Problem-Solving Agents

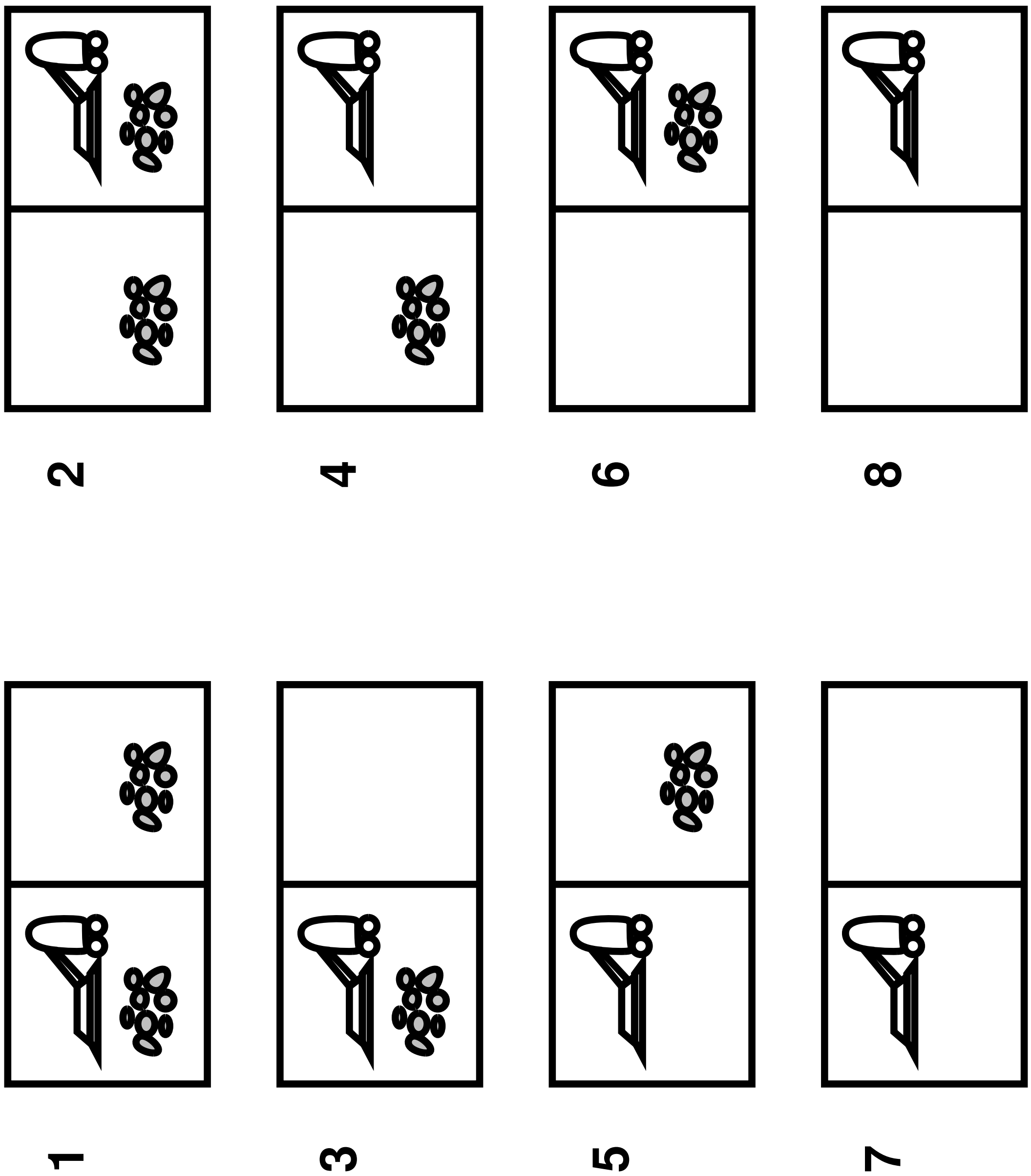
- Key ideas:
  - To determine what the best action is, a problem-solving agent systematically considers the expected outcomes of different possible sequences of actions that lead to some goal state.
  - A systematic search process is conducted in some representation space.
- Problem-solving agents are often goal-directed.
- Steps:
  1. Goal formulation
  2. Problem formulation
  3. Search process
  4. Action execution

A Simple Problem-Solving Agent

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

   state ← UPDATE-STATE(state, p)
   if s is empty then
      g ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, g)
      s ← SEARCH(problem)
   action ← RECOMMENDATION(s, state)
   s ← REMAINDER(s, state)
   return action
```

Problem Formulation: An Illustrative Example



Four Problem Types

- Single-state problems:
  - Accessible world state
  - Known outcome of action
- Multiple-state problems:
  - Inaccessible (incompletely accessible) world state – agent only knows which set of states it is in
  - Known outcome of action
- Contingency problems:
  - Action depends on what is found to be true through perception
  - Interleaving of search and execution
- Exploration problems:
  - No information is known about world state and outcome of action
  - Learning through experimentation is often needed.

Problem Definition

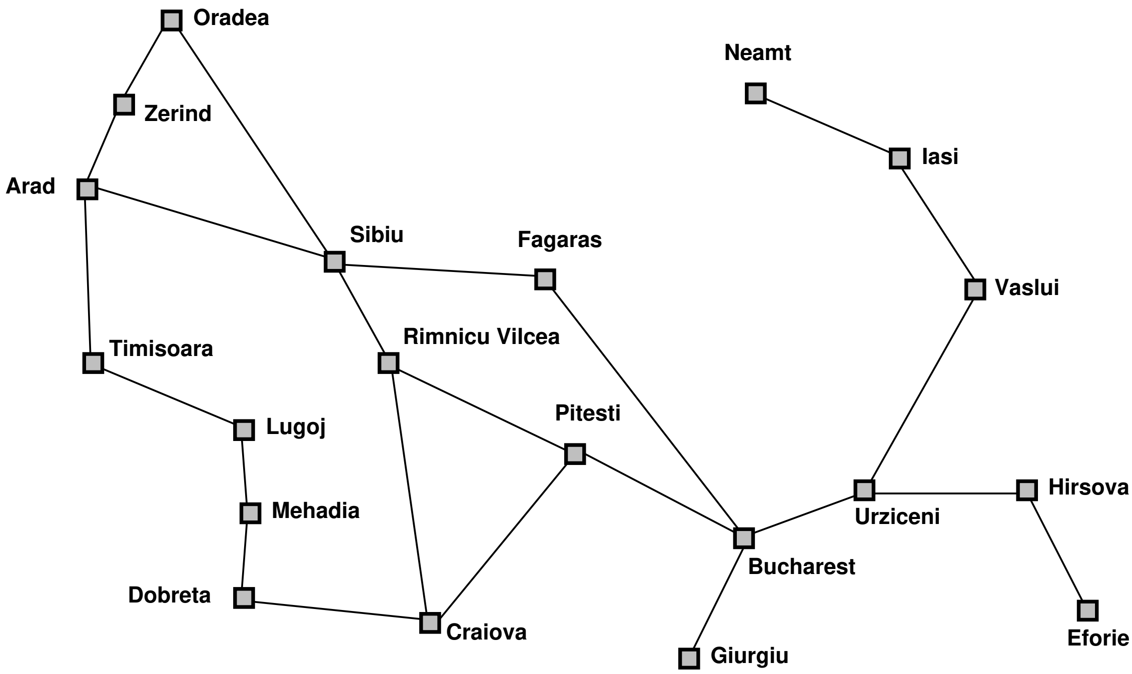
- For single-state problems:
  1. Initial state
  2. Set of possible actions (or operators) and the corresponding next states
  3. Goal test
  4. Path cost function
- For multiple-state problems:
  1. Initial state set
  2. Set of possible actions (or operators) and the corresponding sets of next states
  3. Goal test
  4. Path cost function

  A solution is a path (connecting sets of states) that leads to a set of states all of which are goal states.

Square Apples Problem

- 27 square apples packed into a 3x3x3 cube.
- A worm in the center apple.
- It can move to another apple through a common side.
- each can be visited once.
- *Can the worm eat through all the apples?*

A Route Finding Problem

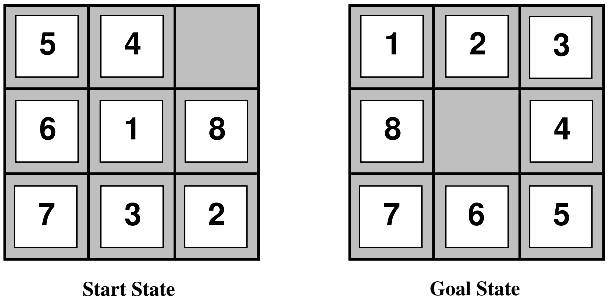


Measuring Problem-Solving Performance

- Correctness:
  Can a solution be found?
- Search cost (off-line cost):
  Is the solution good in terms of the cost (time and memory) of finding *the solution?*
- Path cost (on-line cost):
  Is the solution good in terms of the cost of executing *the solution?*
- Total cost = search cost + path cost (tradeoff is often needed)

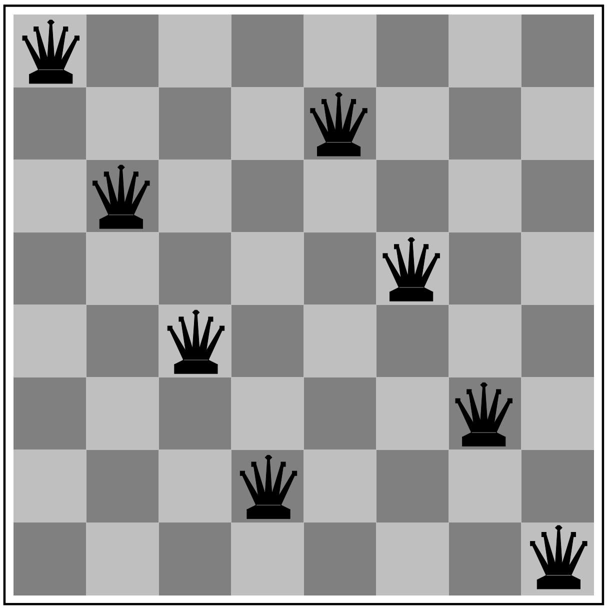
Some Toy Problems

- 8-puzzle:



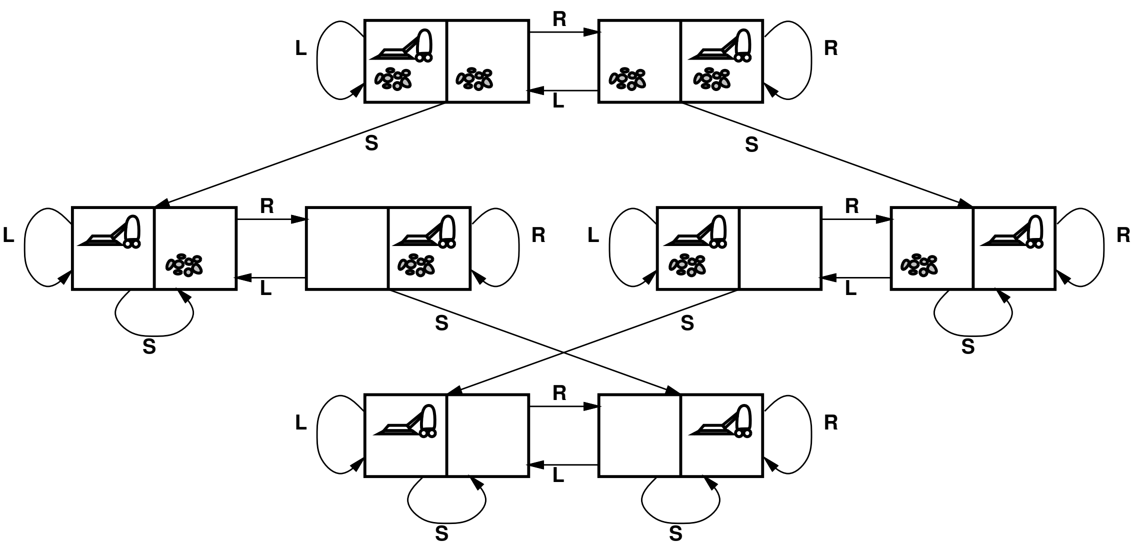
  - Number of states = 9!
  - Number of operators = 4
  - Path cost = length of path

- 8-queens problem:

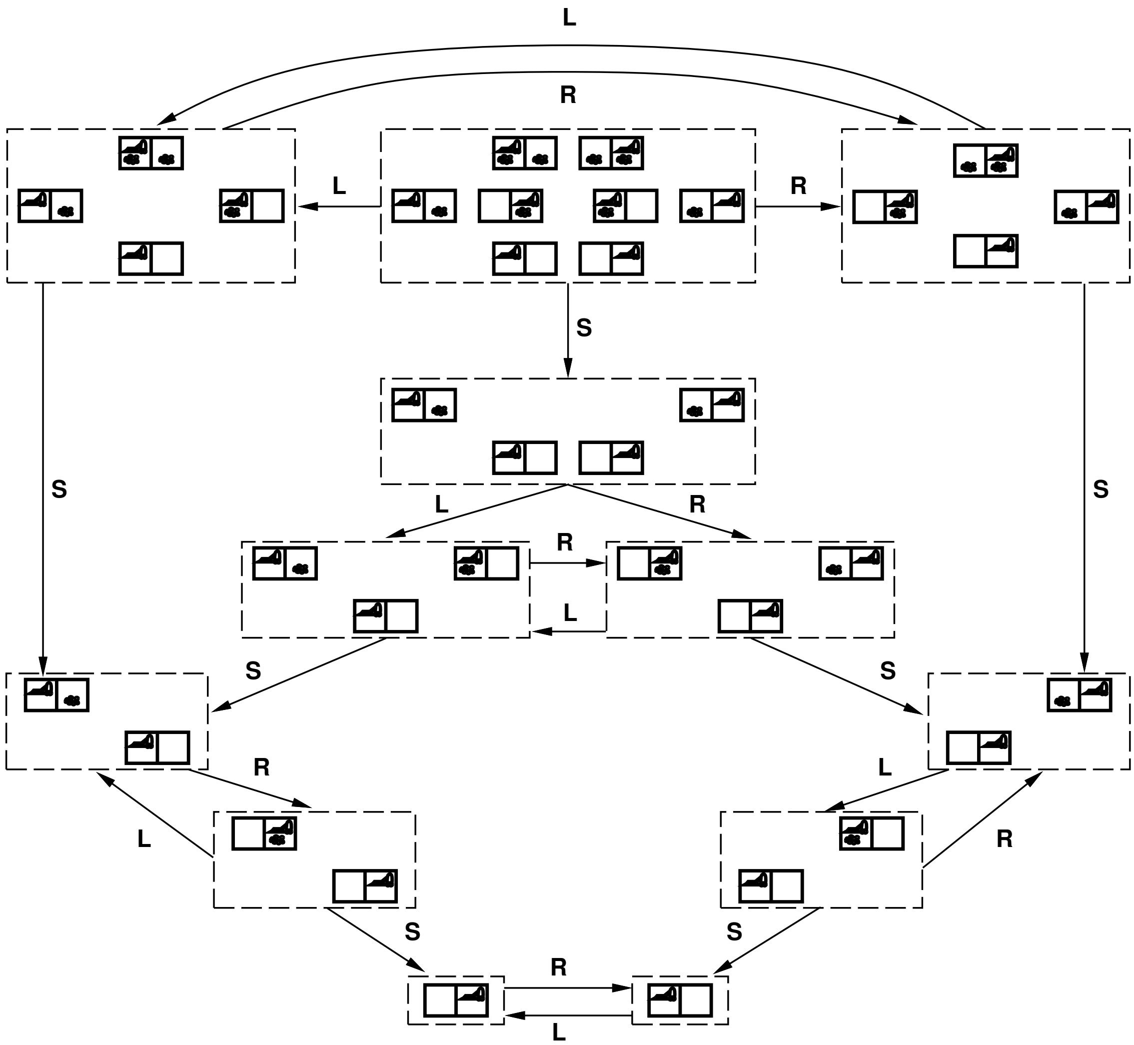


  - Goal test: 8 queens on board, none attacked
  - Path cost: zero (algorithms are compared only on search cost)

Vacuum World: A Single-State Version



Vacuum World: A Multiple-State Version



General Search Algorithm

function GENERAL-SEARCH(*problem, strategy*) **returns** a solution, or failure
    initialize the search tree using the initial state of *problem*
    loop do
        if there are no candidates for expansion **then return** failure
        choose a leaf node for expansion according to *strategy*
        if the node contains a goal state **then return** the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end

Data Structures for Search Trees

- Assume a node to be a data structure with five components:
  - the state to which the node represents.
  - the parent node.
  - the operator that generates this node.
  - the depth of the node.
  - the path cost from the root to this node.
- Assume also that the collection of nodes to be expanded, called *fringe* or *frontier*, be implemented as a queue with the following operations:
  - MAKE-QUEUE(Element) creates a queue with the given elements.
  - EMPTY?(Queue) returns true only if there is no elements in the queue.
  - REMOVE-QUEUE(Queue) removes the element at the front of queue and returns it.
  - QUEUING-FN(Elements, Queue) inserts a set of elements into the queue.

General Search Algorithm (cont'd)

function General-Search(*problem,* Queuing-Fn) **returns** a solution, or failure

    nodes ← Make-Queue(Make-Node(Initial-State[*problem*]))
    loop do
        if *nodes* is empty **then return** failure
        node ← Remove-Front(*nodes*)
        if Goal-Test[*problem*] applied to State(*node*) succeeds **then return** *node*
        nodes ← Queuing-Fn(*nodes*, Expand(*node*, Operators[*problem*]))
    end

Search Strategies

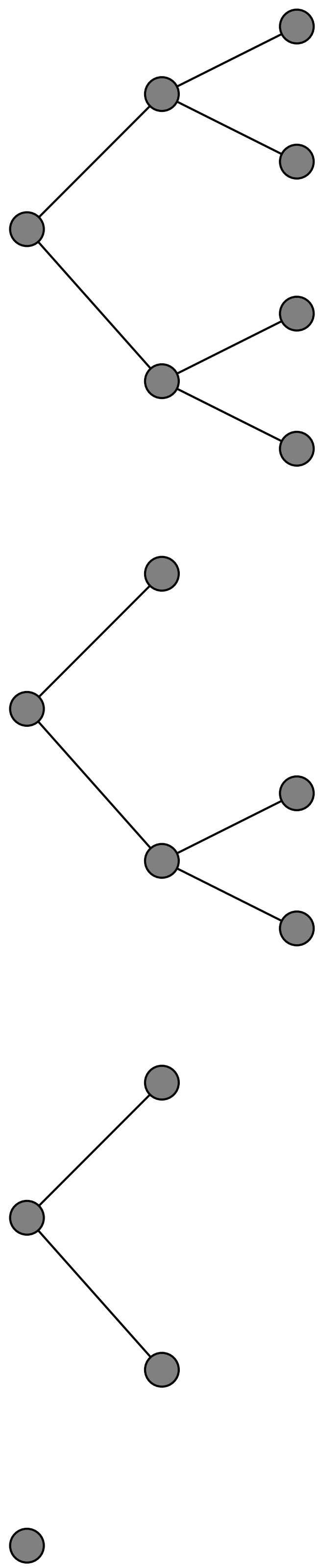
- The search strategy determines the order in which nodes in the search tree are expanded.
- Different search strategies lead to different search trees.
- Four different criteria that determine what the right search strategy is for a problem:
  - Completeness:
    Is it guaranteed to find a solution if one exists?
  - Time complexity:
    How long does it take to find a solution?
  - Space complexity:
    How much memory is needed?
  - Optimality:
    Does it find the "best" solution if several different solutions exist?
- Types:
  - Uninformed (or blind) search strategies
  - Informed (or heuristic) search strategies

  Informed search strategies are usually more efficient, if problem-specific knowledge is available to guide the search.

Uninformed Search Strategies

- Breadth-first search
- Uniform-cost search:
  - Node with least cost so far is expanded.
- Depth-first search
- Depth-limited search:
  - Depth-first search is performed with limit enforced on search depth.
- Iterative deepening search:
  - Depth-limited search is repeated with increasing depth limits.
- Bidirectional search:
  - Possible only if all operators are reversible.

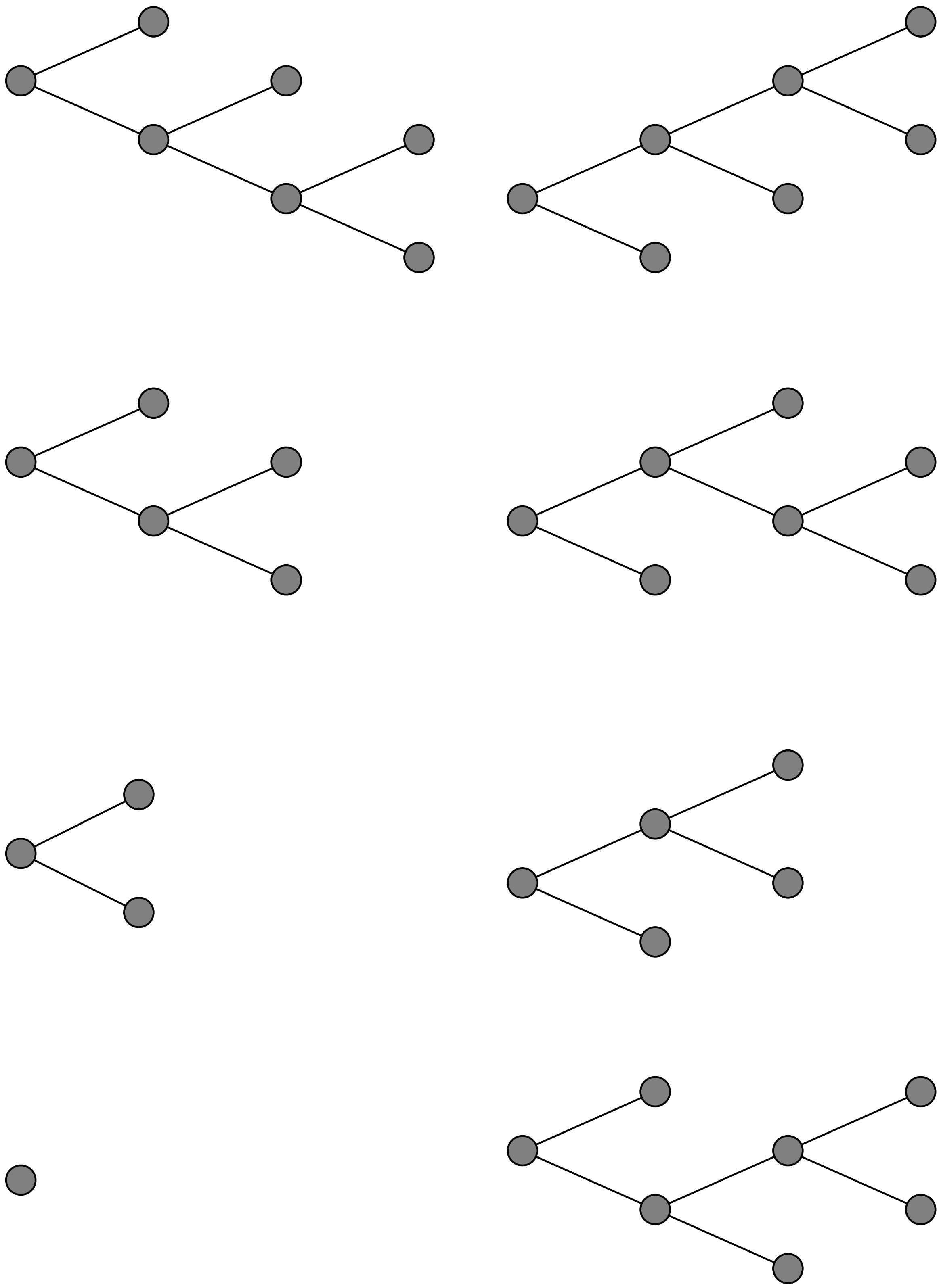
Breadth-first Search



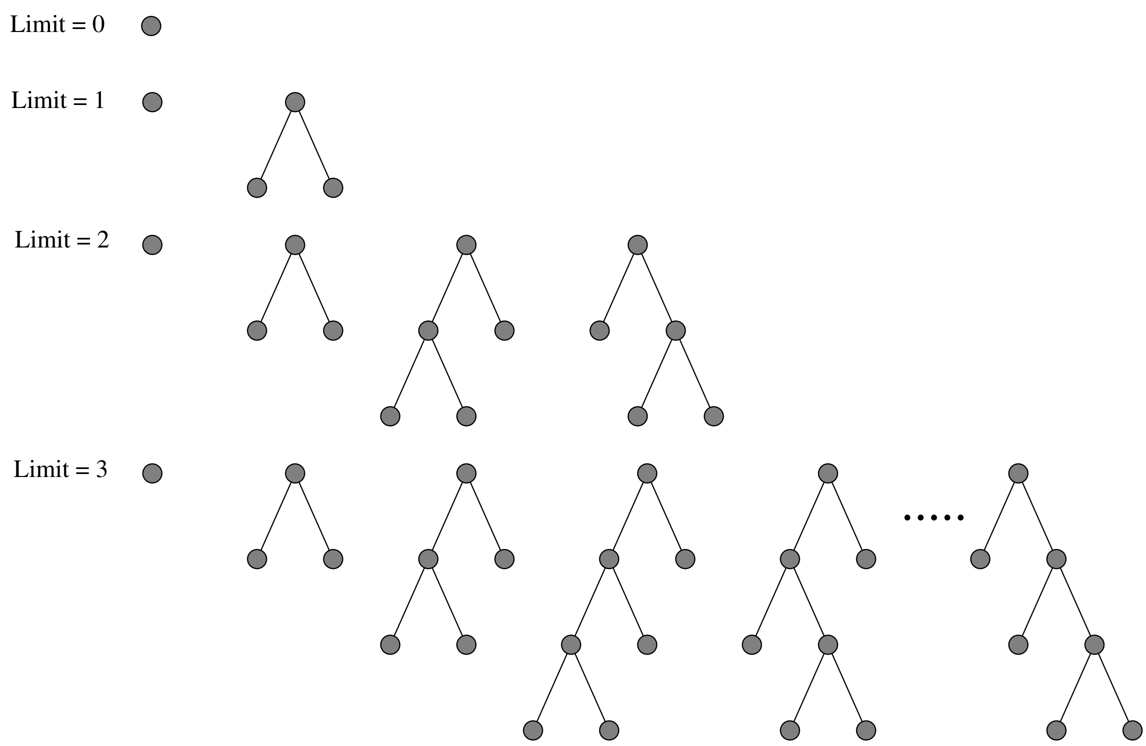
Time and Memory Requirements for Breadth-First Search

Depth	Nodes	Time		Memory	
0	1	1	millisecond	100	bytes
2	111	.1	seconds	11	kilobytes
4	11,111	11	seconds	1	megabyte
6	10^6	18	minutes	111	megabytes
8	10^8	31	hours	11	gigabytes
10	10^{10}	128	days	1	terabyte
12	10^{12}	35	years	111	terabytes
14	10^{14}	3500	years	11,111	terabytes

Depth-first Search



Iterative Deepening Search



Comparing Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes