Infrastructure needed

- A node n of the search tree stores:
  - a state (of the state space)
  - a parent pointer to a node (usually)
  - the action that got you from the parent to this node (sometimes)
  - the path cost g(n): cost of the path *so far* from the initial state to n.
- Frontier is often stored as a stack, queue, or priority queue.
- Explored set is often stored using a data structure that enables quick look-up for membership tests.

Uninformed search methods

- These methods have no information about which nodes are on promising paths to a solution.
- Also called: *blind search*
- Question — What would have to be true for our agent to need uninformed search?
  - No knowledge of goal location; or
  - No knowledge of current location or direction (e.g., no GPS, inertial navigation, or compass)

How do you evaluate a search strategy?

- **Completeness** — Does it always find a solution if one exists?
- **Optimality** — Does it find the best solution?
- **Time complexity**
- **Space complexity**

function TREE-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

loop do

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

expand the chosen node, adding the resulting nodes to the frontier

Frontier = stack, queue, or priority queue.

---

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to the frontier

only if not in the frontier or explored set

Explored set = hash table.

Search strategies

- Breadth-first search
  - Variant — Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening depth-first search
  - Variant — iterative lengthening search

Breadth-first search

- Choose shallowest node for expansion.
- Data structure for frontier?
  - Queue (regular)
- Suppose we come upon the same state twice. Do we re-add to the frontier?
  - No.
- Complete? Optimal? Time? Space?

Uniform-cost search

- Choose node with lowest path cost g(n) for expansion.
- Data structure for frontier?
  - Priority queue
- Suppose we come upon the same state twice. Do we re-add to the frontier?
  - Yes. (And remove old node from frontier.)
- Complete? Optimal? Time? Space?

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Best-first search
(class of algorithms)

- Same algorithm as uniform-cost search.
- Uses a different evaluation function to sort the priority queue.
- Need a heuristic function, h(n).
  - h(n) = Estimate of lowest-cost path from node n to a goal state.

A* Algorithm

- Sort priority queue by a function f(n), which should be the *estimated* lowest-cost path through node n.
- What is f?
  - f(n) = g(n) + h(n)

Heuristics

- A heuristic function h(n) is ***admissible*** if it never over-estimates the true lowest cost to a goal state from node n.
- Equivalent: h(n) must always be less than or equal to the true cost from node n to a goal.
- What happens if we just set h(n) = 0 for all n?

Heuristics

- A heuristic function h(n) is ***consistent*** if values of h(n) along any path in the search tree are non-decreasing.
- Equivalent: given a node n, and an action which takes you from n to node n':
  - h(n) <= cost(n, a, n') + h(n')
  - h(n) – h(n') <= cost(n, a, n')
- Consistency implies admissibility (but not the other way around).
- Difficult to invent heuristics that are admissible but not consistent.