Artificial Intelligence

Instructor: Kietikul Jearanaitanakij
Department of Computer Engineering
King Mongkut’s Institute of Technology Ladkrabang

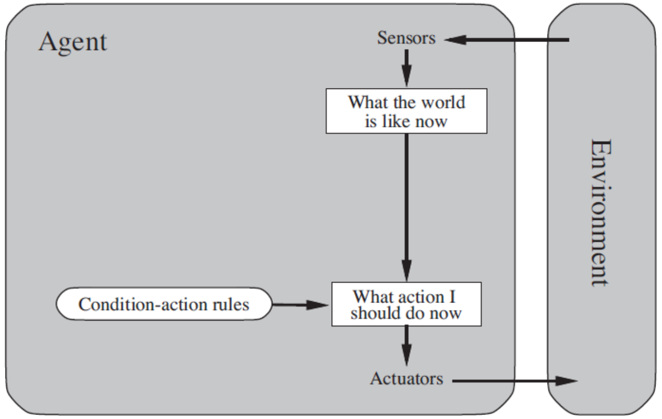
Lecture 2

Solving problems by searching

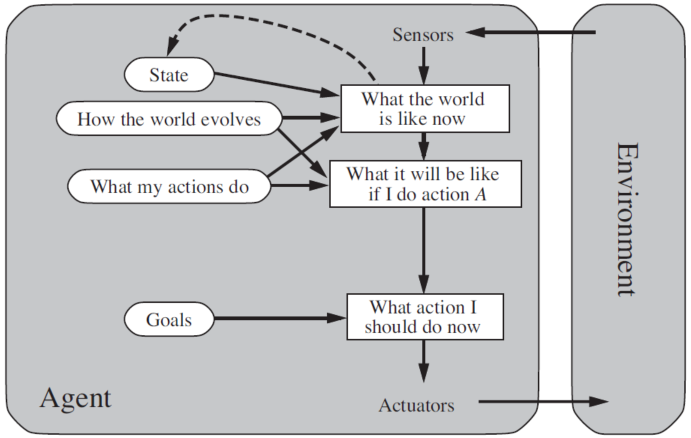
- Problem-solving agents
- Case studies
- Measuring problem-solving performance

Problem-solving agents

- Recall: Simple reflex agents
  - Actions based on only percepts
  - No knowledge
  - No goal



Simple reflex agent

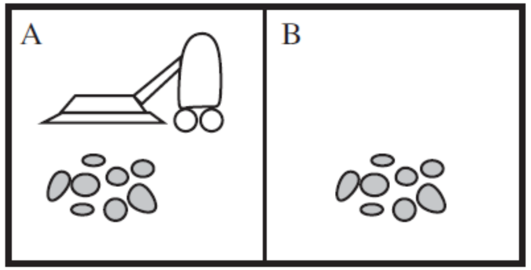


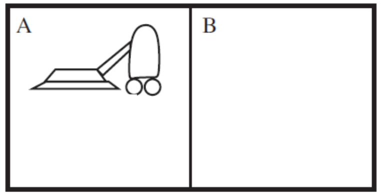
Goal-based agent

- Problem-solving agents.
  - Goal-based agent
  - Finding sequences of actions that lead to goal state.
  - **Goal Formulation**: define a goal state based on the current state. This is the first step in problem solving.
  - **Problem Formulation**: define actions and states to be considered for reaching the goal.

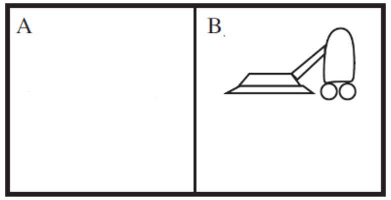
Case study 1: Vacuum World

- This particular world has just two locations: squares A and B.
- The vacuum agent perceives which square it is in and whether there is dirt in the square.
- It can choose to move left, move right, suck up the dirt, or do nothing.
- Goal: { Goal1, Goal2 }





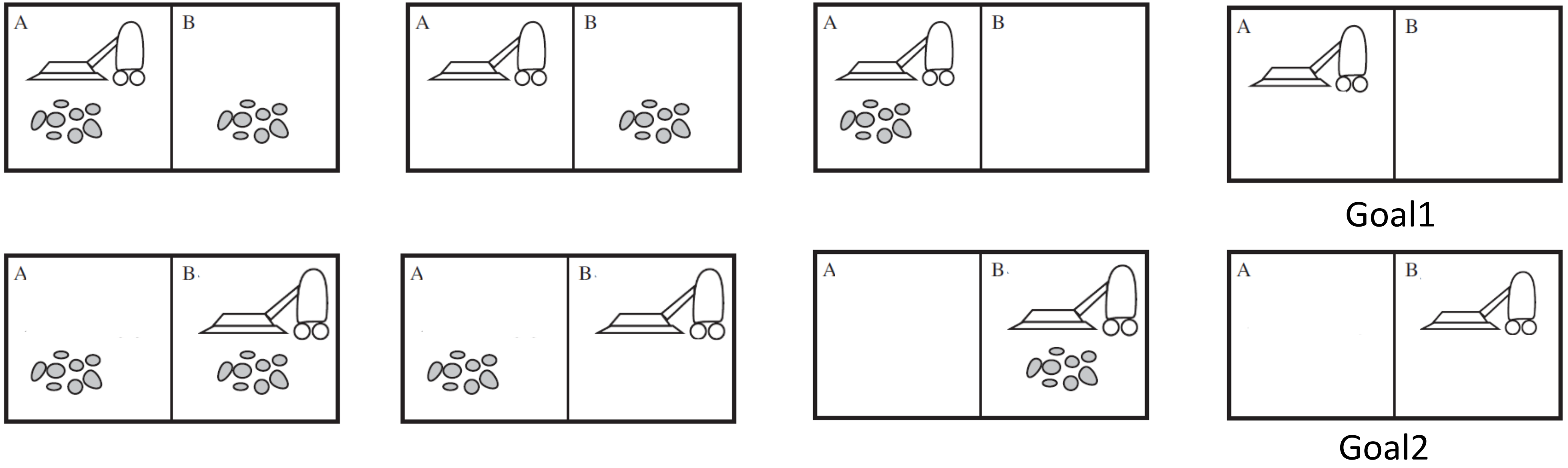
Goal1



Goal2

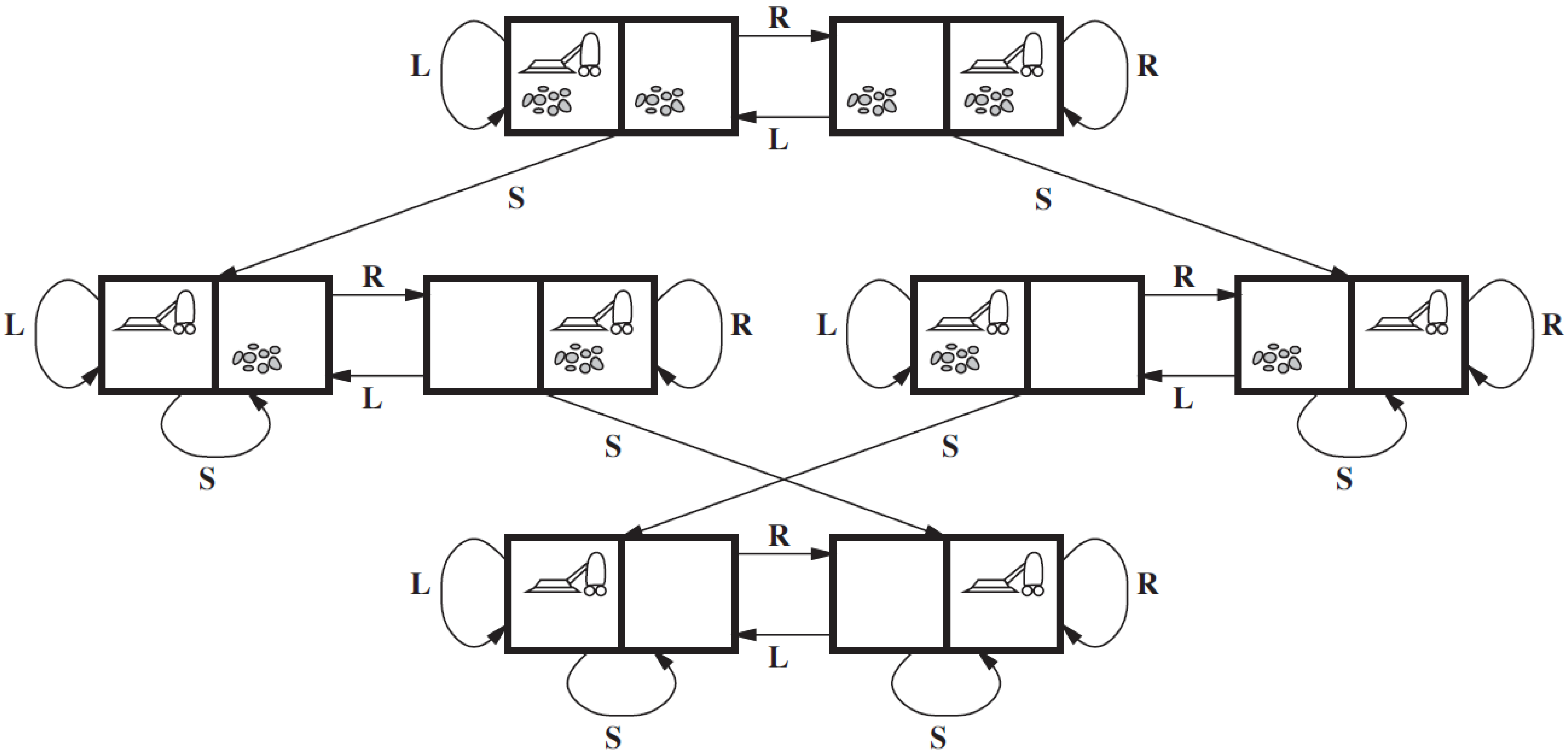
Case study 1: Vacuum World

- Formulating problem
  - Possible **actions** : {Left, Right, Suck, None}
  - **States**: 8 possible states

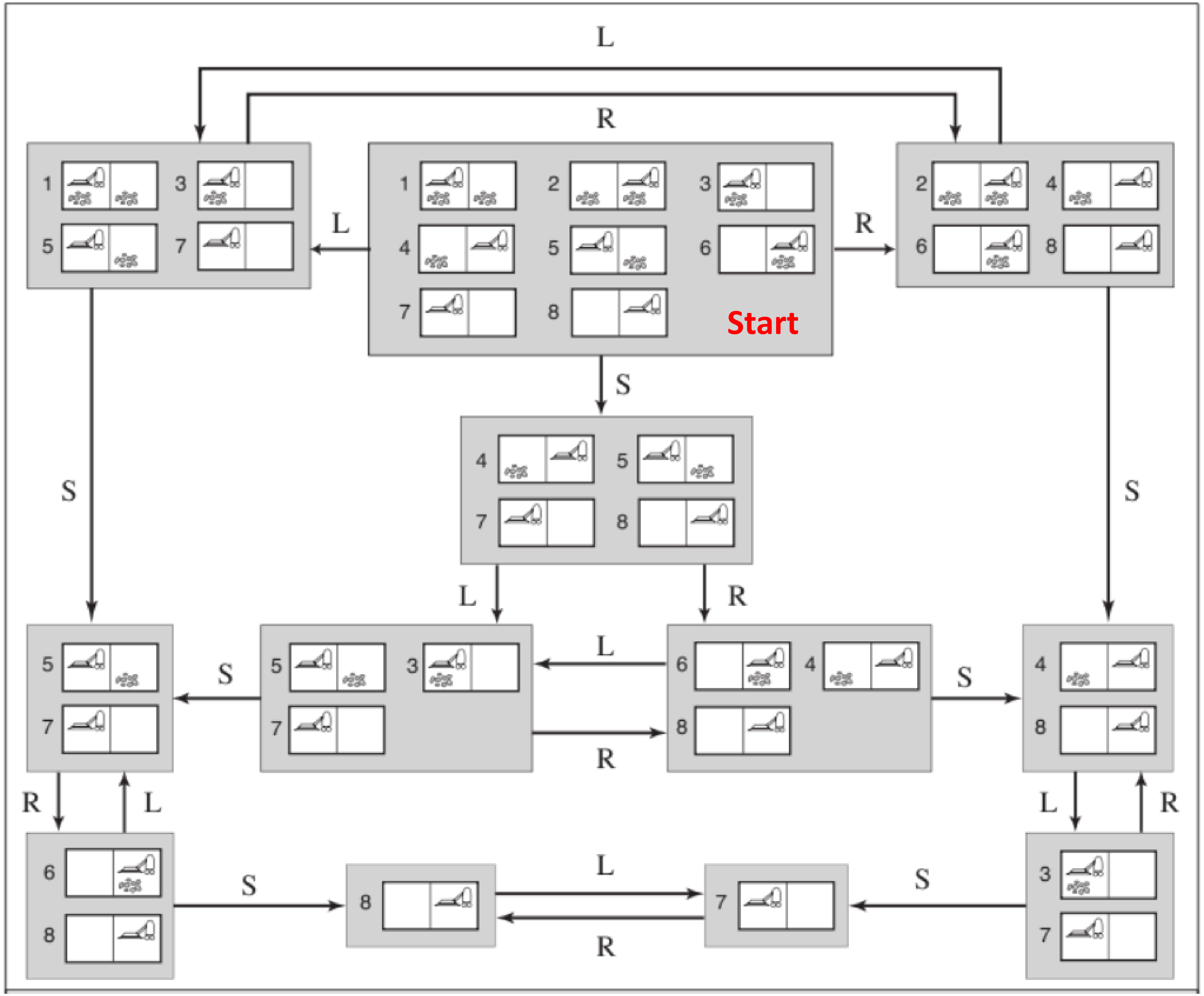


- Path cost : Each step costs 1, so the path cost is the number of steps in the path.

State space of Vacuum world with sensors

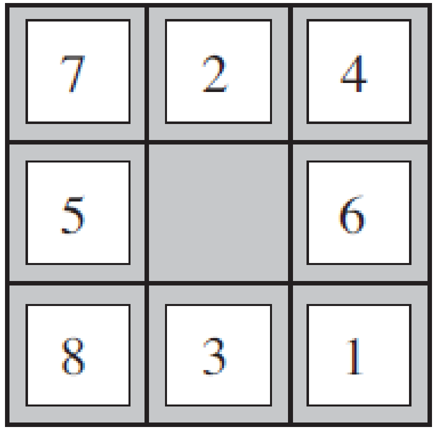


Sensor-less vacuum world

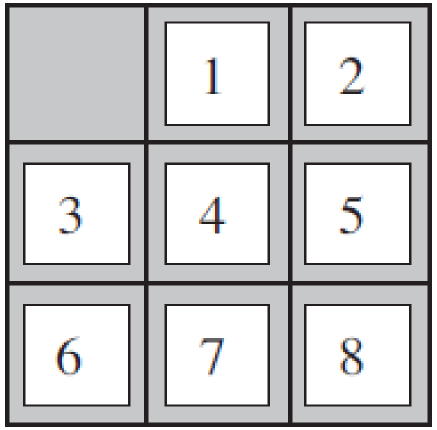


Case study 2: 8-puzzle

- **States**: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
  (Is it 9! ?)



Start State

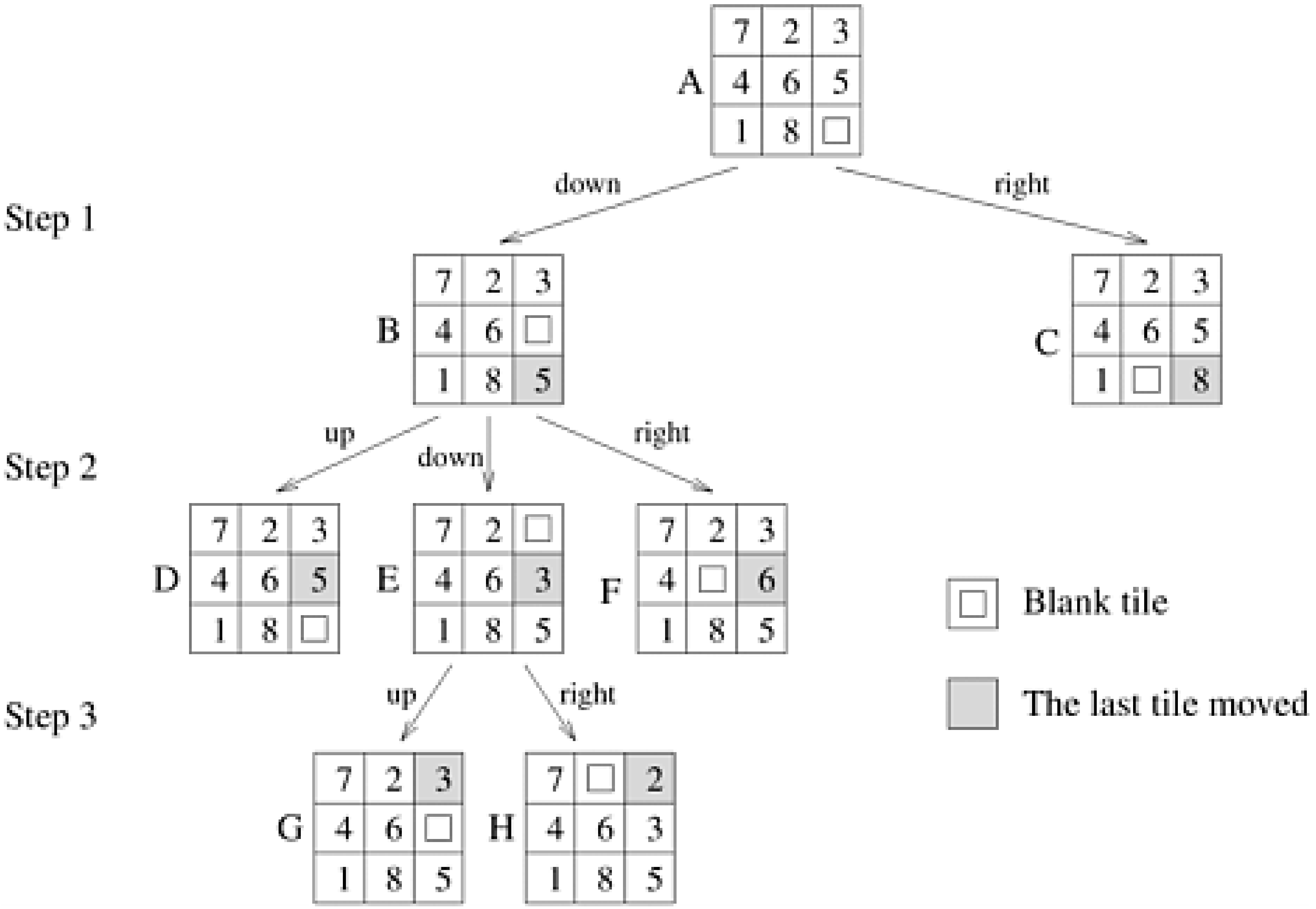


Goal State

- Initial state: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.

- **Actions**: Move the blank space Left, Right, Up, or Down.
- Path cost: Each step costs 1, so it is the number of steps in the path.

The partial search tree of 8-puzzle problem

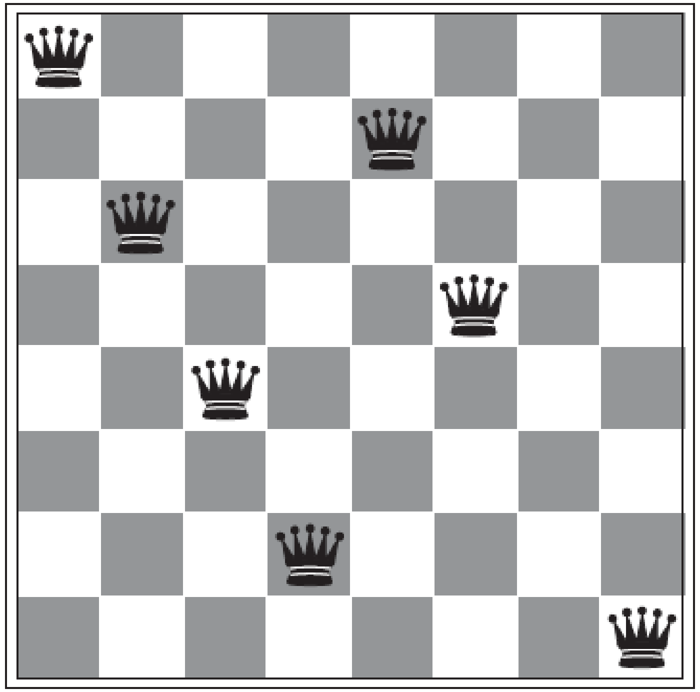


Case study 3: 8-queens problem

8-Queens in JavaScript
http://eightqueen.becher-sundstroem.de/

There are two main kinds of formulation.

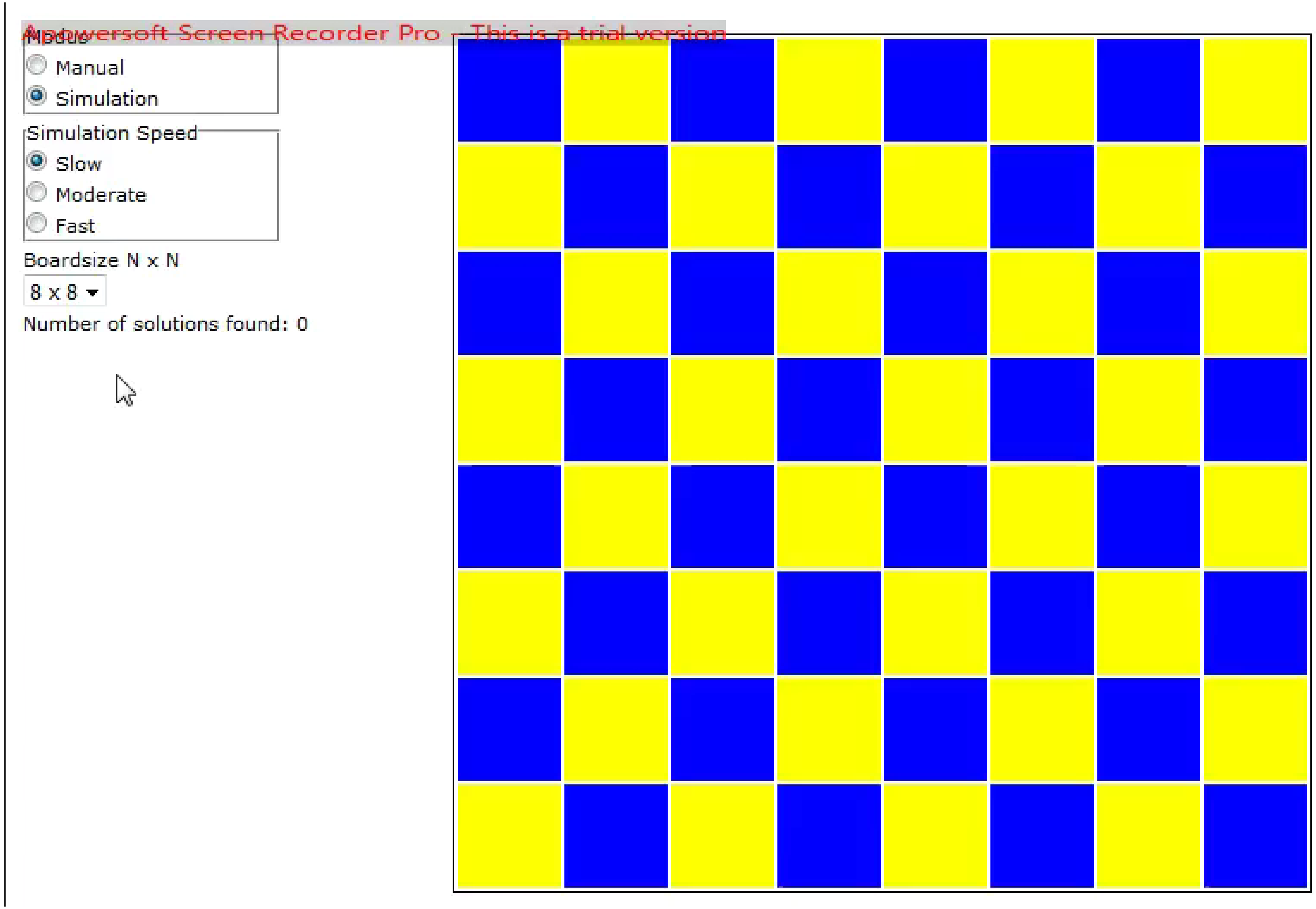
- An **incremental** formulation involves operators that augment the state description, starting with an empty state, each action adds a queen to the state.
  - **States**: Any arrangement of 0 to 8 queens on the board (one column per one queen).
  - **Initial state**: No queens on the board.
  - **Actions**: Add a queen to the leftmost empty column with none attacked.
  - **Goal test**: 8 queens are on the board, none attacked.



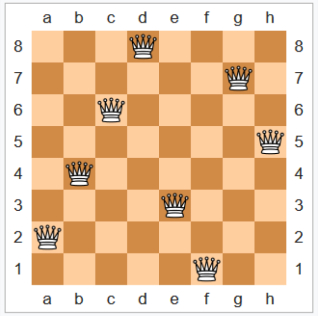
Incomplete 8-queens state

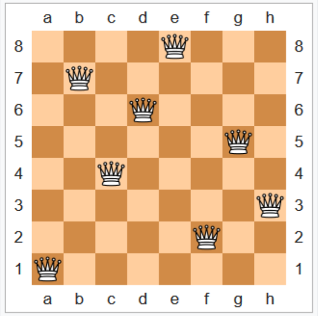
- A **complete-state** formulation starts with all 8 queens on the board and moves them around. (We will discuss about it in the next lecture)

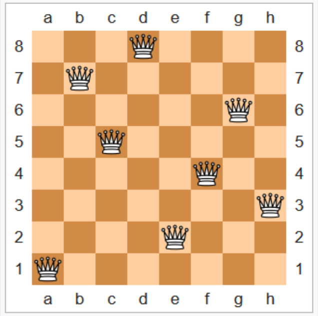
8-Queen : An incremental formulation

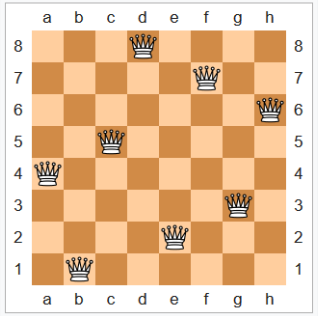


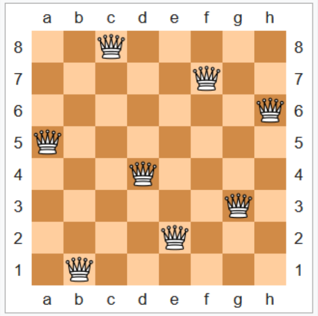
- The eight queens puzzle has 92 distinct solutions.
- If solutions that differ only by the symmetry operations of rotation and reflection of the board are counted as one, there are 12 fundamental solutions.

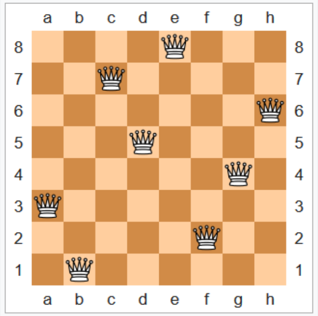

Solution 1

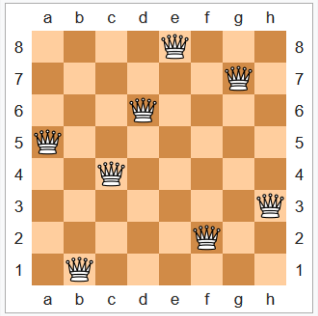

Solution 2

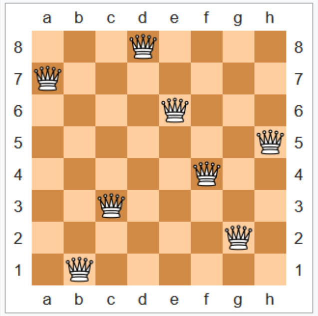

Solution 3

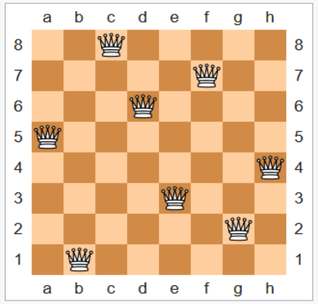

Solution 4

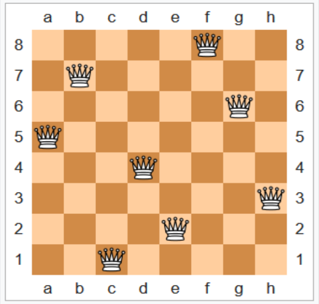

Solution 5

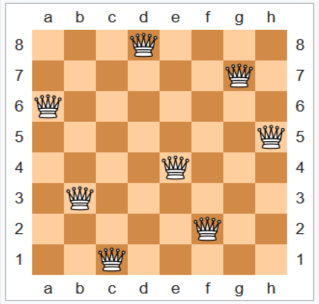

Solution 6

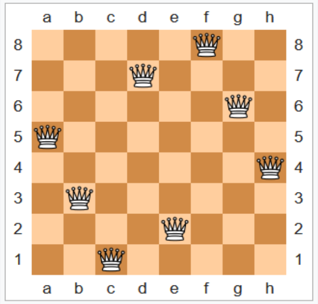

Solution 7


Solution 8


Solution 9


Solution 10


Solution 11


Solution 12

(Pictures taken from Wikipedia)

Case study 4: Donald Knuth (1964)

Knuth conjectured that, starting with the number **4**, a sequence of **factorial**, **square root**, and **floor** operations will reach any desired positive integer.

Example: Positive integer 5 can be derived from the following operations.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

- **States**: Positive numbers.
- Initial state: 4.
- **Actions**: Apply factorial, square root, or floor operation (factorial for integers only).
- Goal test: State is the desired positive integer.

Donald Ervin Knuth : An American computer scientist, mathematician. (Wikipedia)

Exercise: Missionaries & Cannibals Problem

Start <0,0,0>

Goal <3,3,1>

Boat is on right bank

#Cannibals on right bank

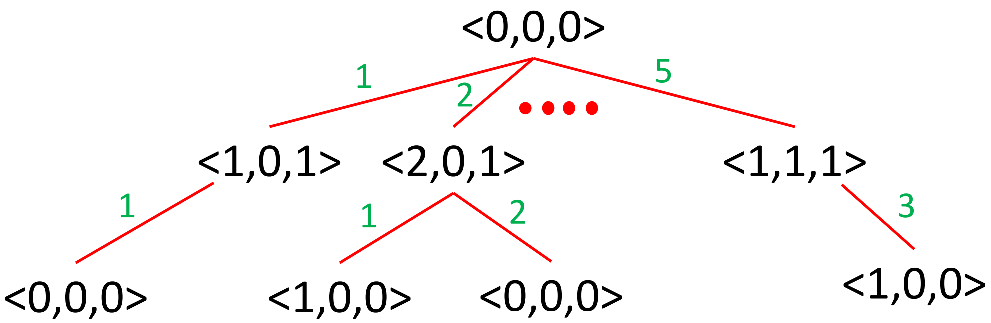
#Missionaries on right bank

- 3 missionaries and 3 cannibals must cross a river using a boat which can carry at most two people.
- If there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries).
- State representation : <3,3,1> means 3 cannibals and 3 missionaries are on the right bank of the river and boat is on the right bank. (This is a goal state)

- **State space** : { <0,0,0> , <1,0,0>, <0,1,0>, <2,2,1>, …, <3,3,1> }
- Note that some states are not reachable, for example, <0,0,1>, <3,3,0>, etc.
- **Actions**:
  1. Move one cannibal to the other side.
  2. Move two cannibals to the other side.
  3. Move one missionary to the other side.
  4. Move two missionaries to the other side.
  5. Move one cannibal and one missionary to the other side.

Start <0,0,0>

- **Partial search tree**:



Find the shortest sequence of actions that leads from the start state to the goal state.

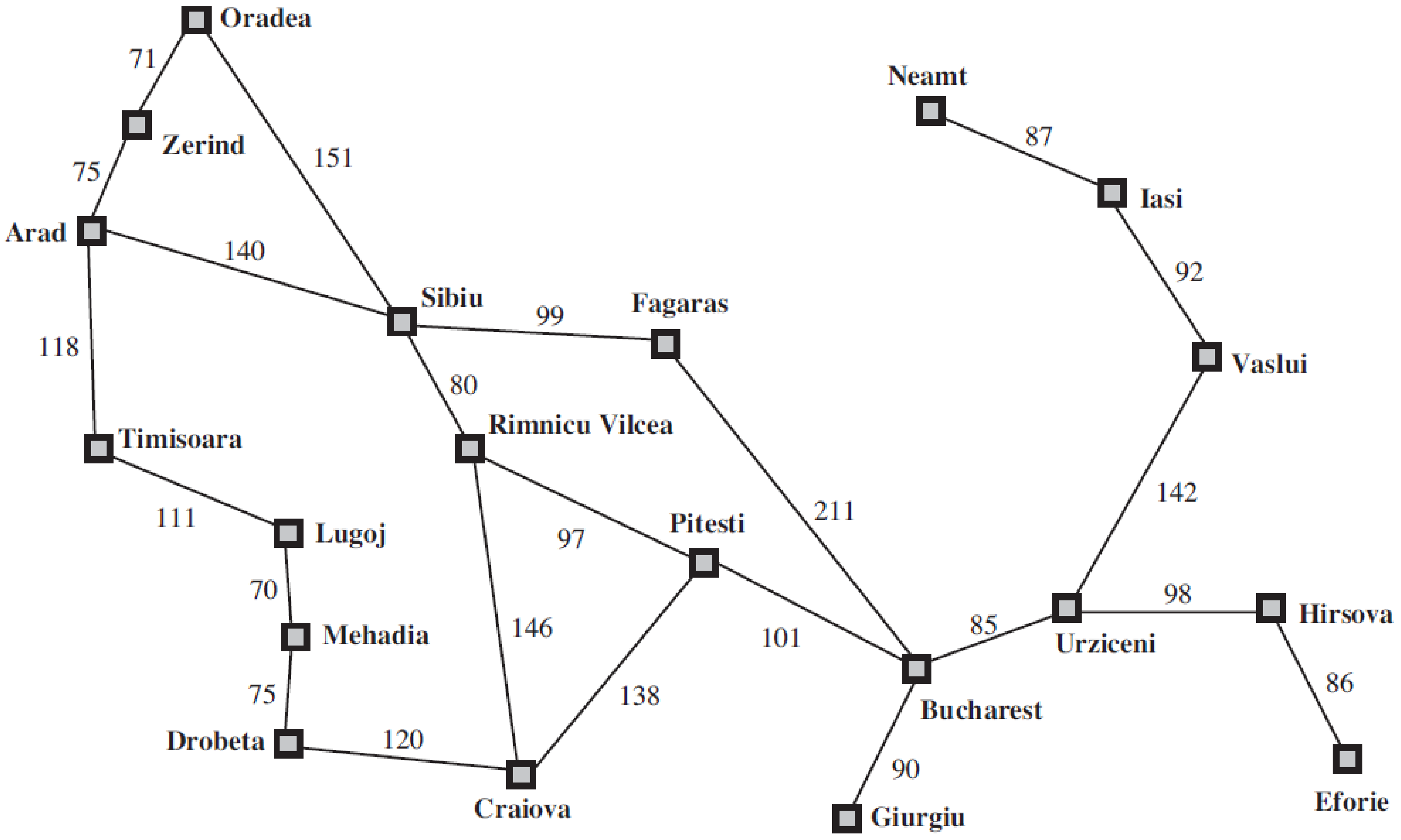
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   persistent: seq, an action sequence, initially empty
               state, some description of the current world state
               goal, a goal, initially null
               problem, a problem formulation

   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
```

Goal Formulation: define a goal state based on the current state.

Problem Formulation: define actions and states to be considered for reaching the goal.

Searching for Solutions



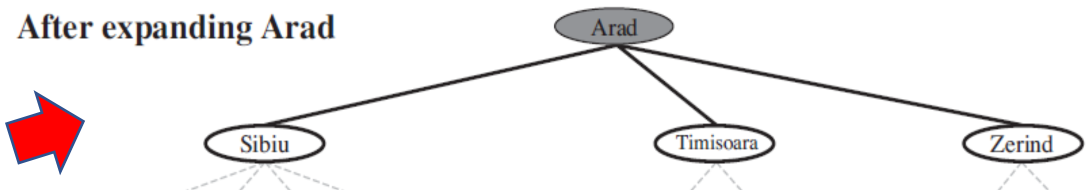
Searching for Solutions

Problem: Find the route starting from Arad to Bucharest

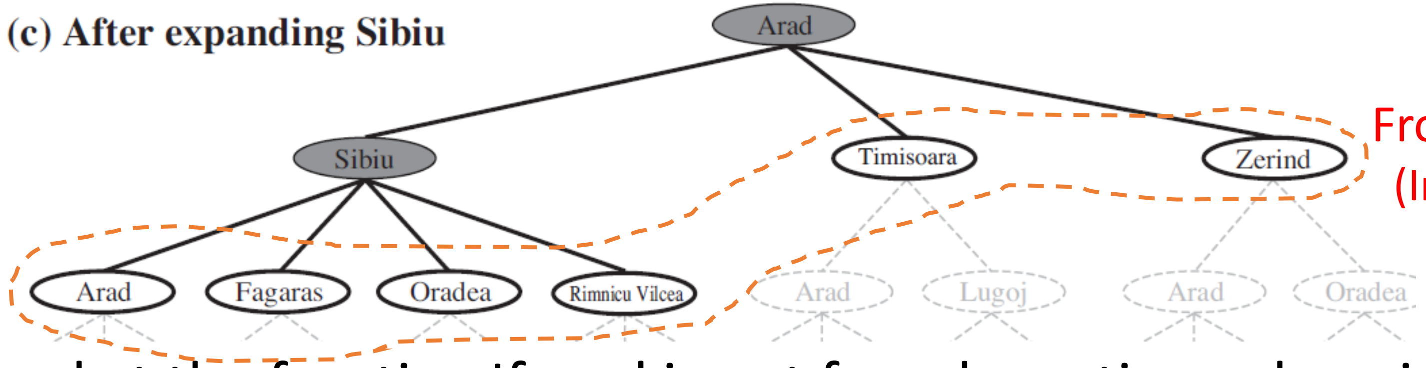
Solution:

1) Test if this is a goal state

2) Expand the current state Arad



The choice of which state to expand first depends on the search strategy.



Frontier
(Implemented by queue)

3) Check goal at the frontier. If goal is not found, continue choosing node to expand and goal checking until the goal is found or no more node to expand.

Searching for Solutions

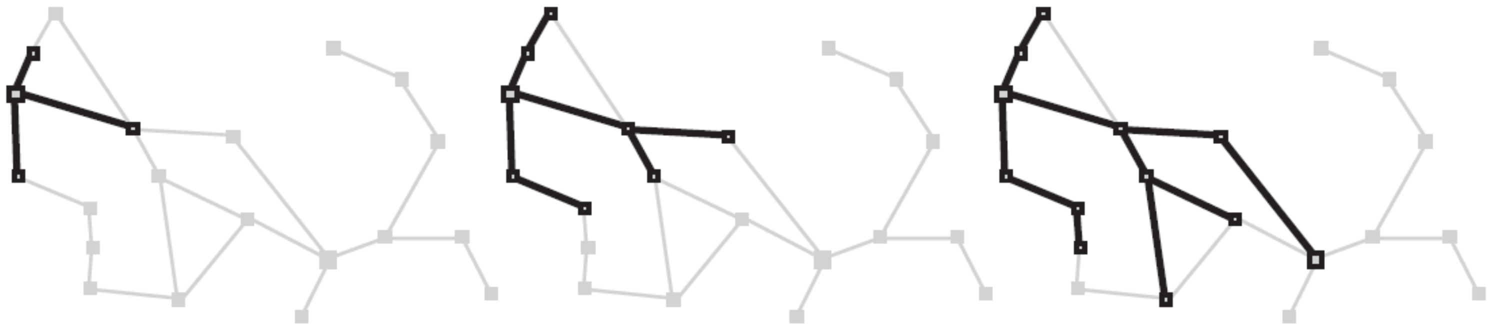
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

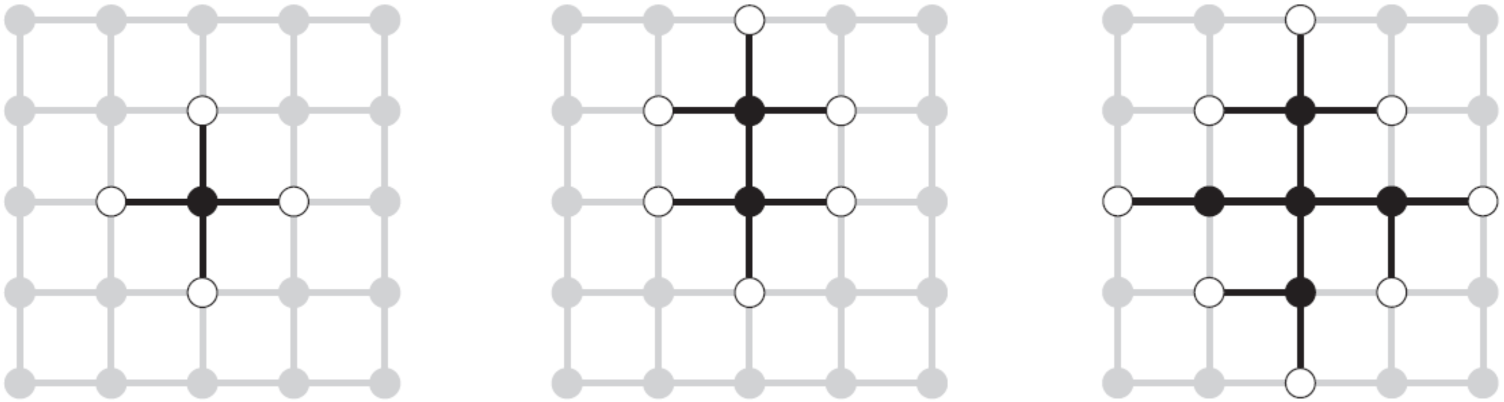
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the 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

Additions needed to handle repeated states

Searching for Solutions



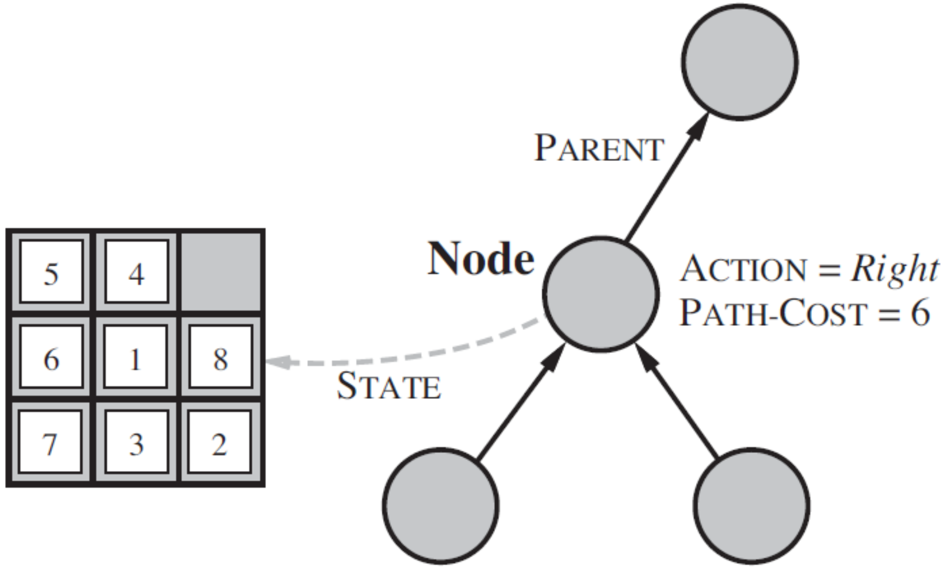
Graph search on Romania map



(a) (b) (c)

Graph search on rectangle grid. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes).

Infrastructure for search algorithms



For each node n of the tree, we have a structure that contains four components:

- **n.STATE** : the state in the state space to which the node corresponds;
- **n.PARENT** : the node in the search tree that generated this node;
- **n.ACTION** : the action that was applied to the parent to generate the node;
- **n.PATH-COST** : the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers.

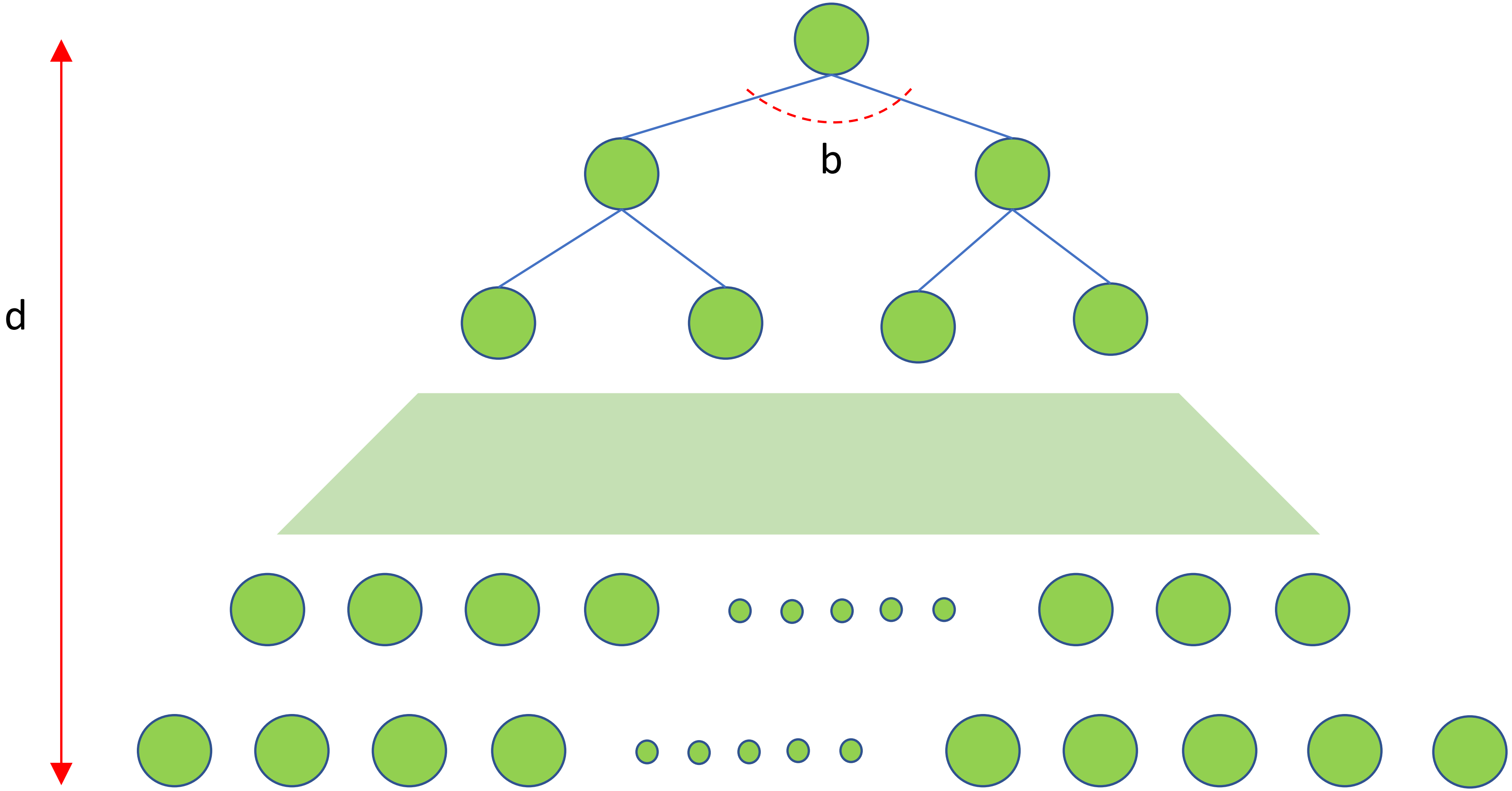
Measuring problem-solving performance

We can evaluate an algorithm’s performance in four ways:

- **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution?
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory is needed to perform the search?

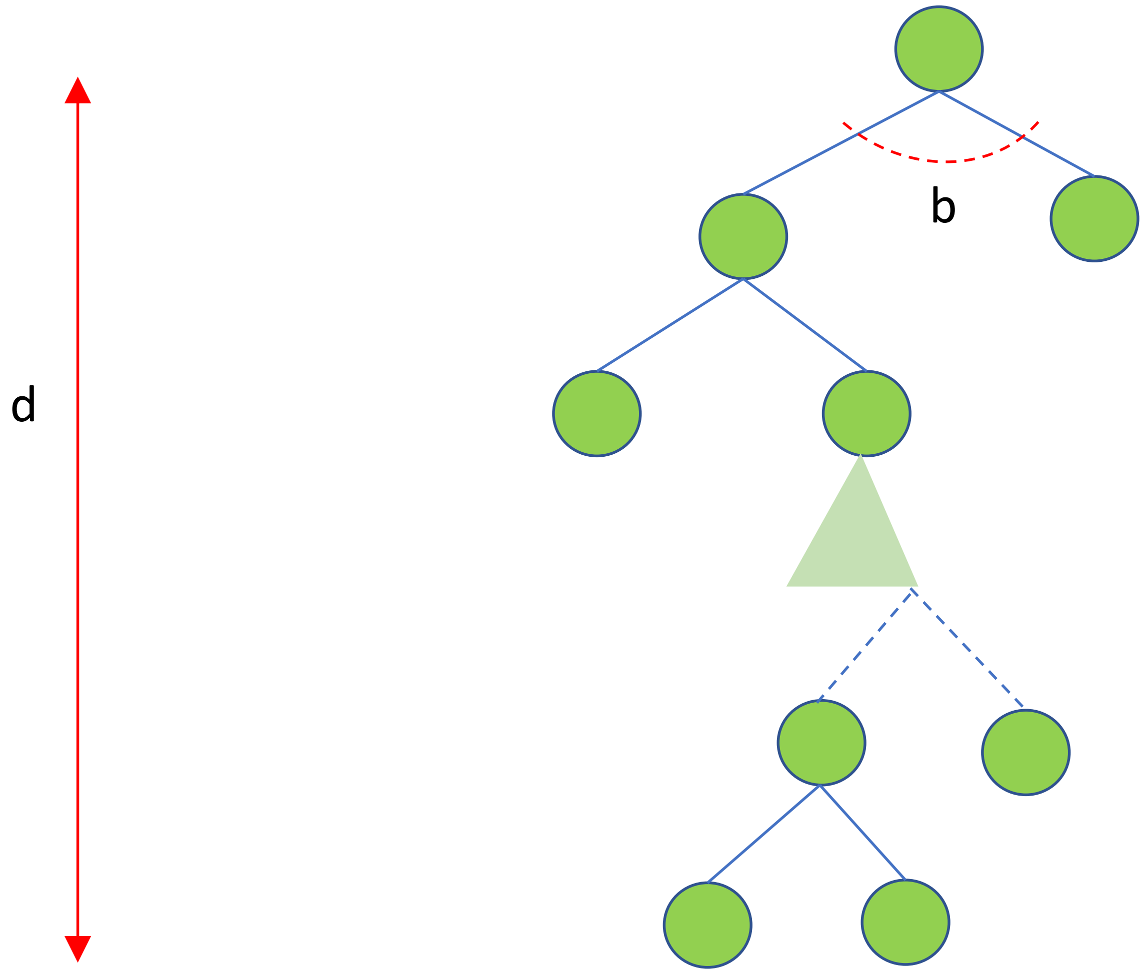
In data structure, space is usually represented by |Edges| + |Vertices|.
In AI, complexity is expressed in terms of three quantities:

- **b :** the **branching factor** or maximum number of successors of any node
- **d :** the **depth** of the shallowest goal node
- **m :** the **maximum length** of any path in the state space.

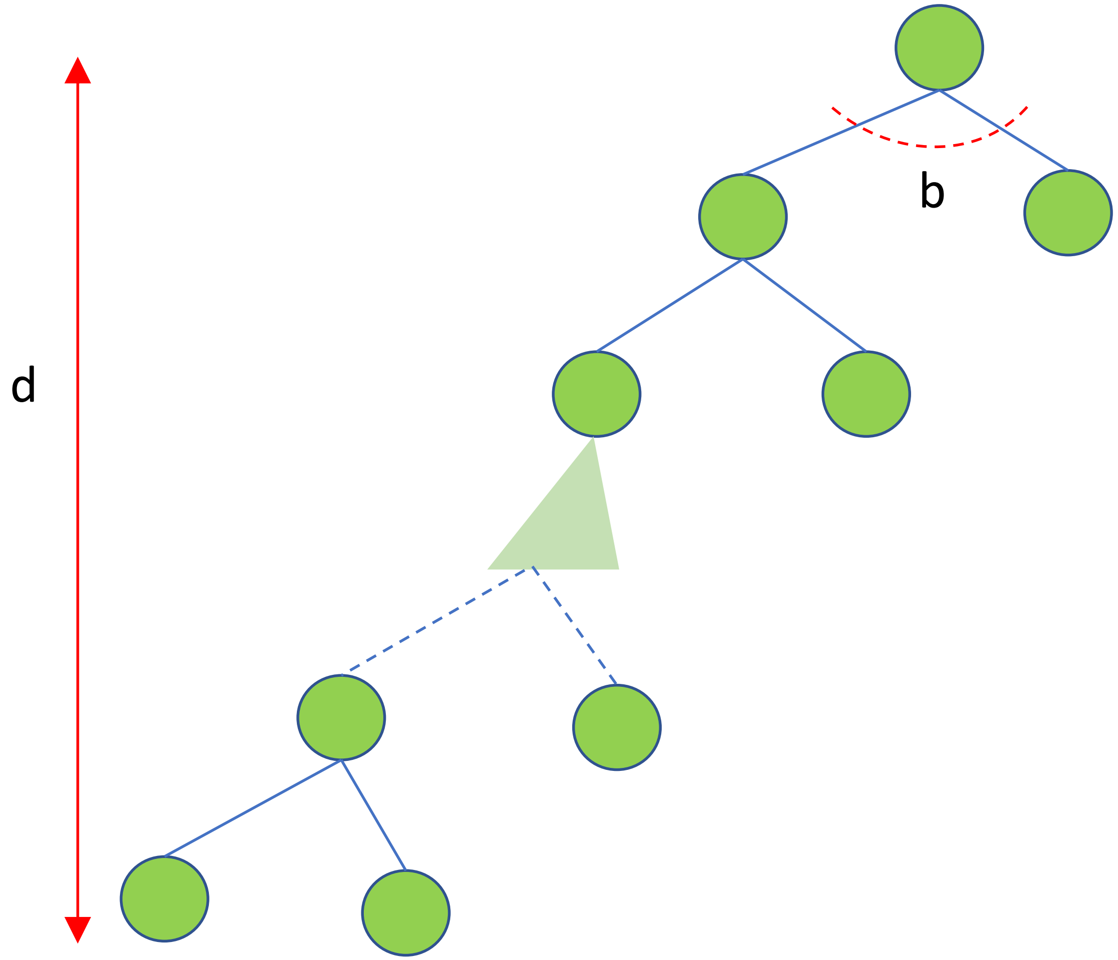


What is the time complexity of visiting all nodes in the tree ? (Worst-case scenario)

What is the space complexity of storing all nodes in the tree ? (Worst-case scenario)

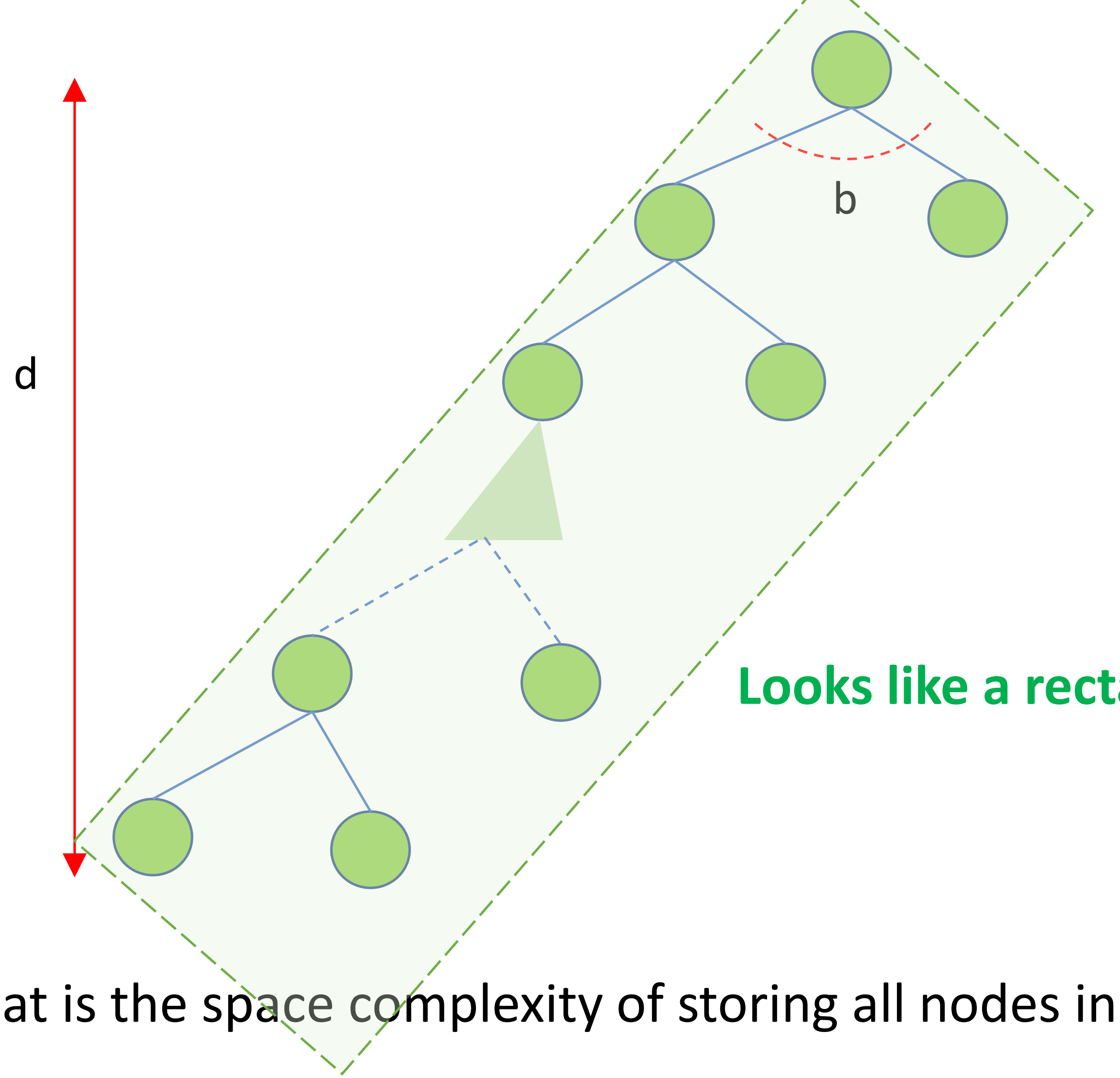


What is the space complexity of storing all nodes in the tree ? (Worst-case scenario)



Similar scenario

What is the space complexity of storing all nodes in the tree ? (Worst-case scenario)



Similar scenario

Looks like a rectangle, doesn't it?

What is the space complexity of storing all nodes in the tree ? (Worst-case scenario)