**Terms**

* **Node:** A node is a data structure that represents a node in the search tree. The search tree is not the same thing as the (tree or graph) of the search space. A node has
  + a state
  + a parent (pointer/reference to the node in the tree that generated this one)
  + an action (the action that was applied to the parent's state to generate this node; often can be omitted from this data structure)
  + path-cost: the total cost of the path from the initial state to this node (aka g(node) or g(n))
* **Frontier**: The data structure that holds nodes we have yet to expand, usually sorted by f(n) via priority queue, though can be a stack or plain queue as well.
* **Reached**: a map/dictionary that stores which states have been "reached" (have had nodes generated for them).

**Best-first-search algorithm**

BEST-FIRST-SEARCH(problem, f)

node 🡨 a new node corresponding to the initial state

frontier 🡨 a priority queue of nodes ordered by f(n), initialized to contain only node

reached 🡨 a map from states to nodes with one entry mapping the initial state to the node above

while not IS-EMPTY(frontier):

node 🡨 pop(frontier) // remove lowest cost node from frontier (smallest f)

if IS-GOAL(node.state), then return node

for each child in EXPAND(node):

s 🡨 child.state

if s is not in reached or child.path-cost < reached[s].path-cost:

reached[s] 🡨 child

add child to frontier

return failure  
  
EXPAND(node) // returns a list or set of nodes

make an empty list or set to hold the child nodes

s 🡨 node.state

for each action in ACTIONS(s):

s' 🡨 RESULT(s, action)

cost 🡨 node.path-cost + ACTION-COST(s, action, s')

add new Node(state=s', parent=node, action=action, path-cost=cost) to list or set of child nodes

return the list or set of child nodes

**Breadth-first search**

Text

Description automatically generated