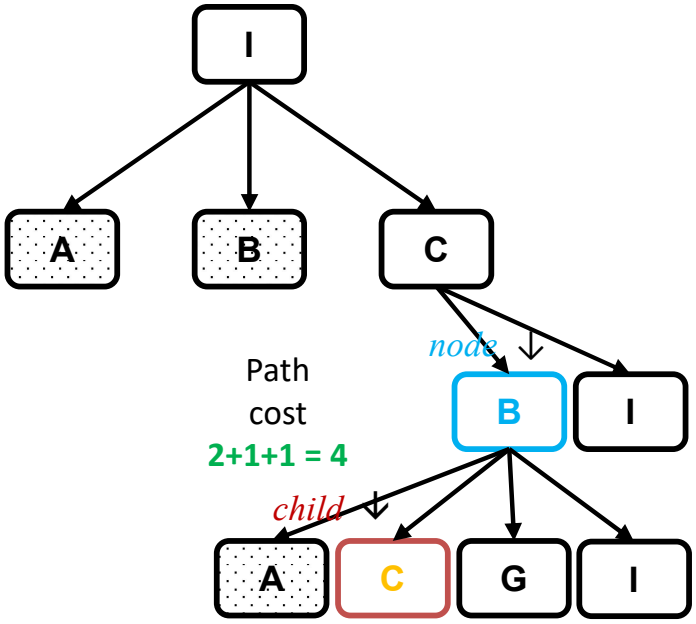
s ← *child*.STATE

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

reached[*s*] ← *child*

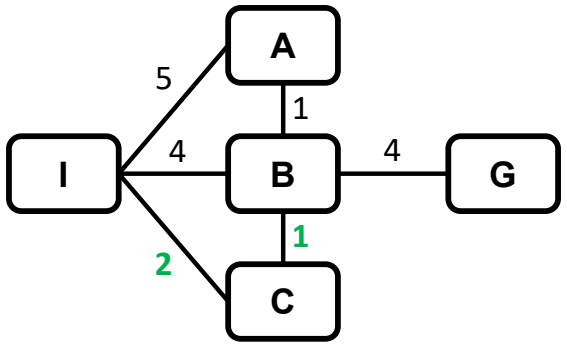
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

Frontier	Parent	B	I	I				
	Node	A	B	A				
	f(Node)	6	7	7				
Reached	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	**2**			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

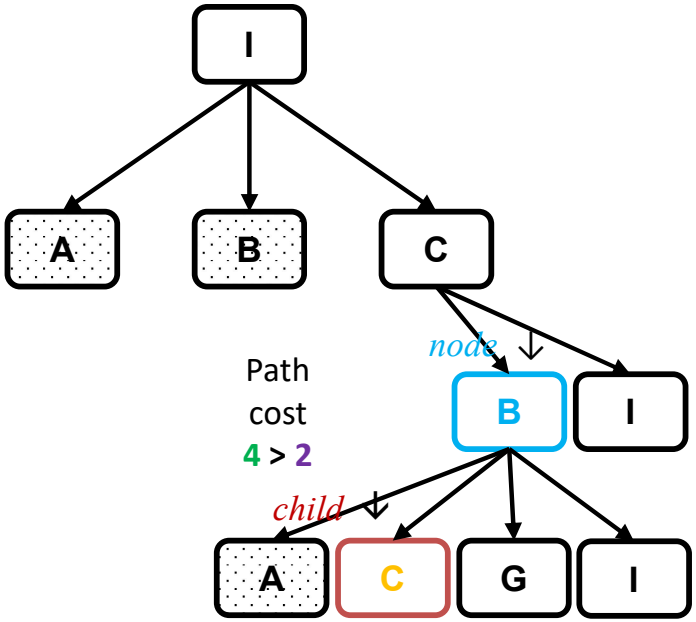
s ← *child*.STATE

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

reached[*s*] ← *child*

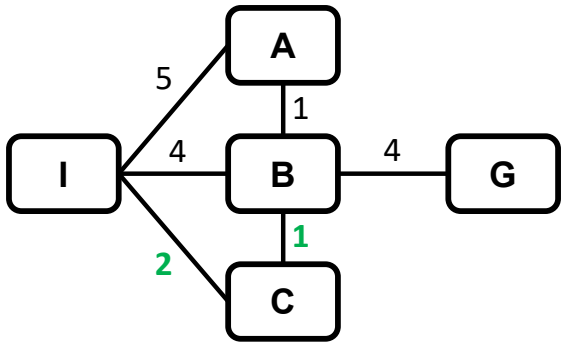
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	**2**			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

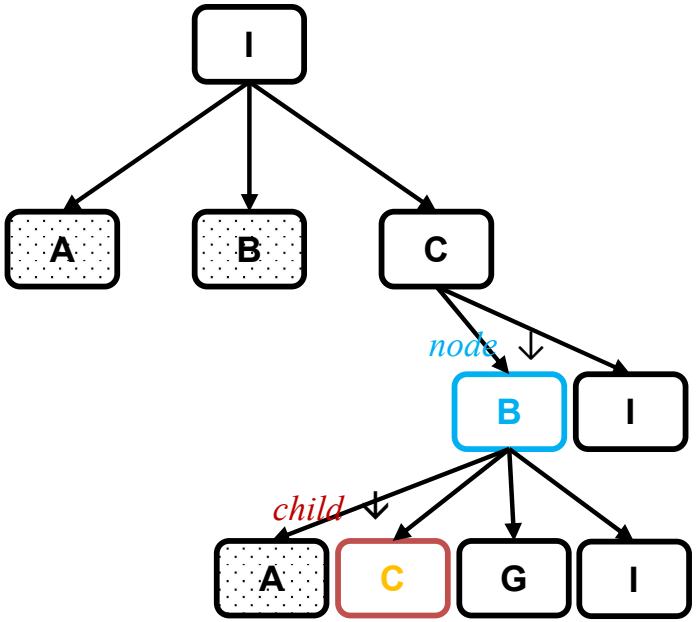
s ← *child*.STATE

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

reached[*s*] ← *child*

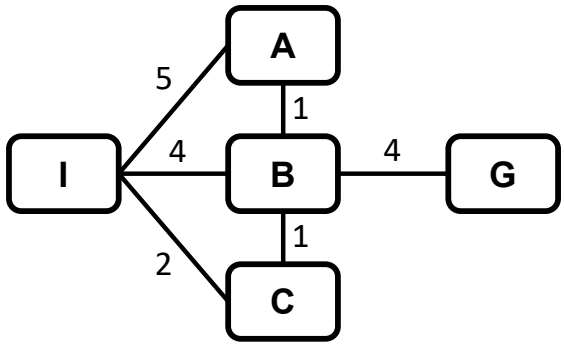
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

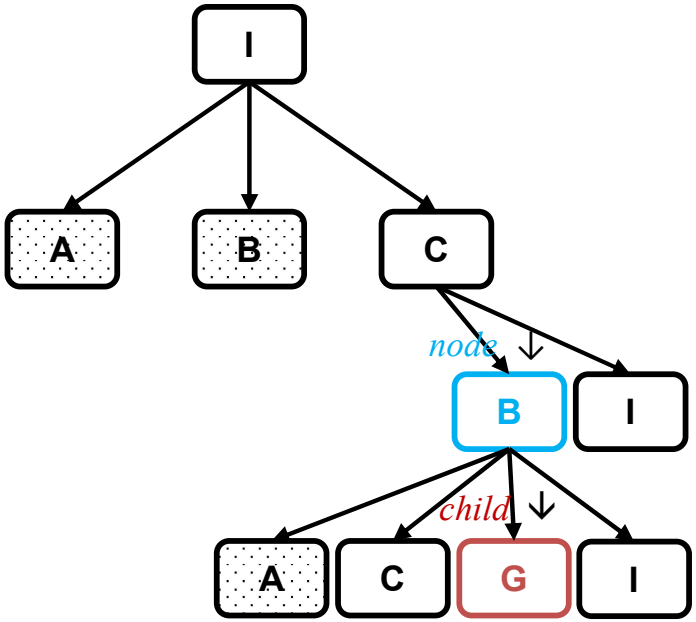
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

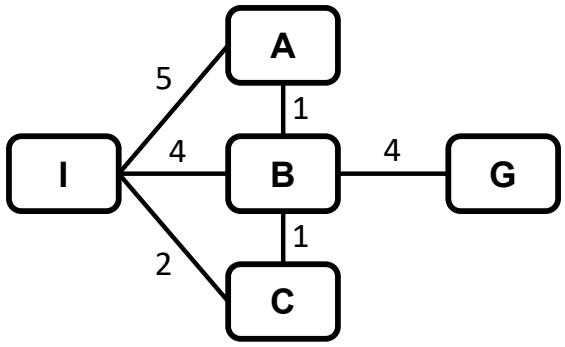
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

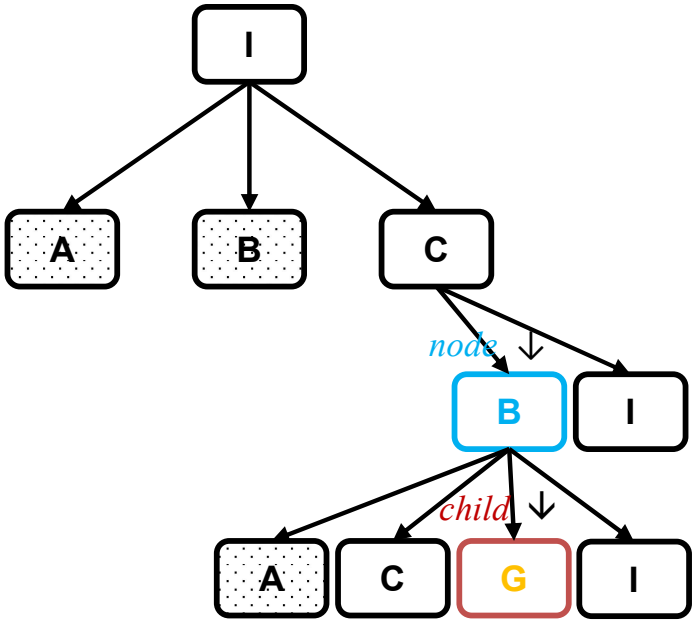
s ← *child*.STATE

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

reached[*s*] ← *child*

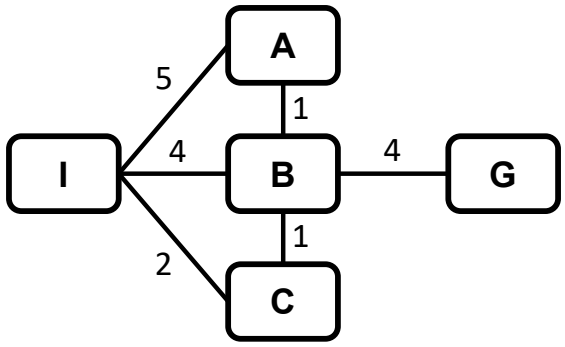
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

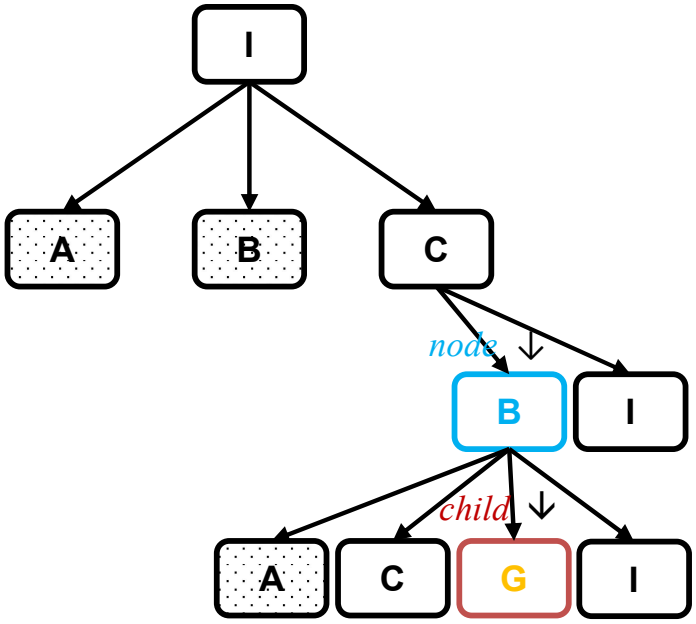
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

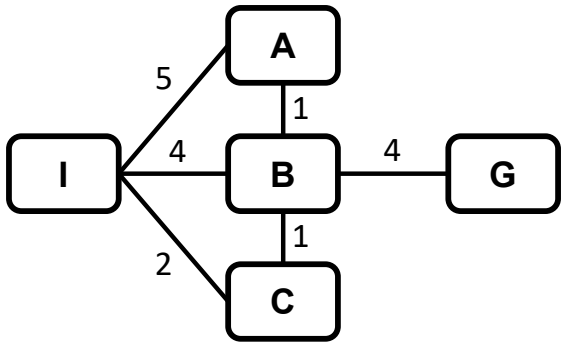
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

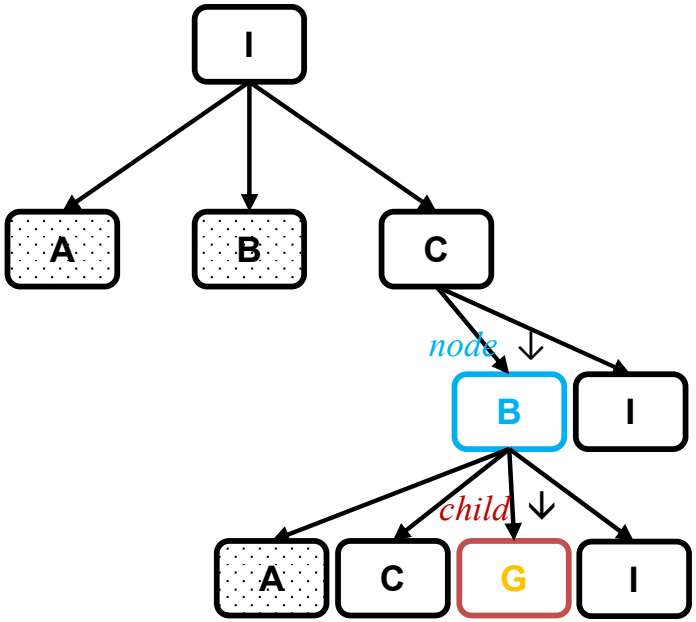
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

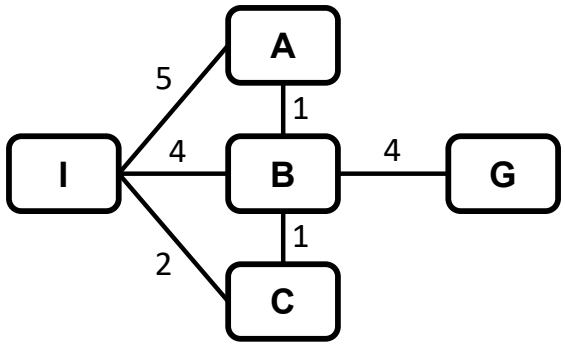
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I			
	Key/State	I	A	B	C			
	Path cost	0	4	3	2			

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

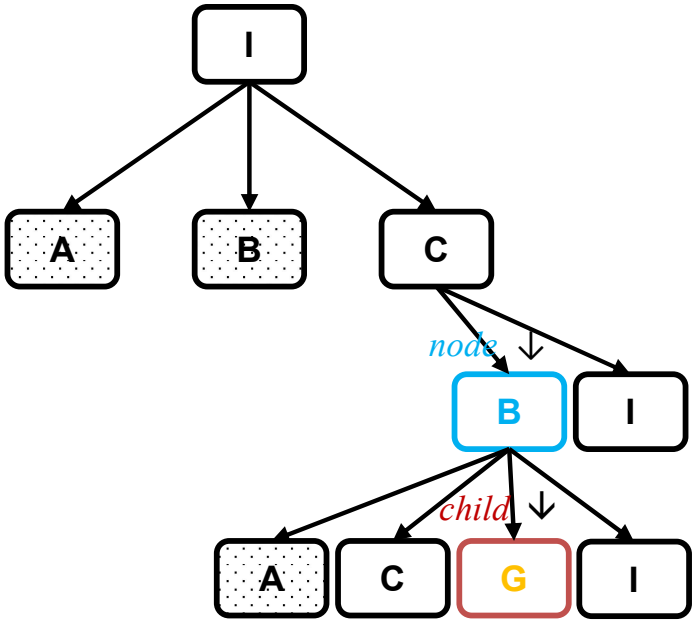
s ← *child*.STATE

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

reached[*s*] ← *child*

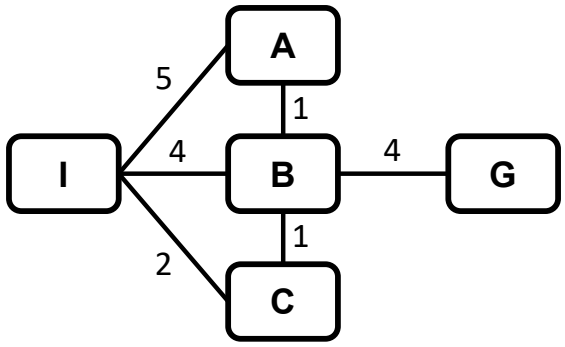
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

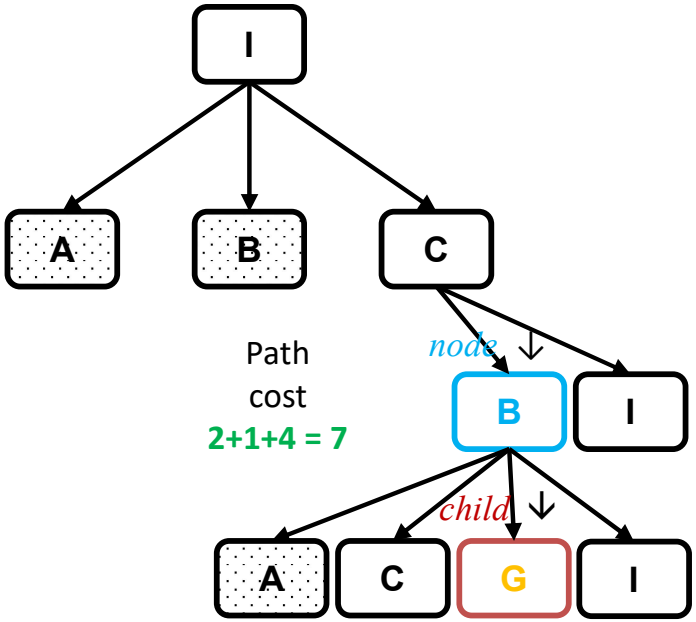
      s ← *child*.STATE

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

        reached[*s*] ← *child*

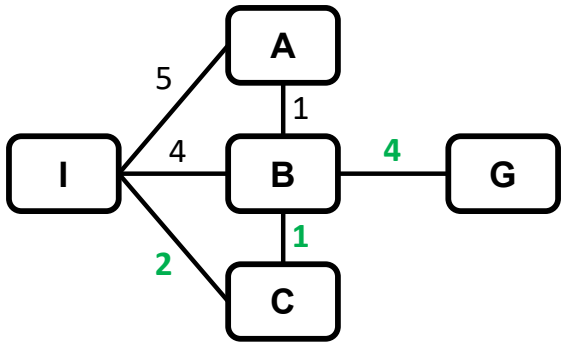
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

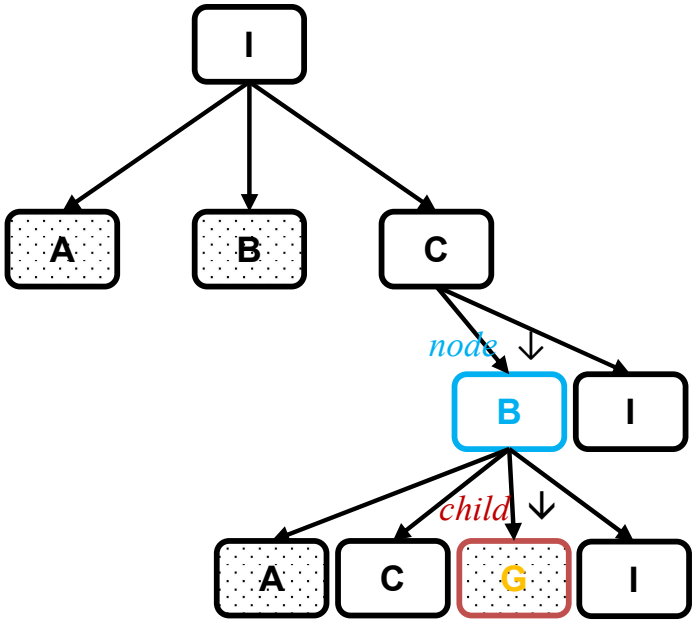
INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

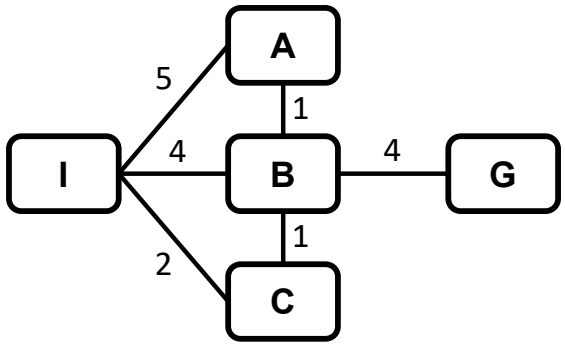
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

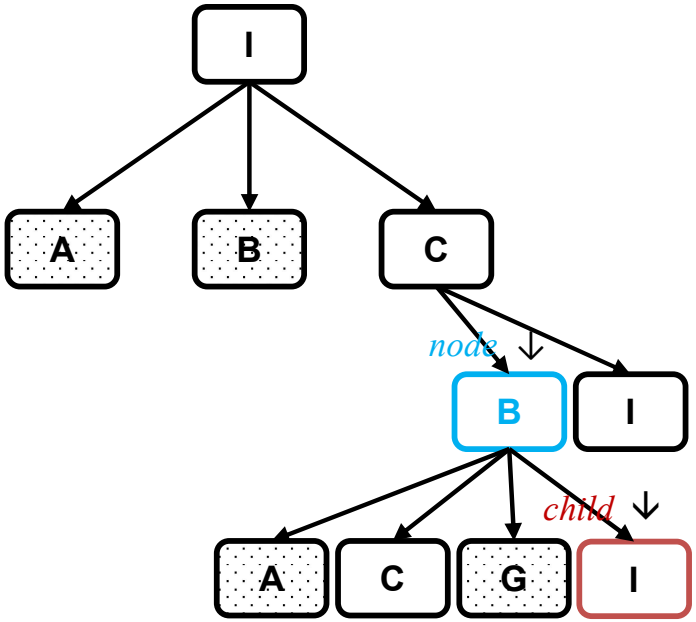
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

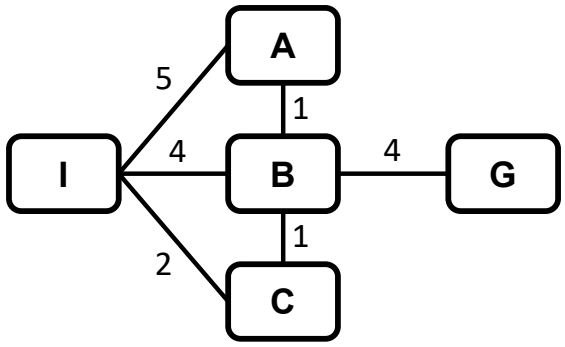
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

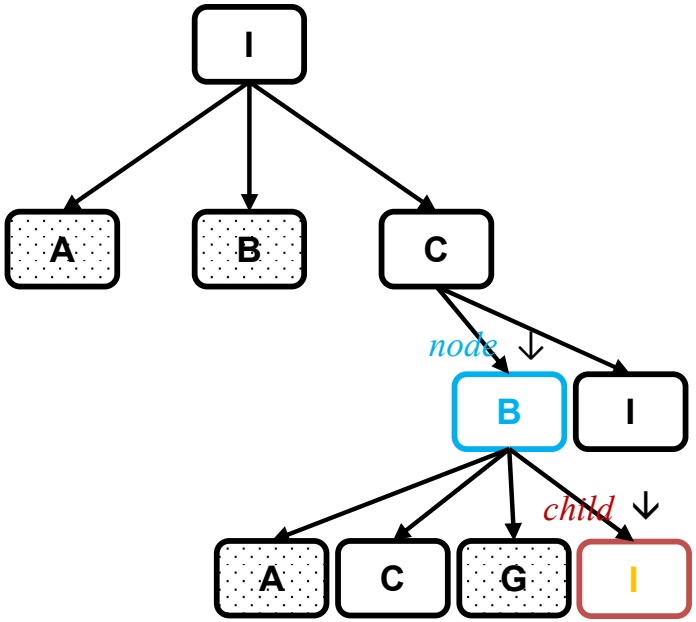
s ← *child*.STATE

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

reached[*s*] ← *child*

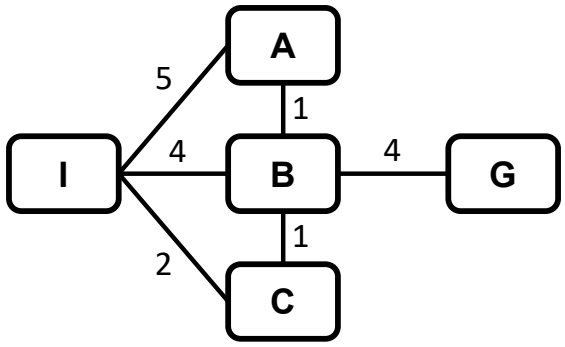
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

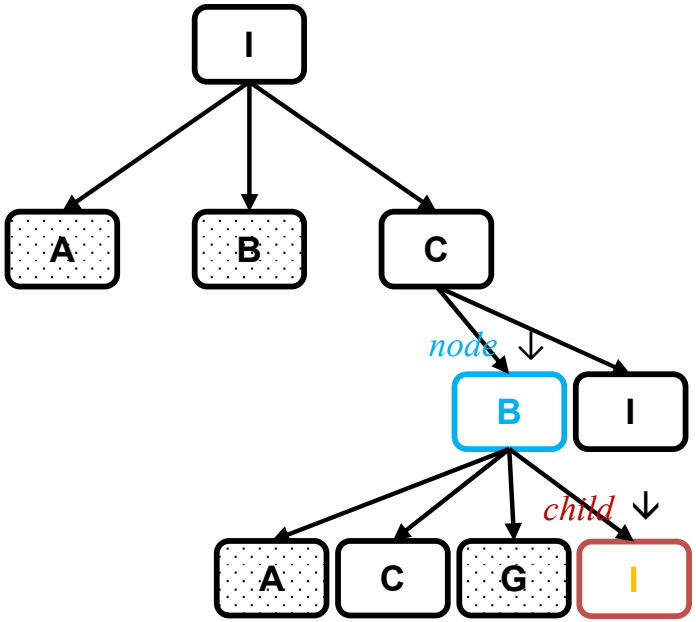
INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

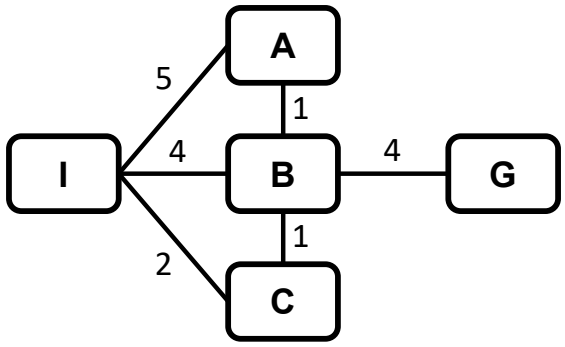
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

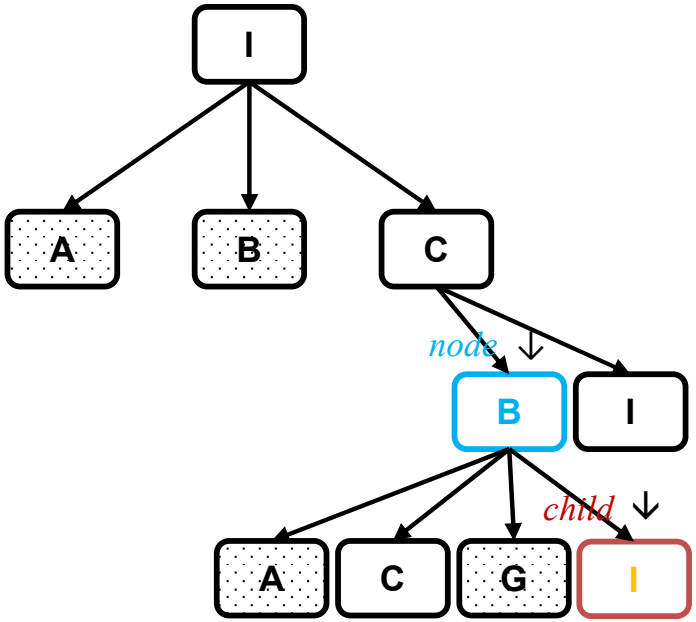
s ← *child*.STATE

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

reached[*s*] ← *child*

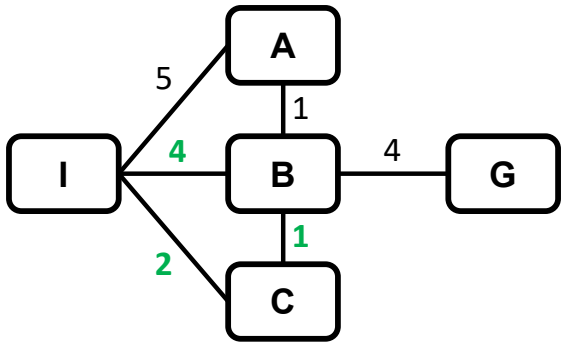
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

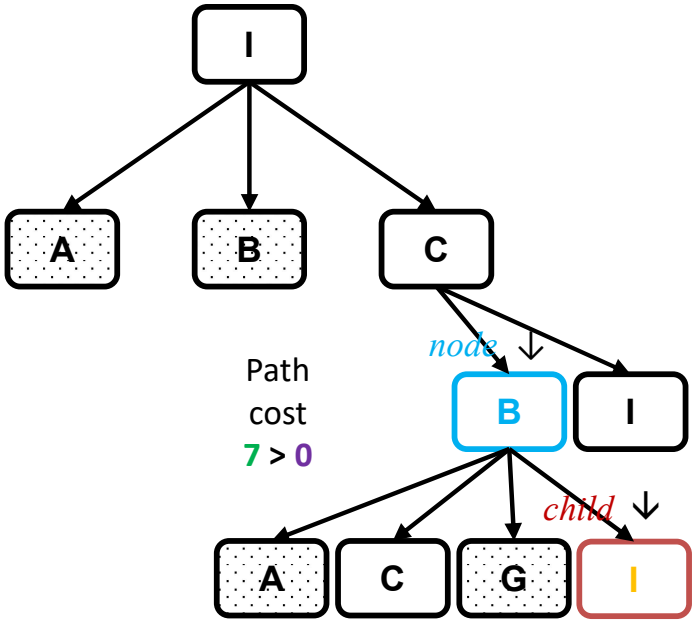
s ← *child*.STATE

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

reached[*s*] ← *child*

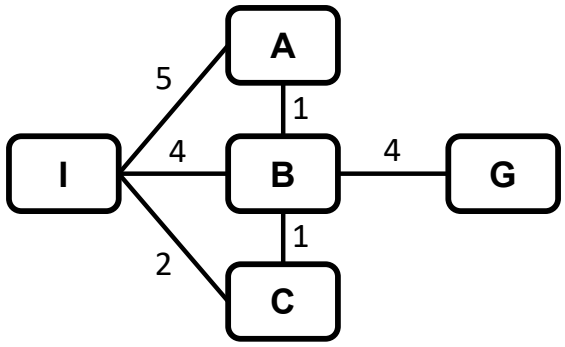
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

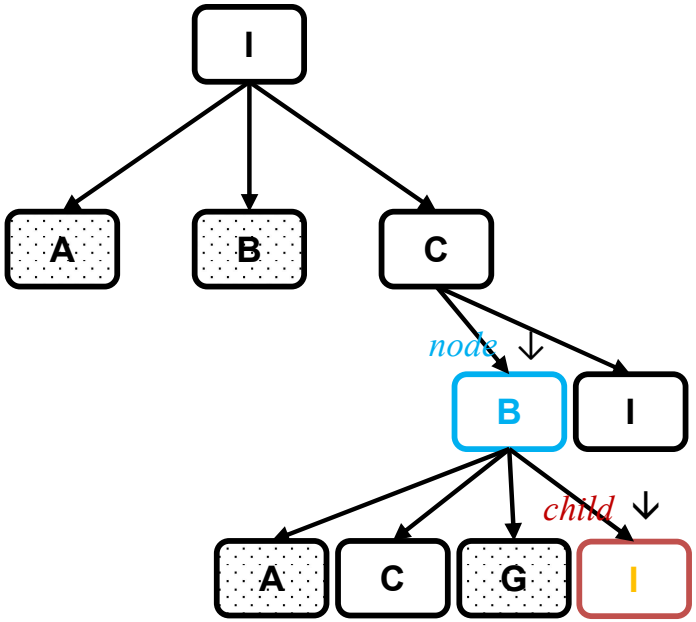
s ← *child*.STATE

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

reached[*s*] ← *child*

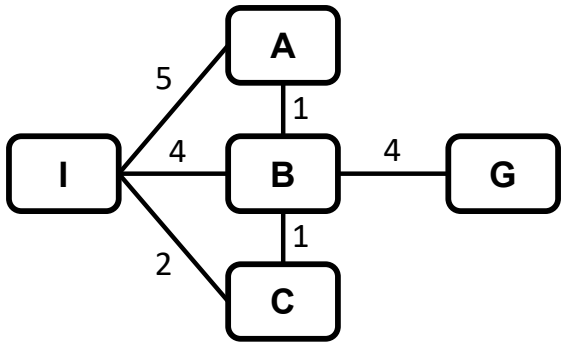
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

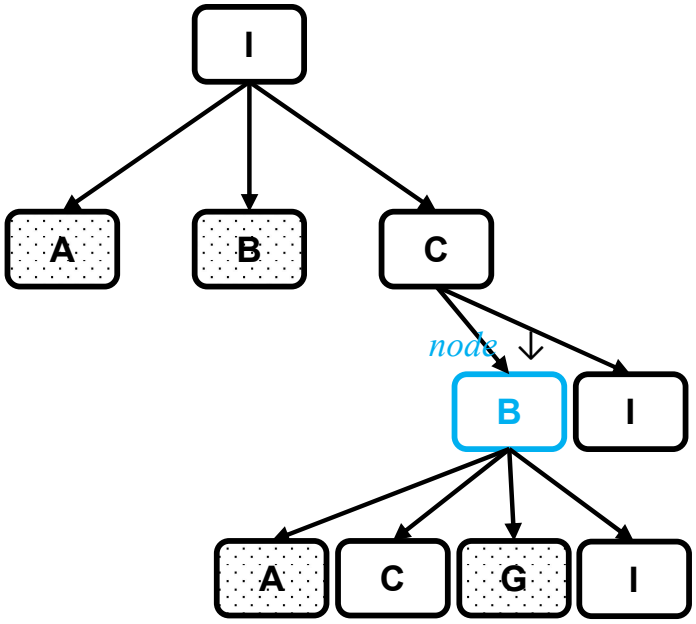
s ← *child*.STATE

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

reached[*s*] ← *child*

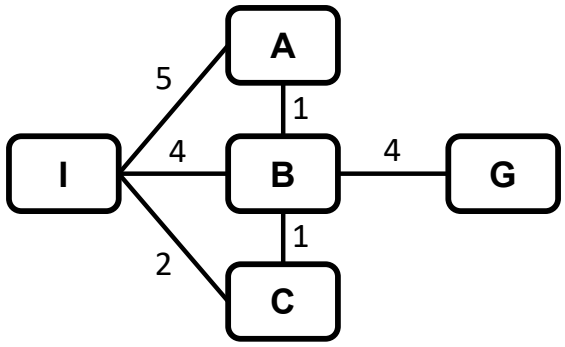
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

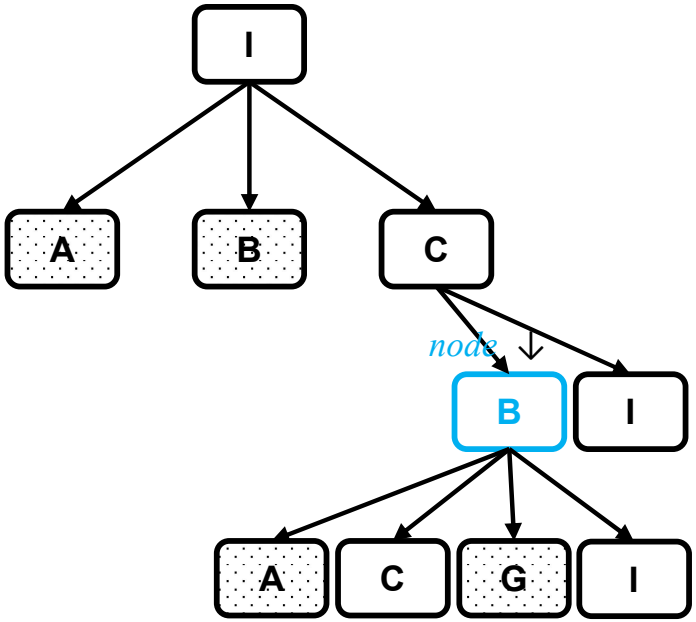
s ← *child*.STATE

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

reached[*s*] ← *child*

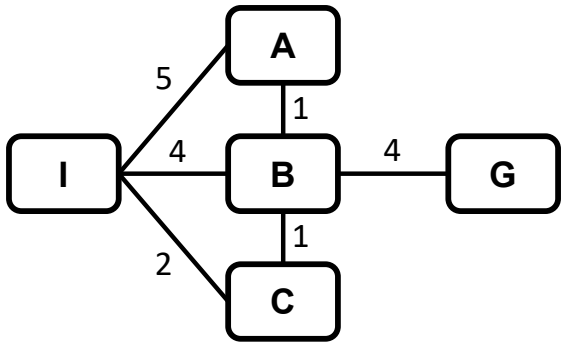
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

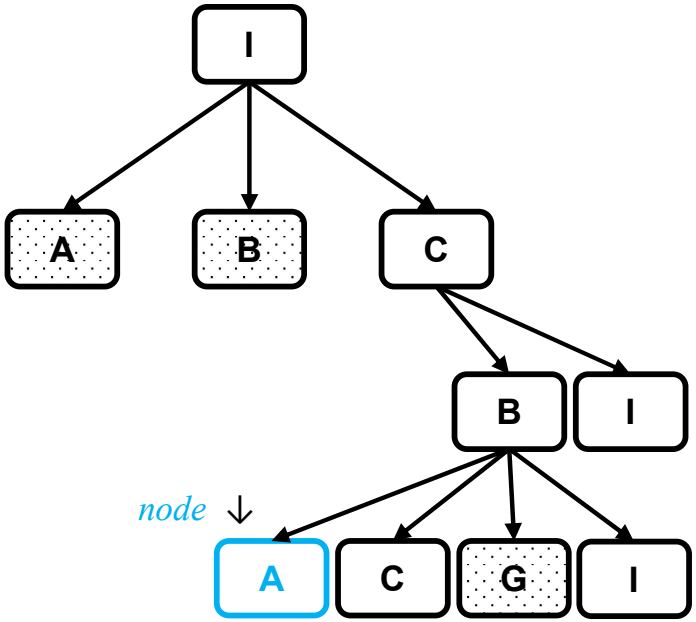
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

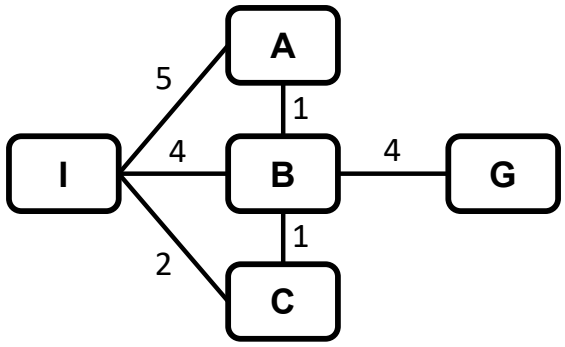
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

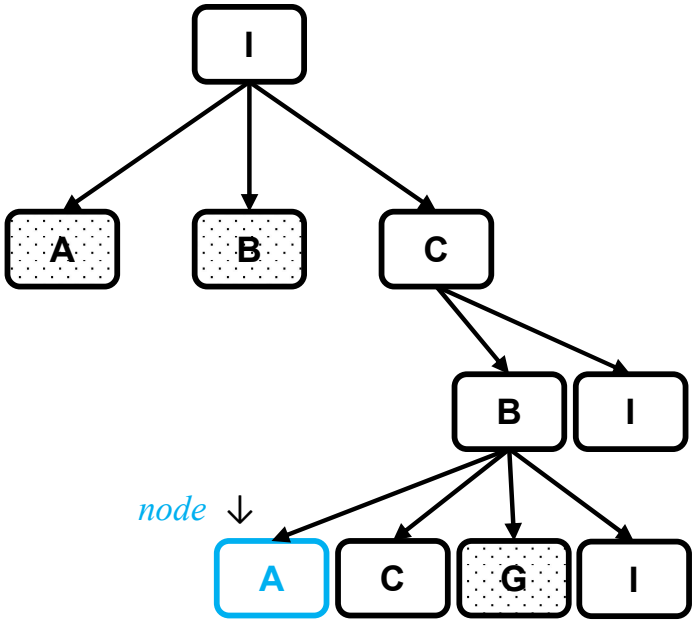
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

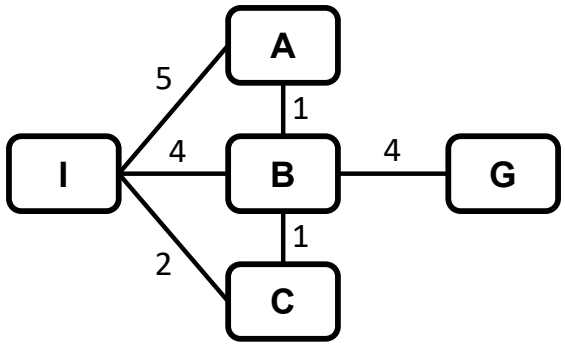
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

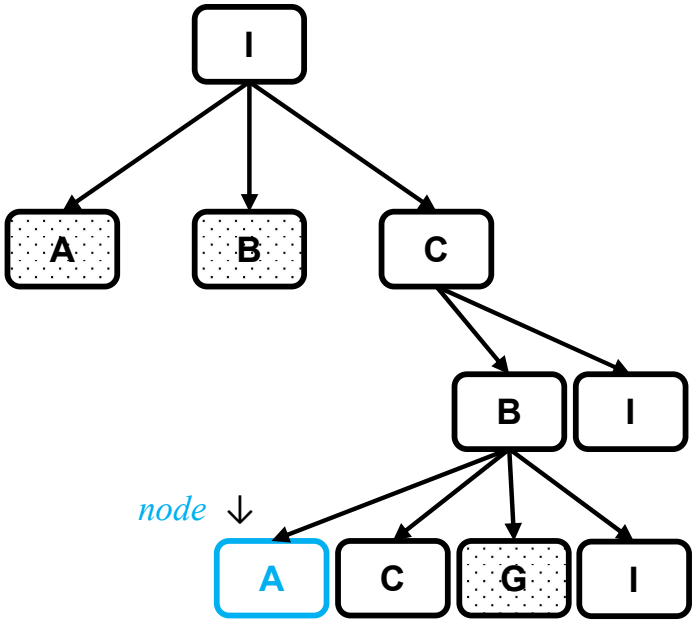
INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

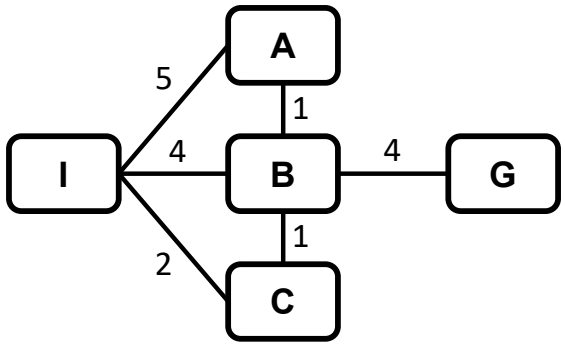
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

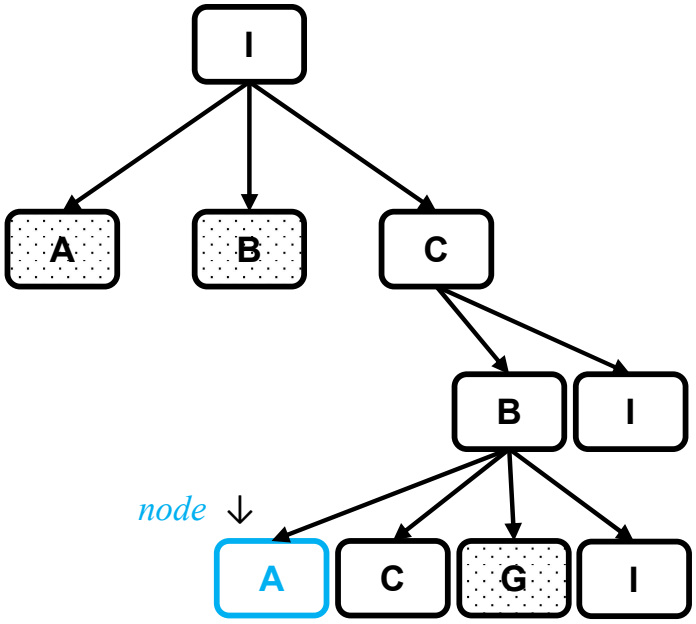
      s ← *child*.STATE

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

        reached[*s*] ← *child*

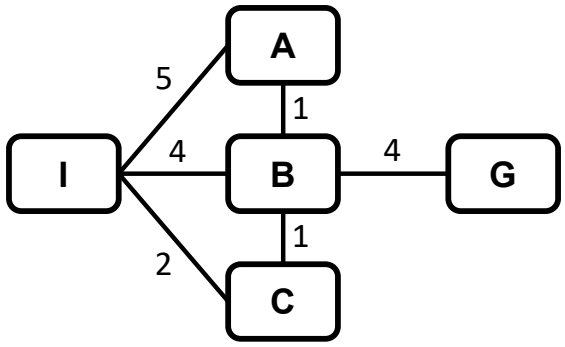
        add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

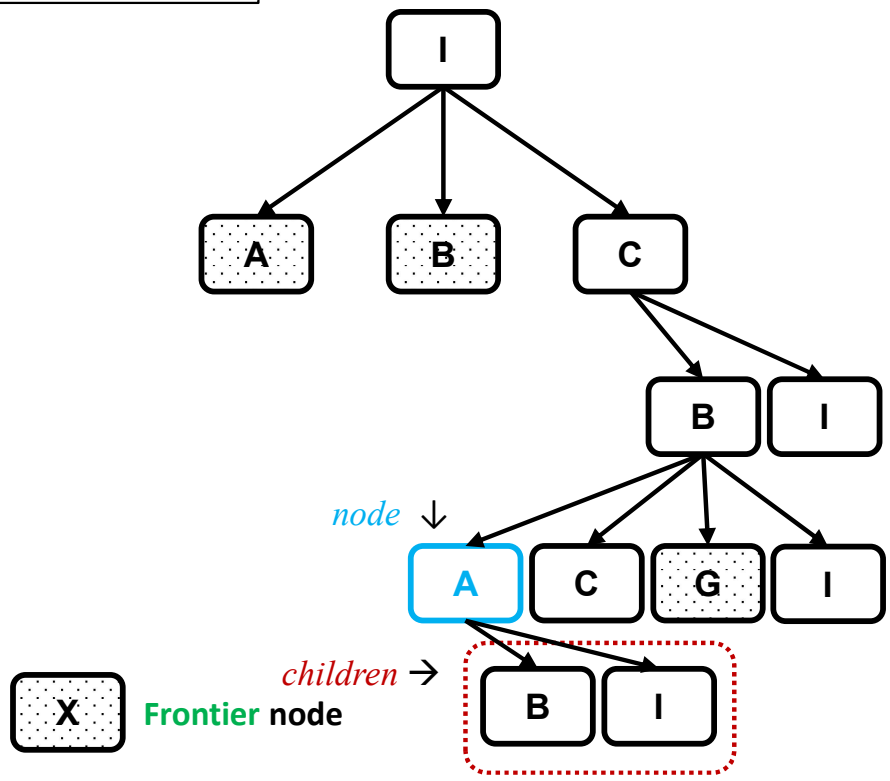
      s ← *child*.STATE

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

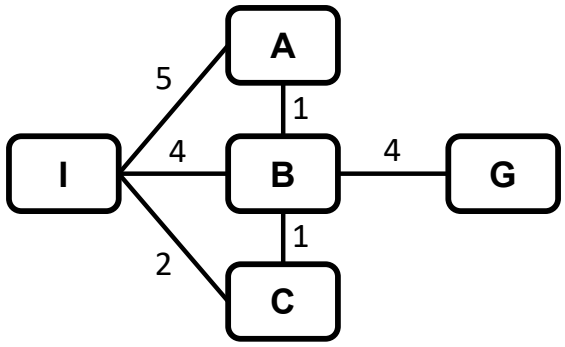
        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

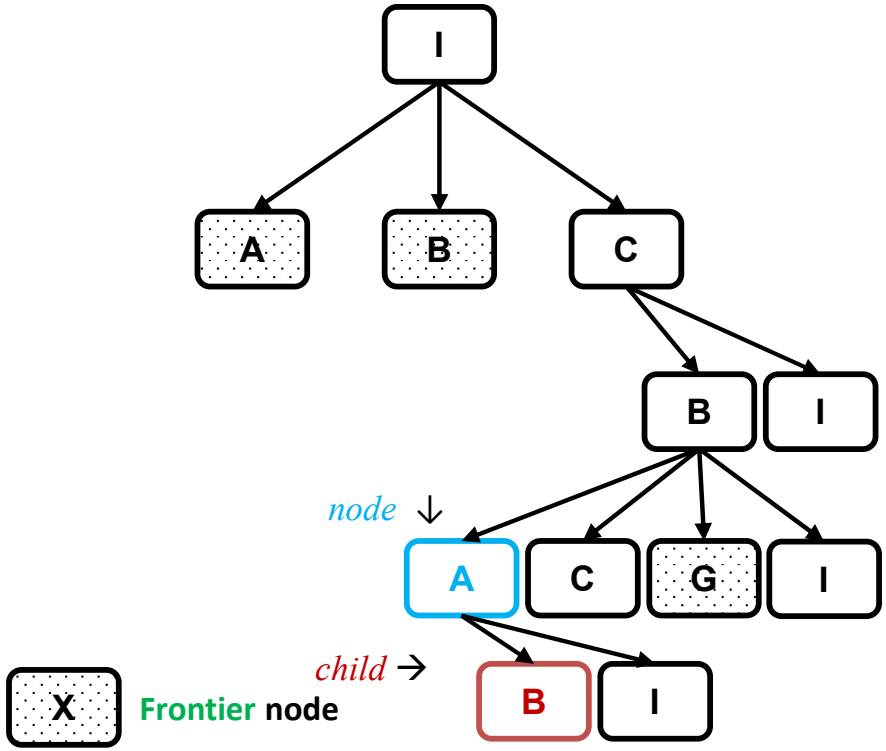
      s ← *child*.STATE

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

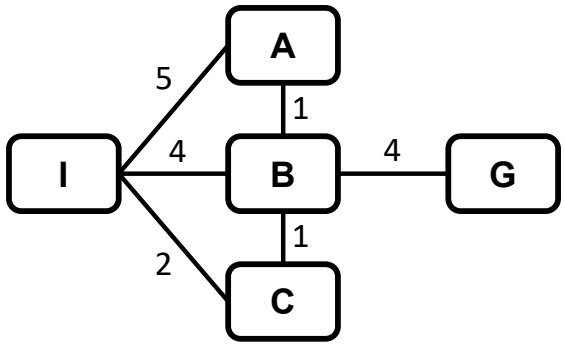
        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

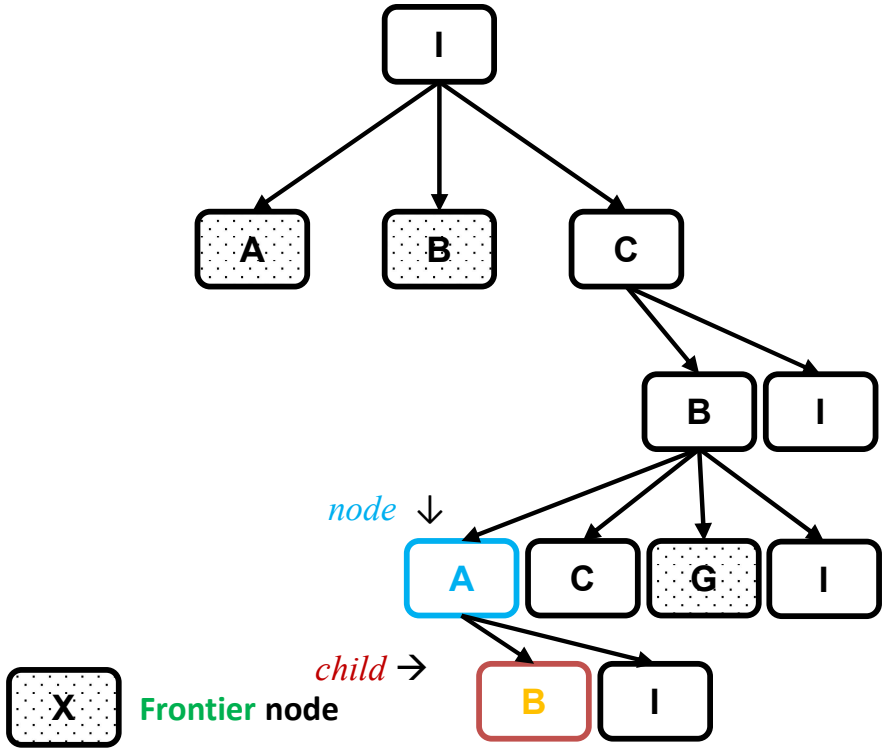
      s ← *child*.STATE

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

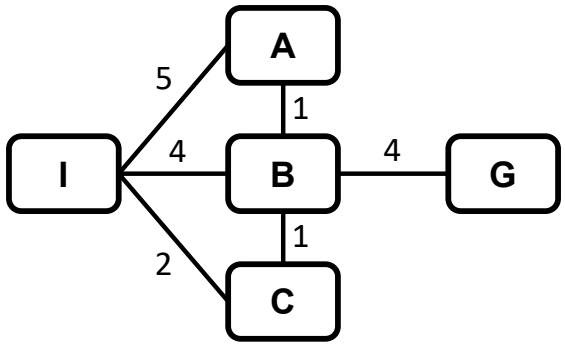
        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

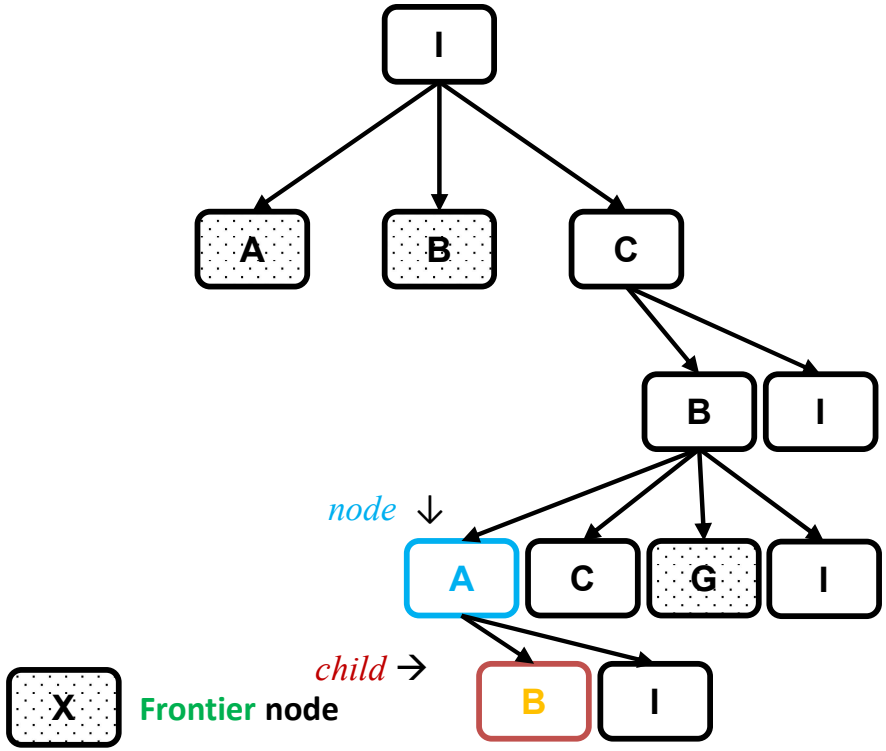
      s ← *child*.STATE

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

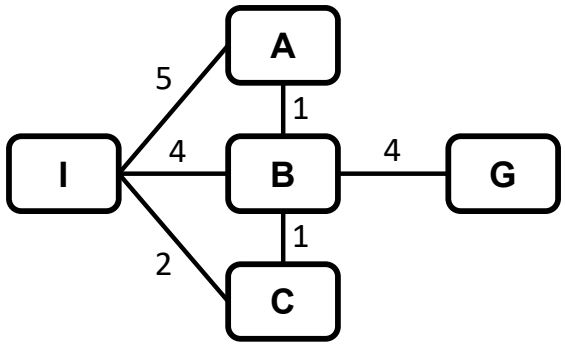
        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

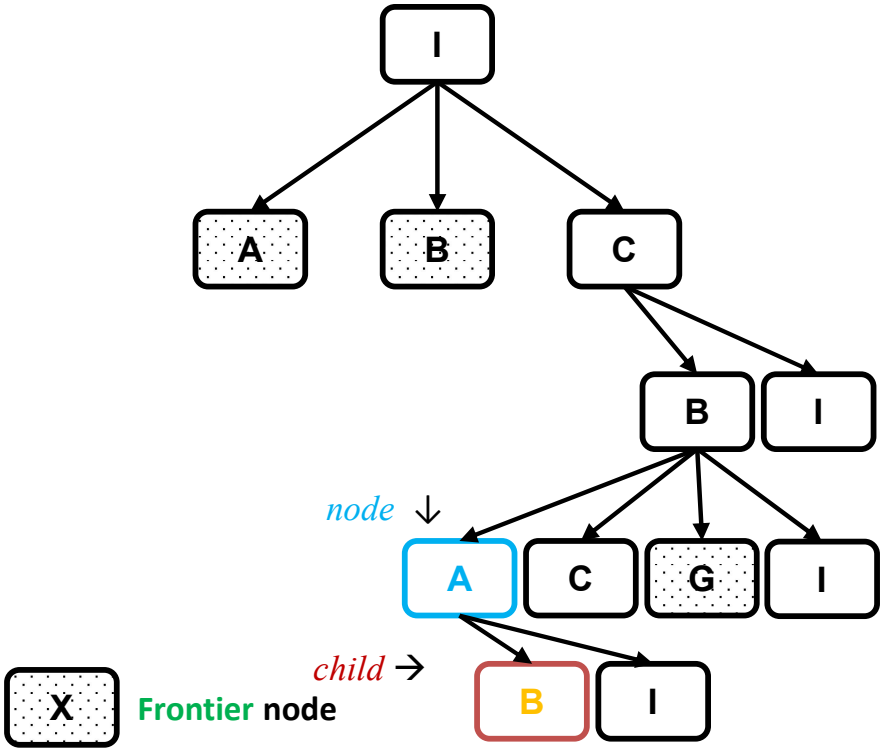
s ← *child*.STATE

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

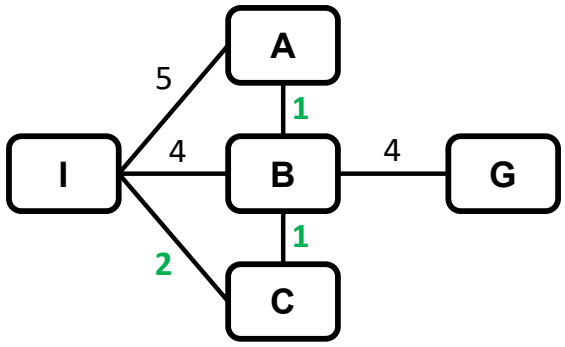
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	**3**	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

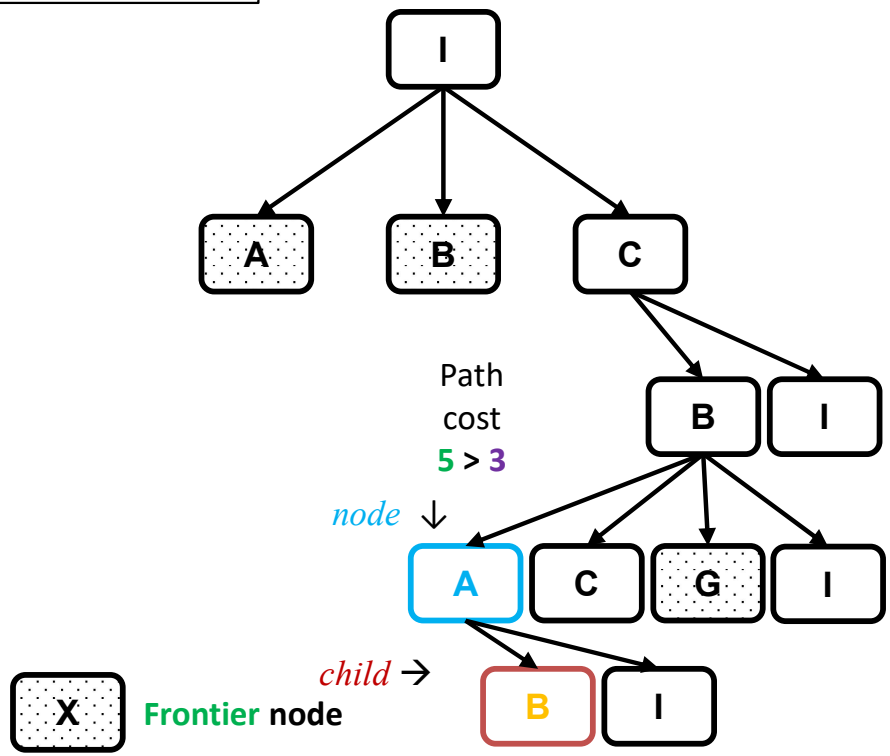
s ← *child*.STATE

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

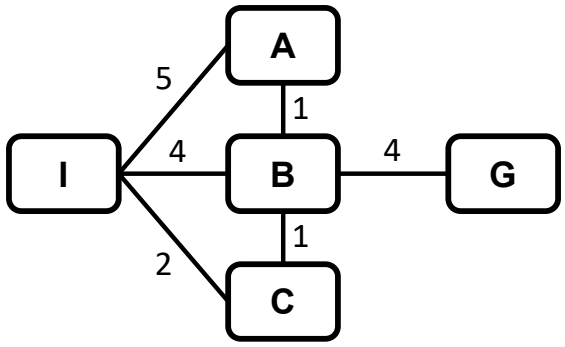
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

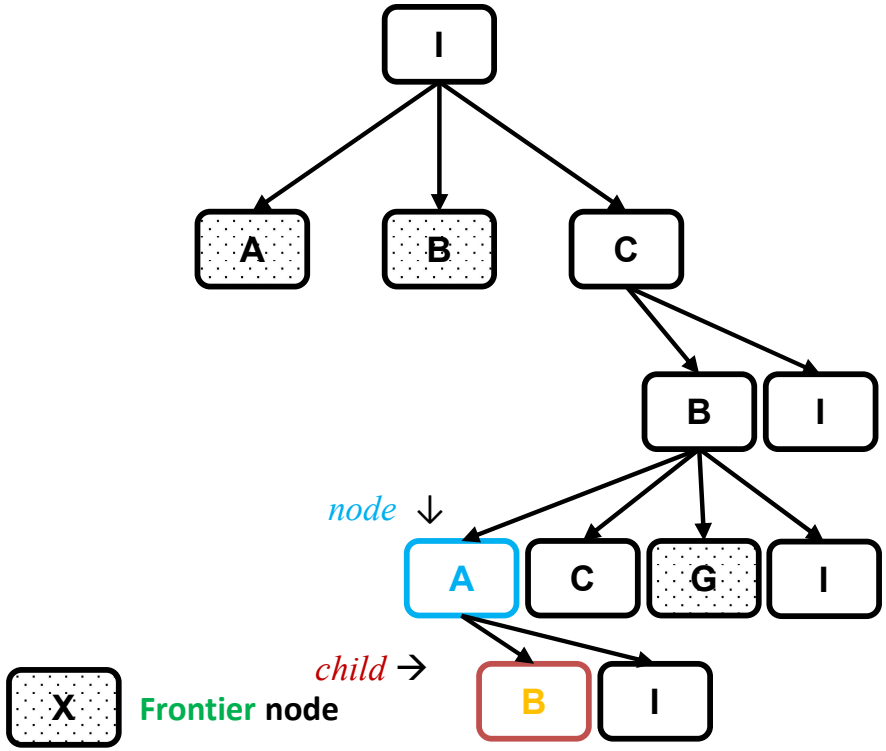
s ← *child*.STATE

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

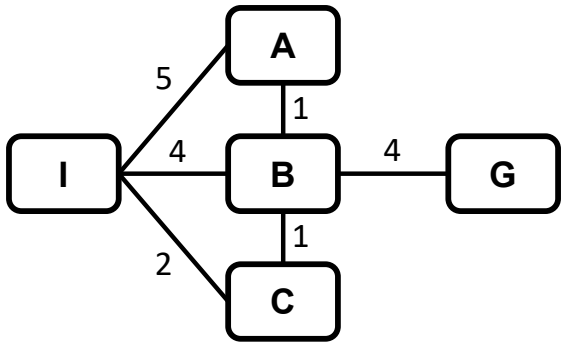
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

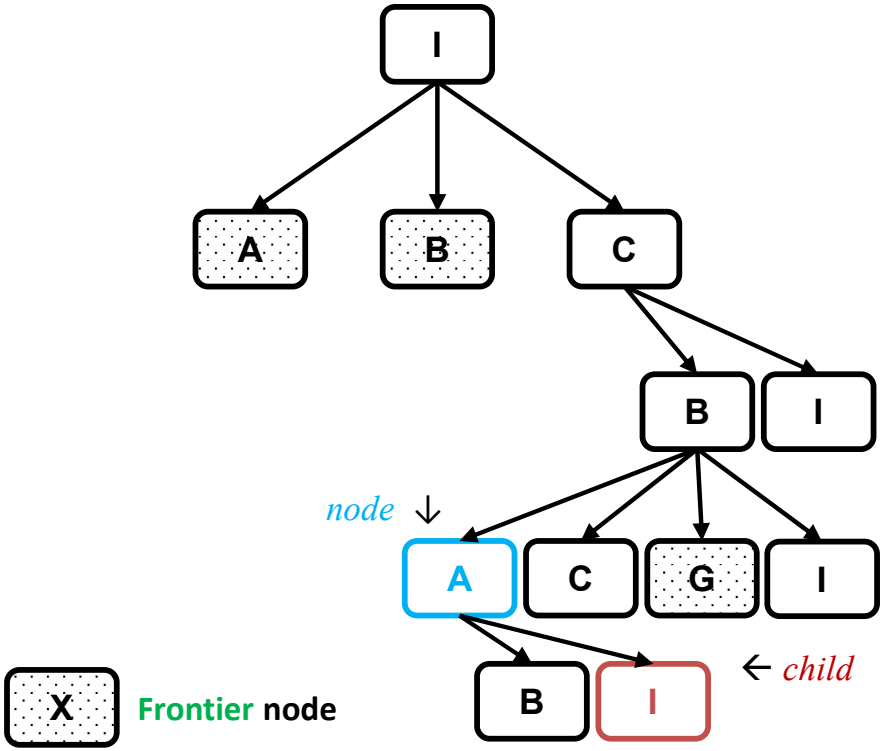
      s ← *child*.STATE

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

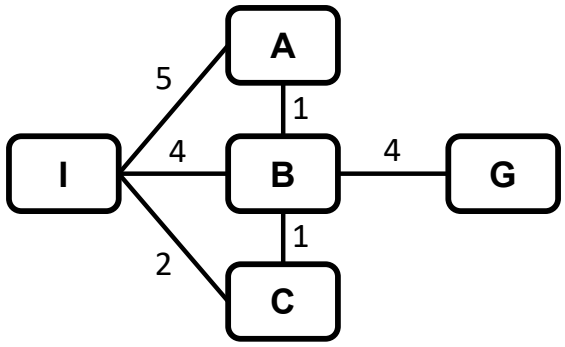
        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

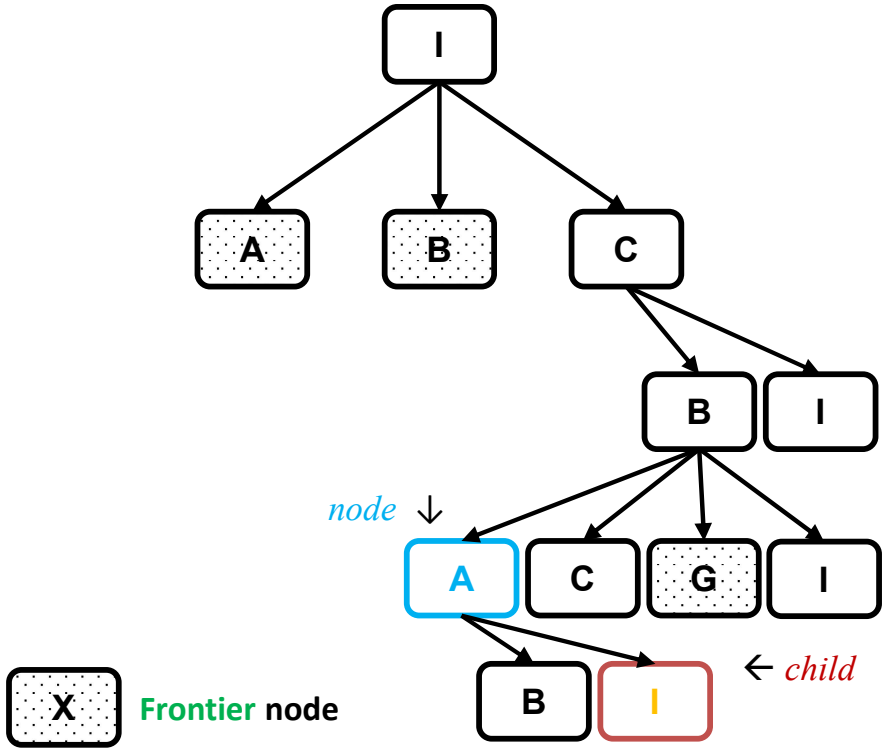
      s ← *child*.STATE

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

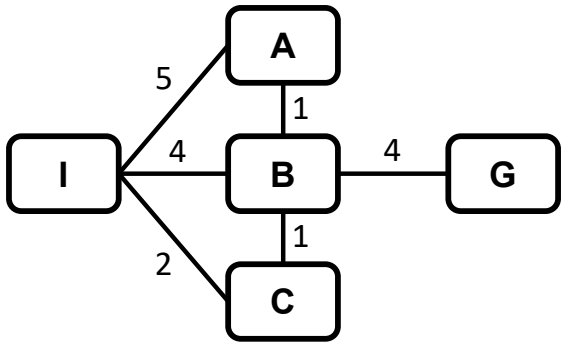
        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

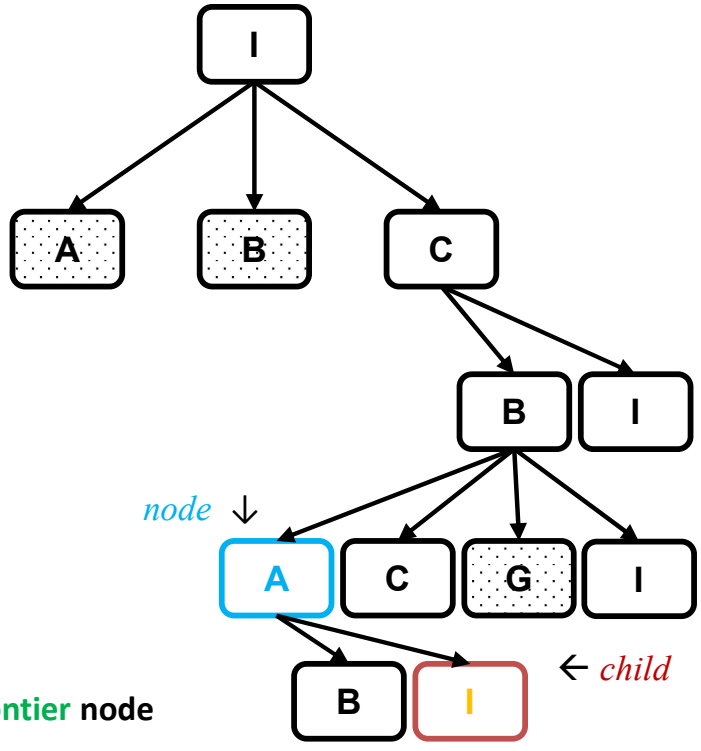
s ← *child*.STATE

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

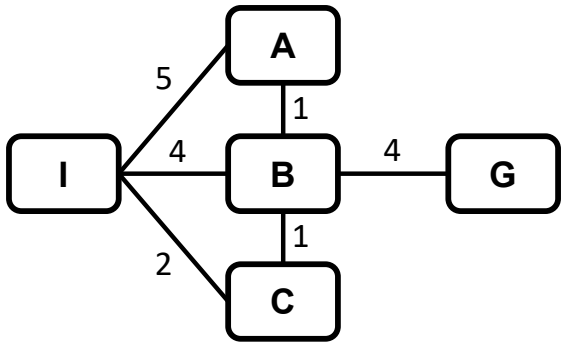
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

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

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

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

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

    node ← POP(*frontier*)

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

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

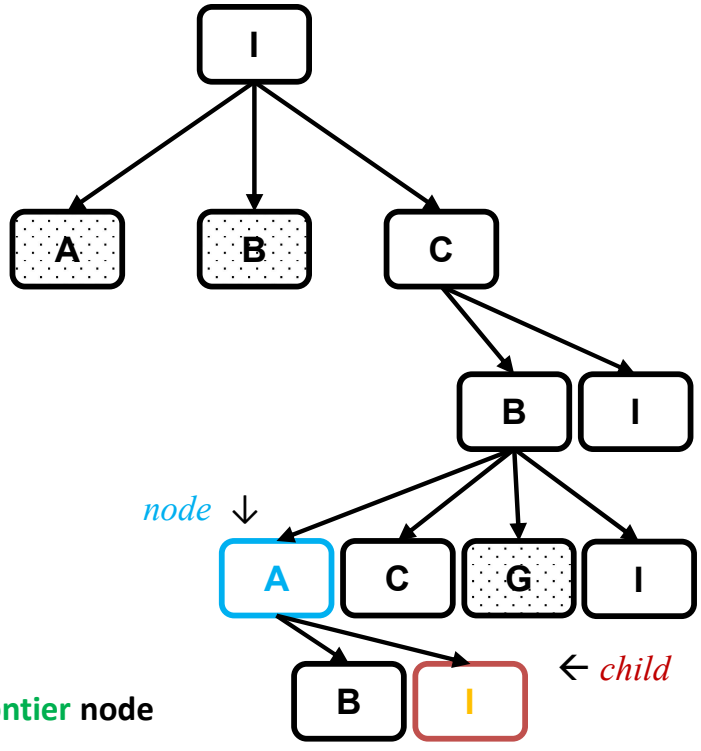
      s ← *child*.STATE

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

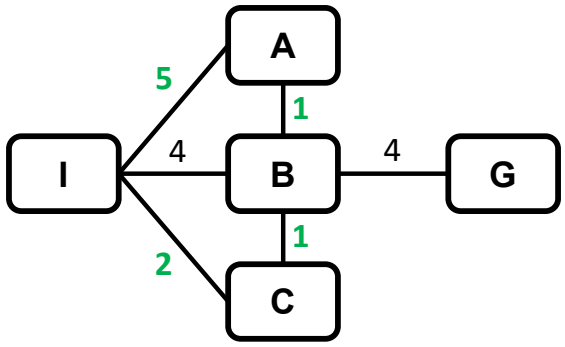
        reached[*s*] ← *child*

        add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	**0**	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

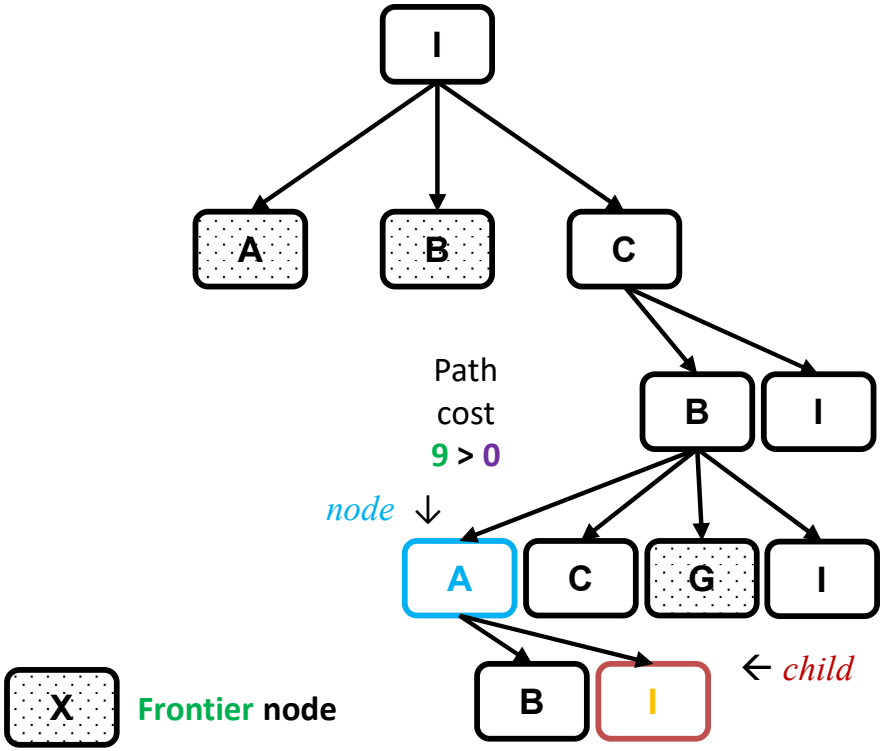
s ← *child*.STATE

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

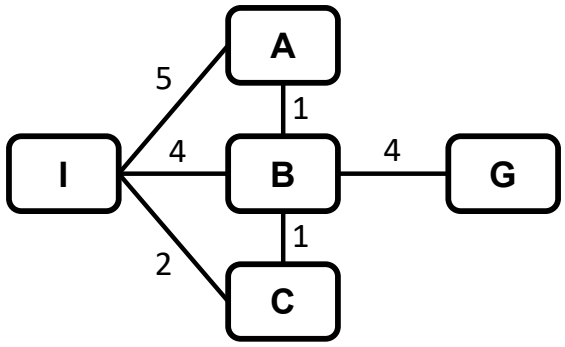
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

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

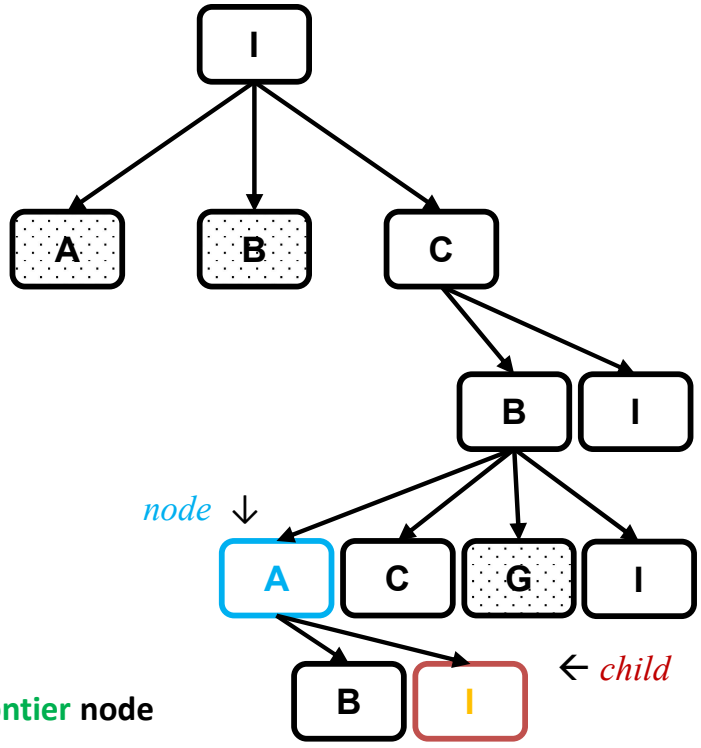
s ← *child*.STATE

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

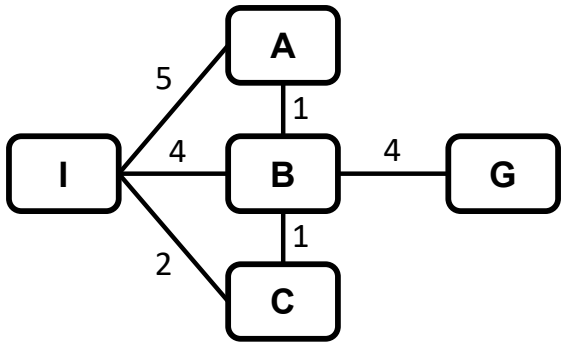
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



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

INITIAL STATE: I
GOAL STATE: G

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

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

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

node ← NODE(STATE=*problem*.INITIAL)

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

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

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

node ← POP(*frontier*)

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

for each *child* **in** EXPAND(*problem*, *node*) **do**

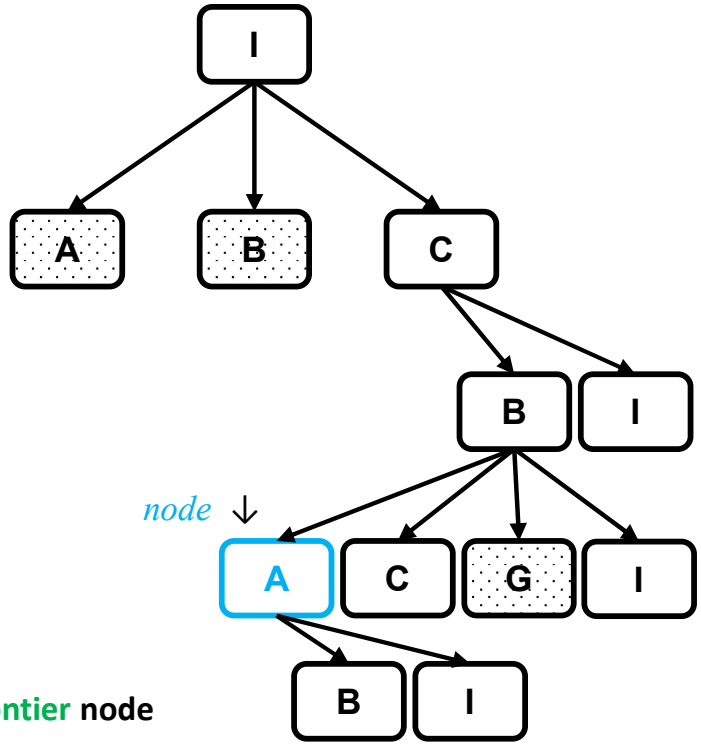
s ← *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

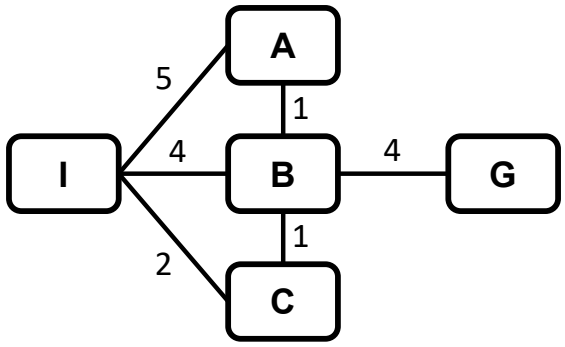
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	B	I	I				
	Node	G	B	A				
	f(Node)	7	7	7				

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

function BEST-FIRST-SEARCH(*problem*, f)

node ← NODE(STATE=*problem*.INITIAL)

frontier ← a priority queue ordered by *f*, with *node* as an element

reached ← a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do** **NOT EMPTY!**

node ← POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node*

for each *child* **in** EXPAND(*problem*, *node*) **do**

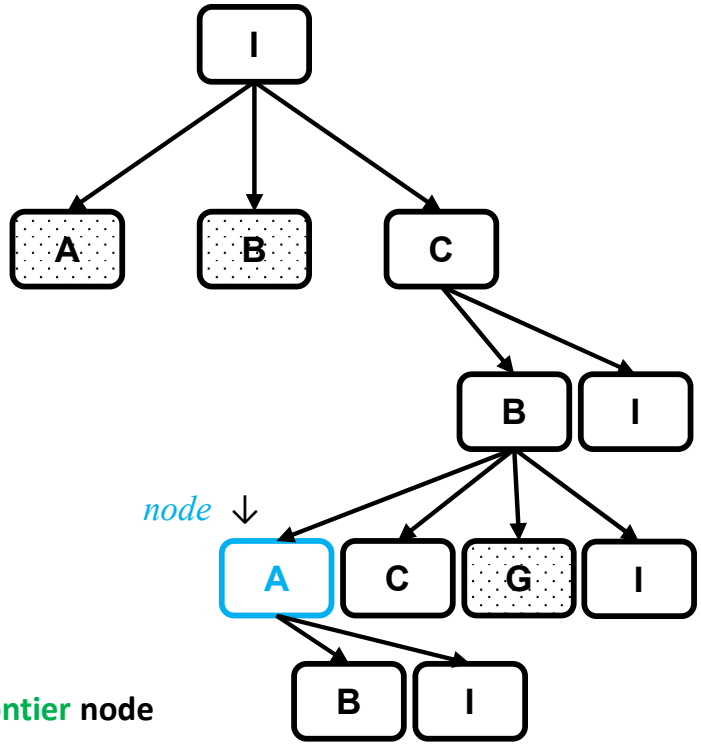
s ← *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

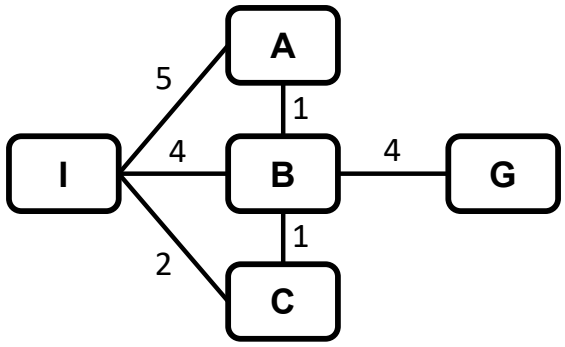
reached[*s*] ← *child*

add *child* to *frontier*

return *failure*



A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

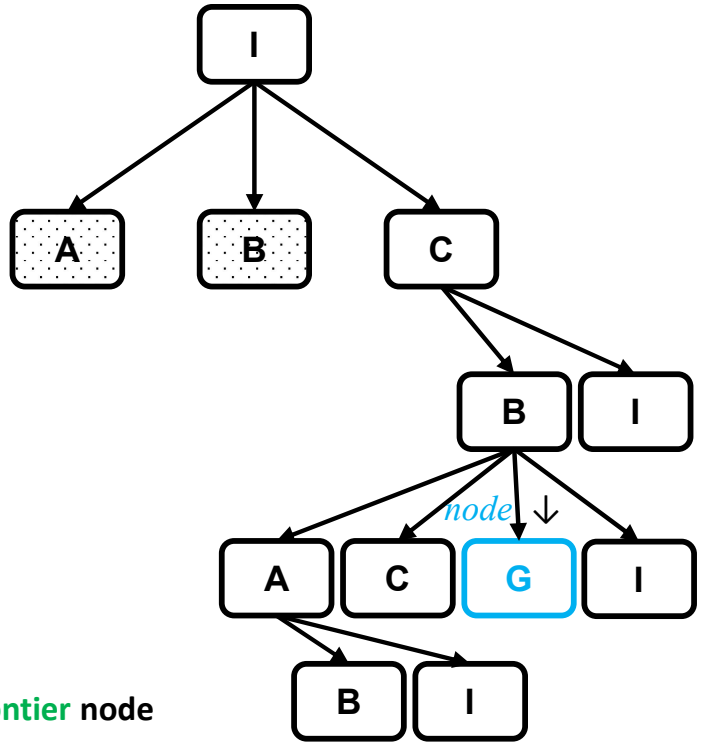
INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I	I					
	Node	B	A					
	f(Node)	7	7					

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

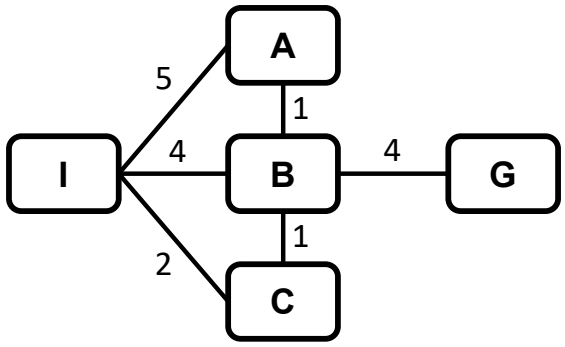
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```



X **Frontier node**

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

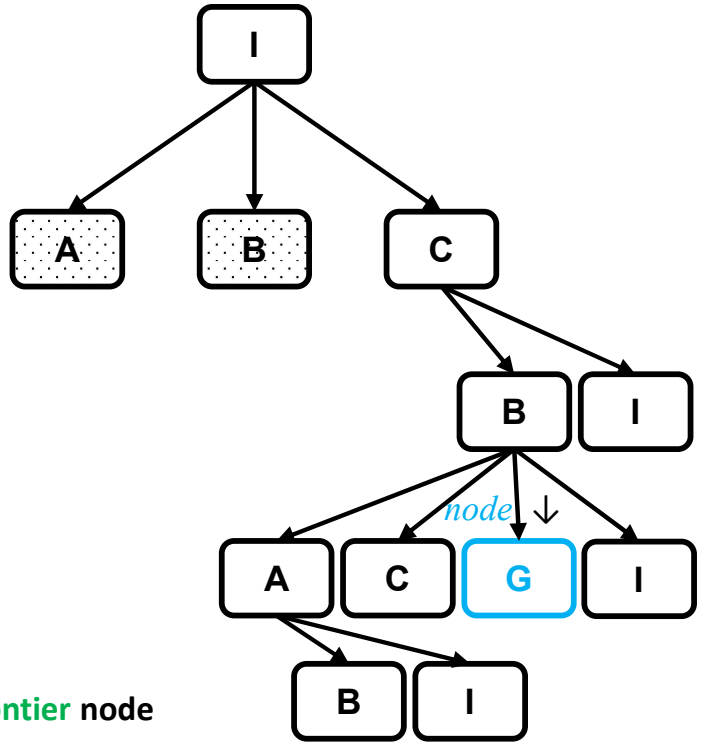
INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I	I					
	Node	B	A					
	f(Node)	7	7					

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

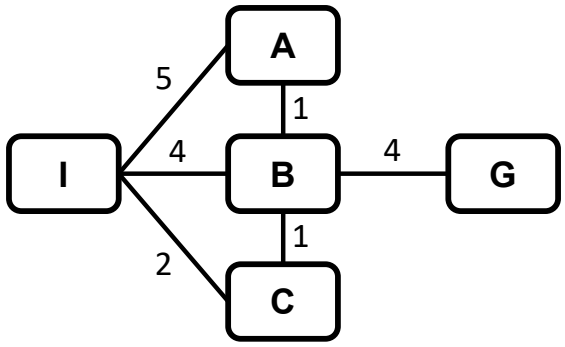
State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```



X **Frontier node**

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I	I					
	Node	B	A					
	f(Node)	7	7					

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

function BEST-FIRST-SEARCH(*problem*, f)

node ← NODE(STATE=*problem*.INITIAL)

frontier ← a priority queue ordered by *f*, with *node* as an element

reached ← a lookup table, with one entry key *problem*.INITIAL and value *node*

while not IS-EMPTY(*frontier*) **do**

node ← POP(*frontier*)

if *problem*.IS-GOAL(*node*.STATE) **then return** *node* **TRUE!**

for each *child* **in** EXPAND(*problem*, *node*) **do**

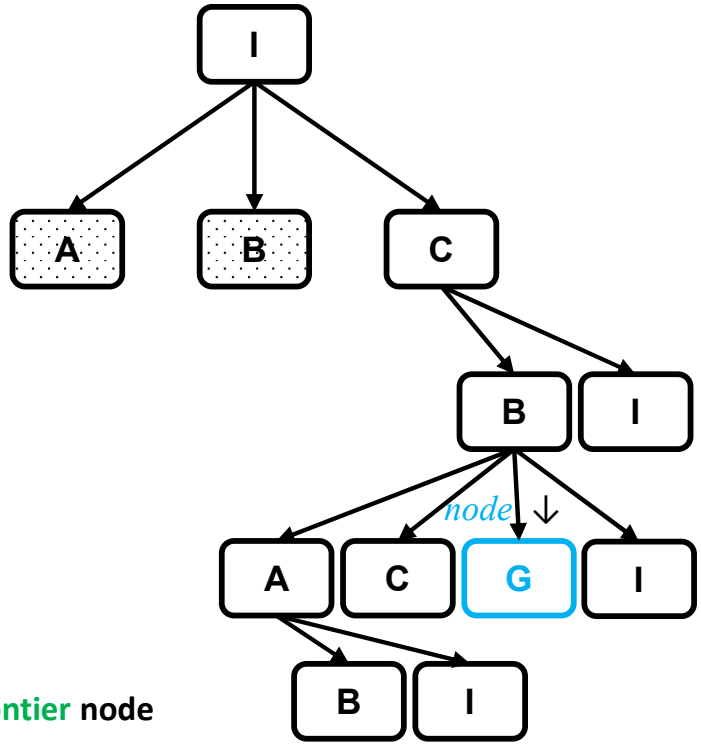
s ← *child*.STATE

if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**

reached[*s*] ← *child*

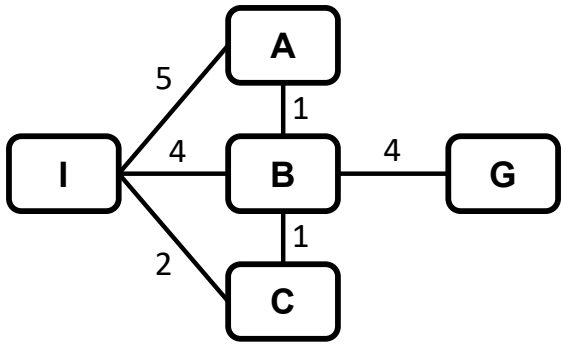
add *child* to *frontier*

return *failure*



X **Frontier node**

A* Search: Example



Straight-line distance to Goal state					
State	I	A	B	C	G
h(State)	7	2	3	4	0

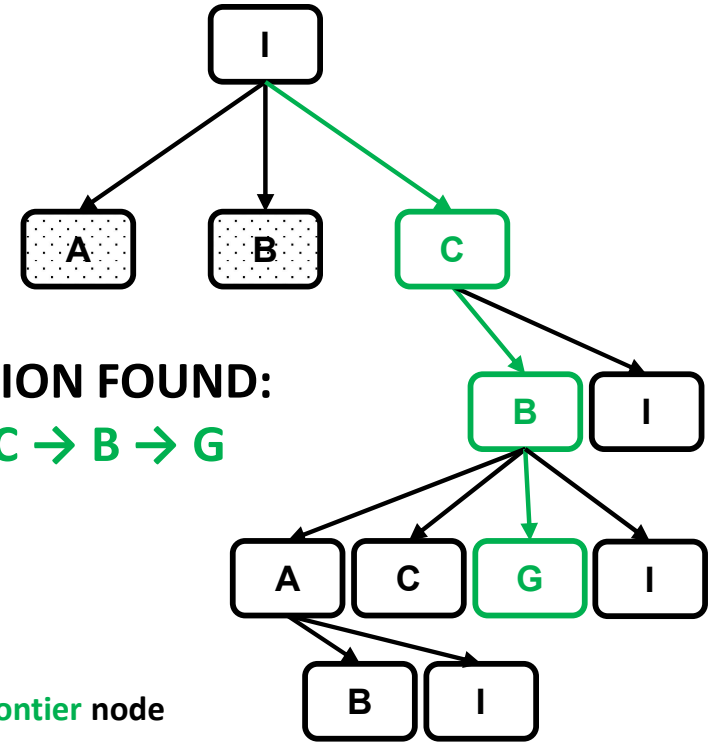
INITIAL STATE: I
GOAL STATE: G

Frontier								
	Parent	I	I					
	Node	B	A					
	f(Node)	7	7					

Reached								
	Parent	----	B	C	I	B		
	Key/State	I	A	B	C	G		
	Path cost	0	4	3	2	7		

State Space **Graph**	Frontier / Reached
Algorithm	**Search Tree**

```
function BEST-FIRST-SEARCH(problem, f)
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node TRUE!
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```



SOLUTION FOUND:
I → C → B → G

X **Frontier node**