Solving Problems by Searching

Problem-Solving Agent

- **Goal-based agents** take account of future actions and their related outcomes.
- **Problem-solving agents** are goal-based agents handling atomic representations.
- **Planning agents** handle factored or structured representations.

Environment Assumptions

- **Observable**: the current state is always known by the agent.
- D**iscrete**: a finite set of actions can be choose from a state.
- **Known:** the agent know the consequence of each action, which means to know the next state from the applied action.
- D**eterministic**: the action has only one well defined outcome.

Solution within the Environment

The solution is given by a sequence of actions which leads the agent to the goal state.

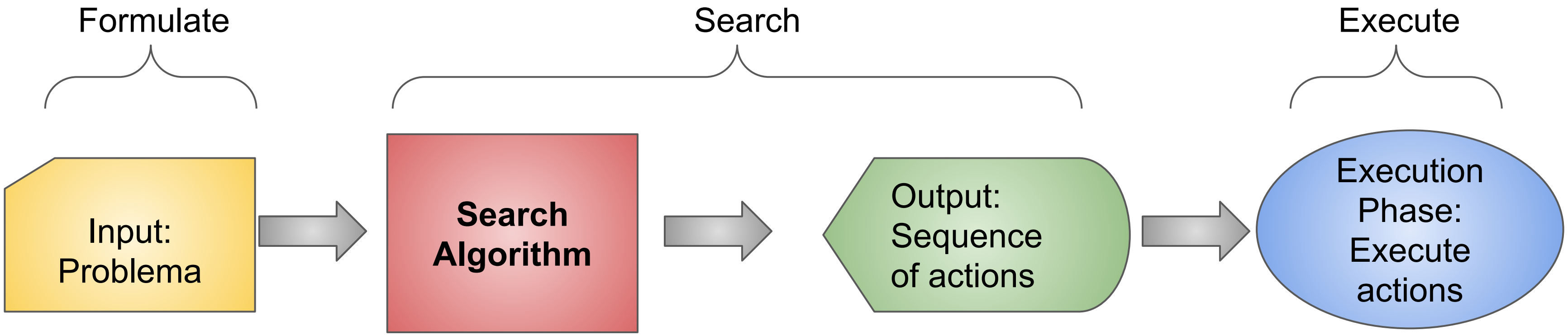
This sequence of action is a solution since:

- the environment is know and deterministic;
- the initial state of the agent is also known.

Thus, the solution will provide an action based agent perception from the environment. One action from one perception.

Search

Search can be defined as the steps to find a sequence of actions aiming the goal state.



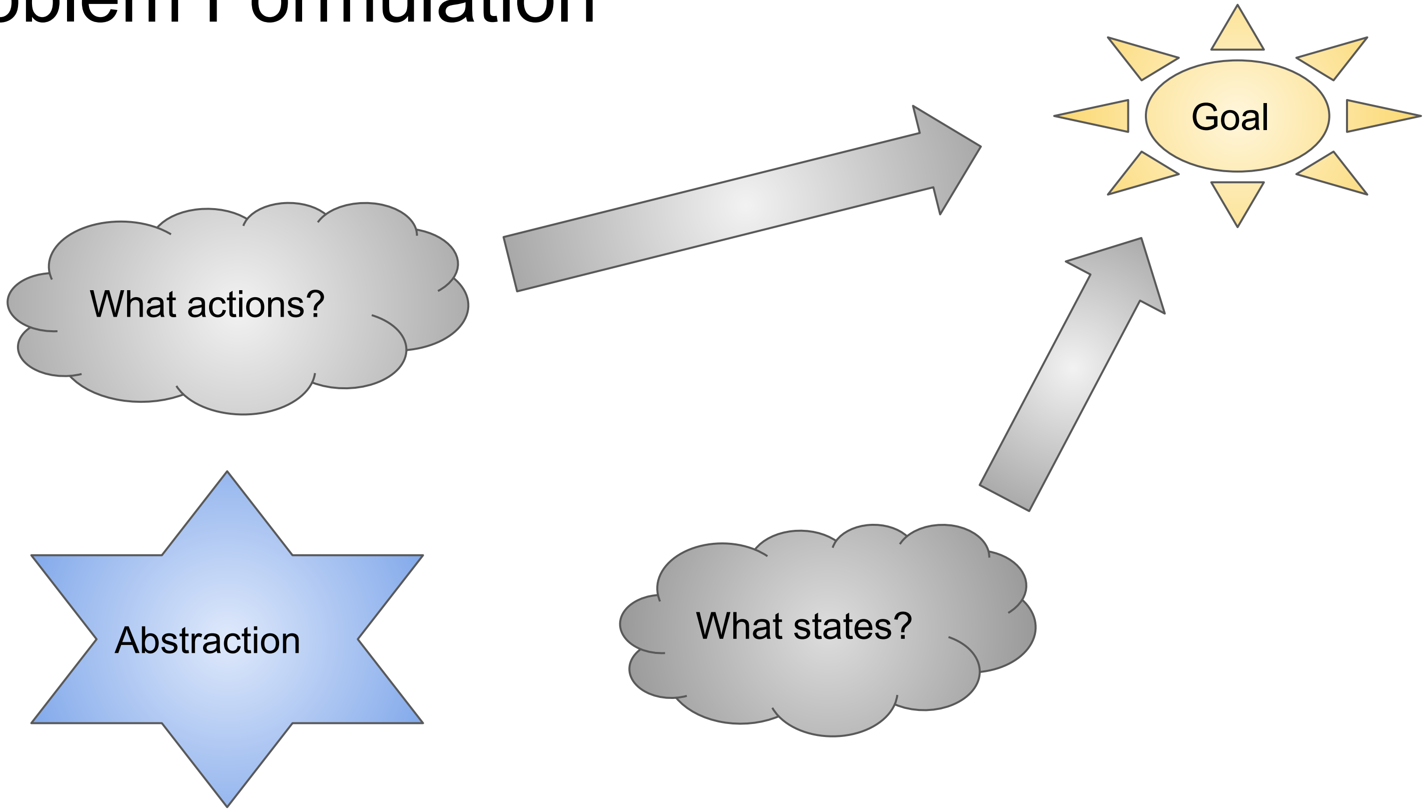
Problem-Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   persistent: seq, an action sequence, initially empty
               state, some description of the current world state
               goal, a goal, initially null
               problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH(problem)
       if seq = failure then return a null action
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Problem Formulation



Problem formulation

- **Initial state**: the state of the agent starts in the beginning.
- **Actions**: set of actions applicable in the current state.

  ACTIONS(s): return possible actions applicable in state s.

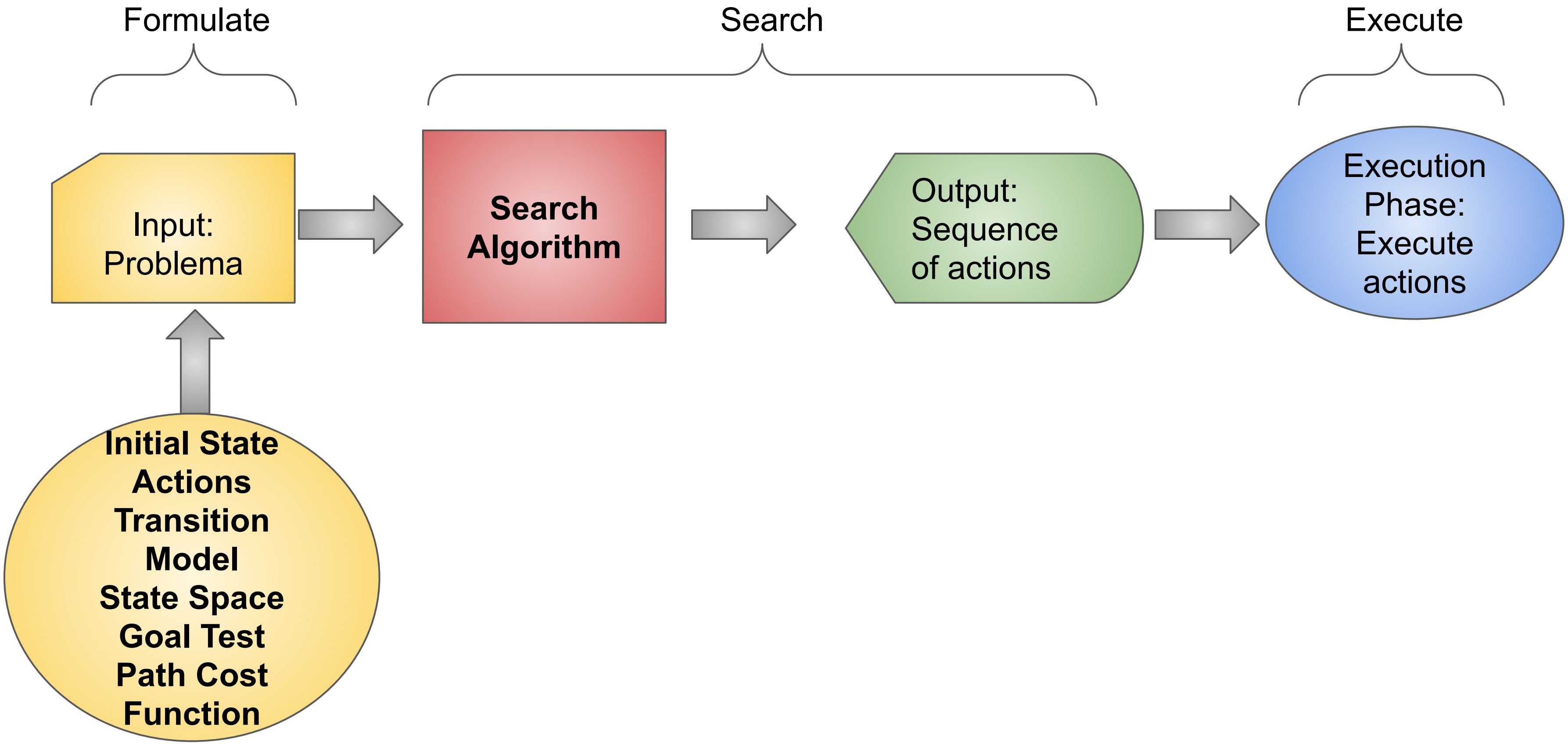
- **Transition model**: define the consequence of the each action application over the state.

  RESULT(s, a): returns the state achieve after applying action a over state s.

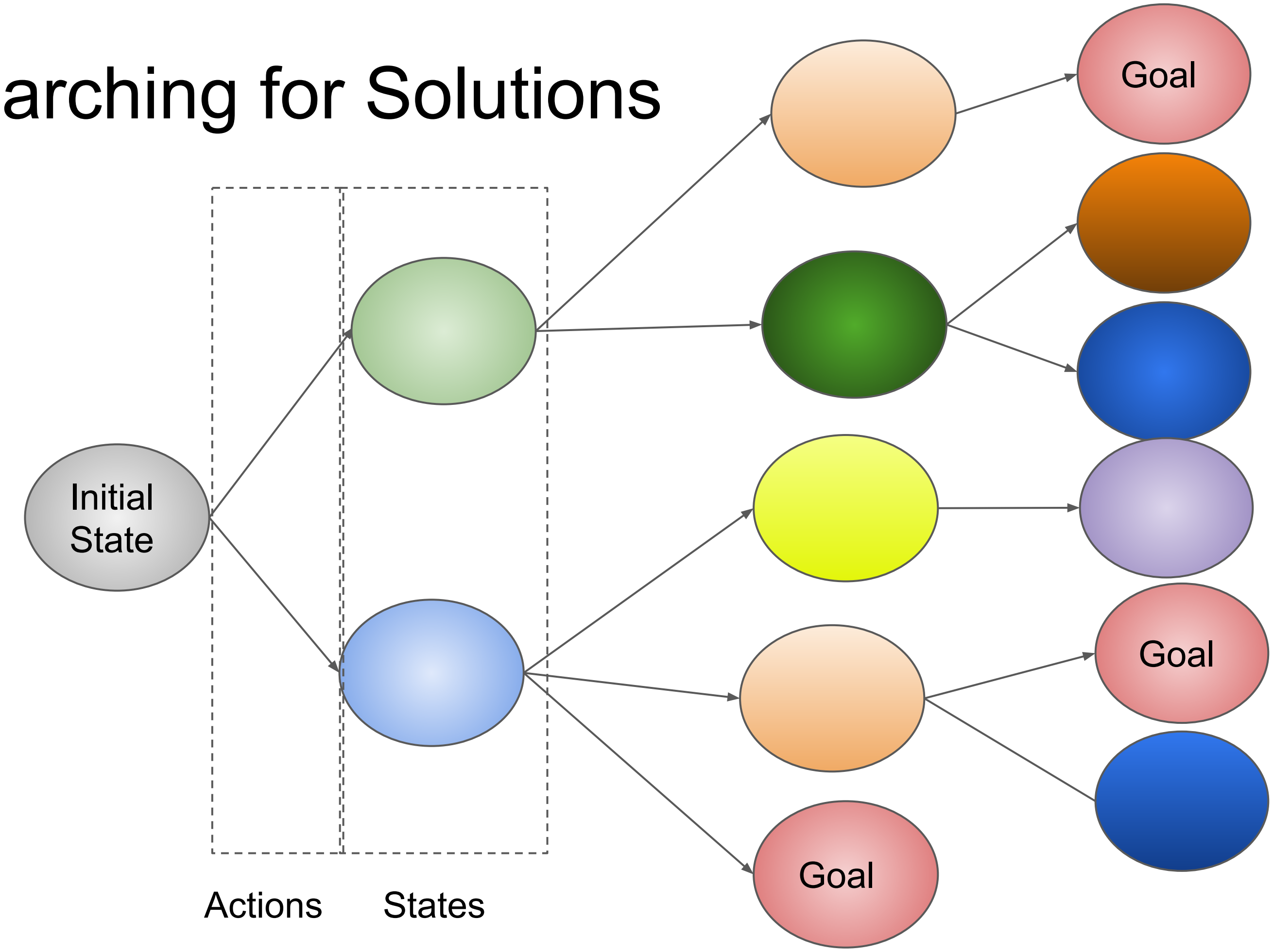
Problem formulation

- **State space**:
  - It is defined by the initial state, actions, and transition model.
  - It has all states which can be obtained by applying a sequence of actions.
- **Goal test**: verify when a goal state is reached.
- **Path cost function:** measures the solution quality by evaluating each path cost.
  - The path is the sequence of states within the state space provided by a sequence of actions
  - c(s,a,s'): step cost to reach state s' from state s by applying action a.
  - The optimal solution will have the lowest path cost.

Problem Formulation



Searching for Solutions



Searching for Solutions

```
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
```

```
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
```

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Searching for Solutions

The node can be coded to provide the following attributes:

- State: current state within the state space.
- Parent: the previous node from which the current node was expanded.
- Action: the selected action applied to the parent.
- Path cost: the cost to reach the current node from the root node (initial state).

function CHILD-NODE(*problem*, *parent*, *action*) **returns** a node
  return a node with
    STATE = *problem*.RESULT(*parent*.STATE, *action*),
    PARENT = *parent*, ACTION = *action*,
    PATH-COST = *parent*.PATH-COST + *problem*.STEP-COST(*parent*.STATE, *action*)

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Searching for Solutions

A search algorithm performance can be evaluated based on some measures:

- Completeness: if there is solution, the algorithm will find one.
- Optimality: the optimal solution is found by such algorithm.
- Time complexity: define how long the algorithm takes to return a solution.
- Space complexity: define the amount of memory consumption demanded by the algorithm to execute the search process.

Searching for Solutions

The complexity is evaluated from:

- The depth of the goal node with minimum depth.
- Maximum length of a path in the state space.
- Maximum number of successors (branching factor).

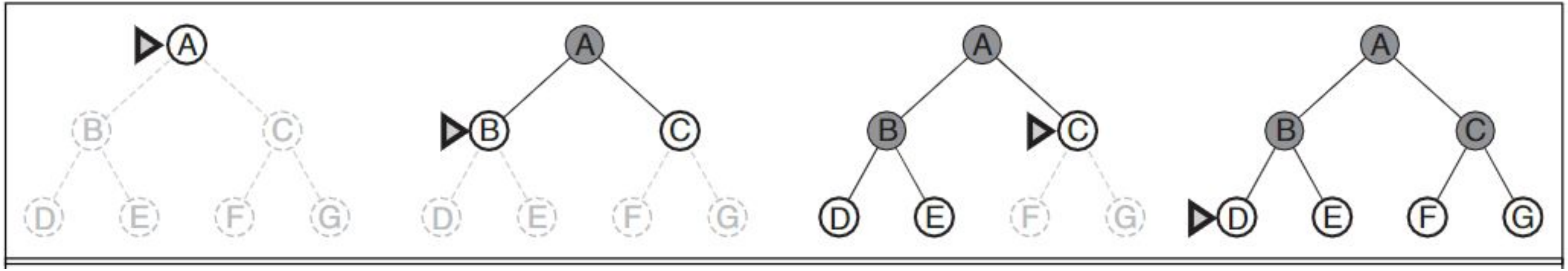
Time ⇔ number of nodes created.

Space ⇔ number of nodes stored in memory.

Uninformed Search Strategy

Generate states and identify if it is not a goal-state.

There is no other information provided by the problem.



Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Uninformed Search Strategy

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the shallowest node in frontier */
    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
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Uninformed Search Strategy

Breadth-first search is optimal when:

- Path cost is a nondecreasing function of the depth of the node since it always expands the shallowest node.

The uniform-cost search is optimal for any step-cost function:

- The node with lowest path cost is always expanded.
- The method requires to store the frontier as queue ordered by the path cost.
- A node selected for expansion is goal tested.
- Another evaluation takes place, if a better path is found to a node in frontier.

Uninformed Search Strategy

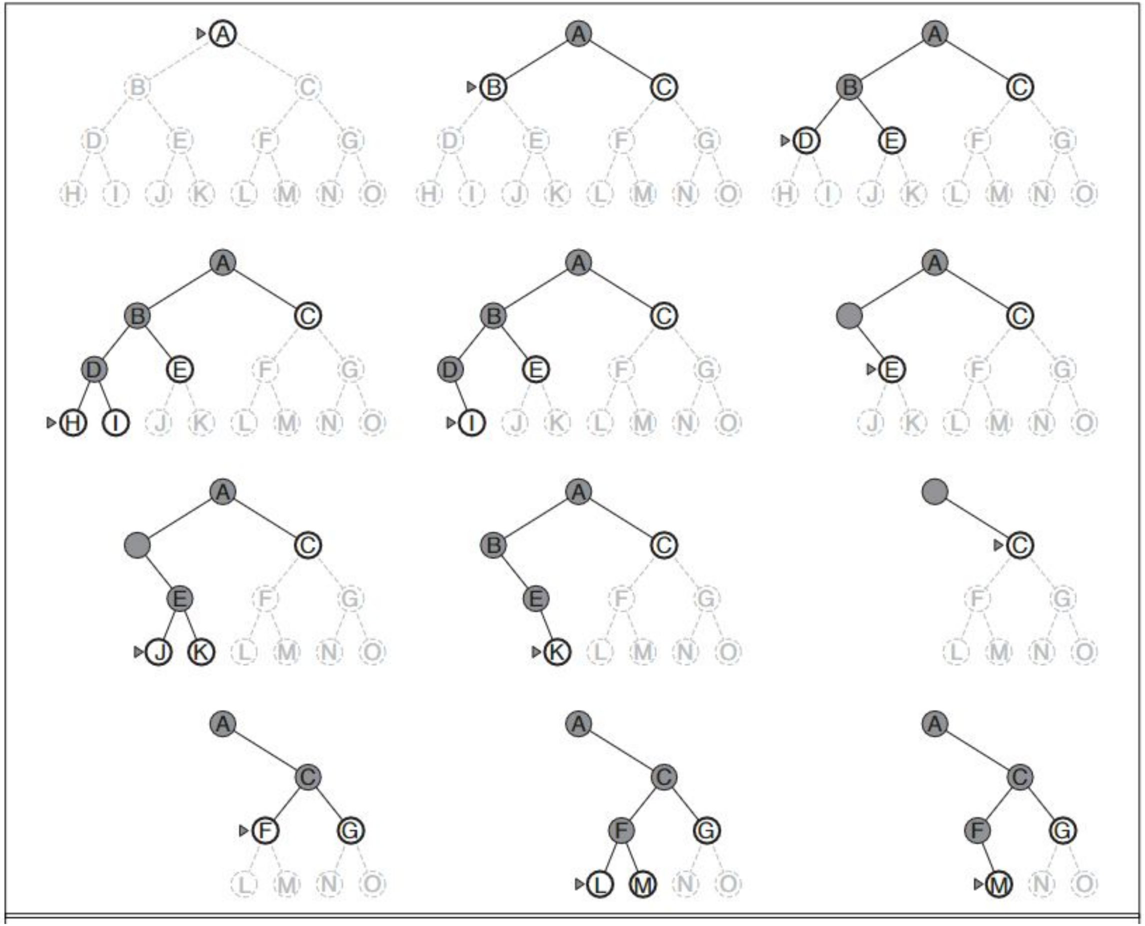
The deepest node in the frontier is expanded.

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Uninformed Search Strategy



Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Uninformed Search Strategy

Criterion	Breadth-First	Uniform-Cost	Depth-First
Complete?	Yes[a]	Yes[a,b]	No
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$
Optimal?	Yes[c]	Yes	No

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Informed Search

Problem information is handled by the search strategy to find solutions.

Best-first search:

- It has an evaluation function to estimate the cost at node n, f(n).
- It is similar to uniform-cost search, but the nodes in the queue are ordered following the evaluation function.
- Depth-first search is a special case.
- There may be a heuristic function h(n) that estimates the minimum cost to reach the goal state from node n.

Informed Search

Greedy best-first search: it expands first the node with best value of h(n), which means, f(n)=h(n).

A*: evaluates the nodes using f(n)=g(n)+h(n), where g(n) is to path-cost. .