22년 삼성 AI 전문가과정
6월 7일 화요일 3교시
장병탁

인공지능

3차시 : Problem-Solving Agents

서울대학교 컴퓨터공학부
담당 교수: 장병탁

Seoul National University
Byoung-Tak Zhang

Lecture Overview

인공지능

3차시 : Problem-Solving Agents

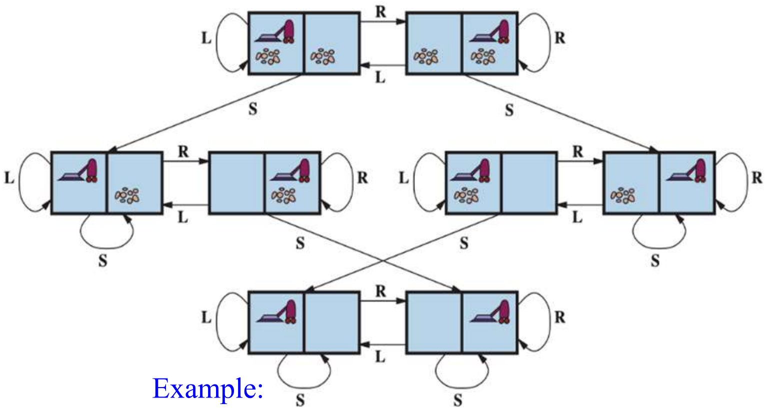
서울대학교 컴퓨터공학부
담당 교수: 장병탁

Seoul National University
Byoung-Tak Zhang

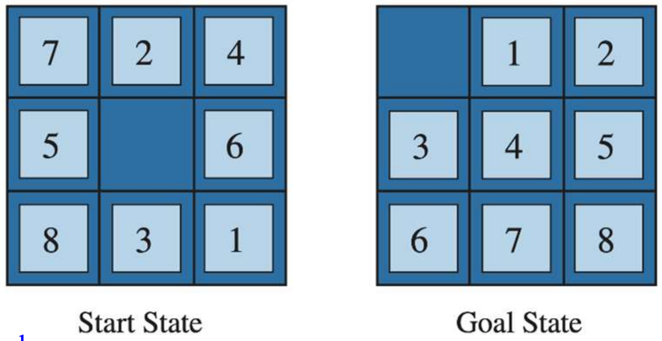
exhaustive search,

Lecture 3. Problem-Solving Agents

- ➢ Problem-Solving Agents
  - What are the Problem-Solving Agents?
  - Examples of Problems, Goals, Elements
- ➢ Searching for Solutions == solving problem
  - Tree search algorithms
- ➢ Uninformed Search Strategies
  - Breadth-first search, Uniform-cost search, Depth-first search, Depth-limited search, Iterative-deepening search
- ➢ Informed Search Strategies
  - Greedy Search, A* Search
- ➢ Heuristic Functions
  - The 8-puzzle, Relaxed problems



Example:
State-space graph for the two-cell vacuum world



Example:
A typical instance of the 8-puzzle

Problem-Solving Agents

Reflex agents

Goal-based agents

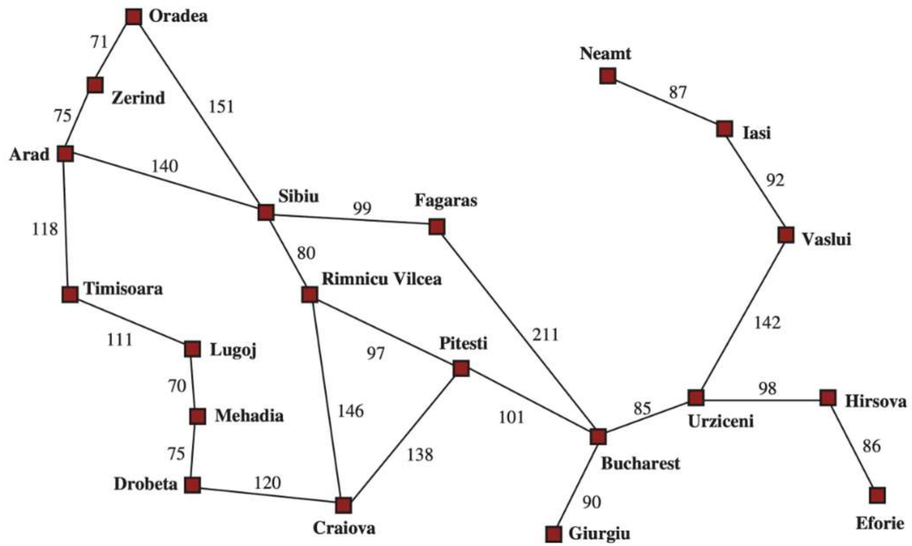
- Consider future actions and the desirability of their outcomes
- Problem-solving agents (Chapter 3)
  - Atomic representation (of states)
- Planning agents (Chapter 11)
  - Factored or structured representation

Problem-solving agents (this lecture)

- Problems, solutions, search algorithms
- Uninformed search algorithms
- Informed search algorithms

Problem Solving as Search in State Space: Touring (Driving)

Start State



Goal State

Goal: be in Bucharest

Problem formulation:

- ➢ states = various cities
- ➢ actions = drive between cities

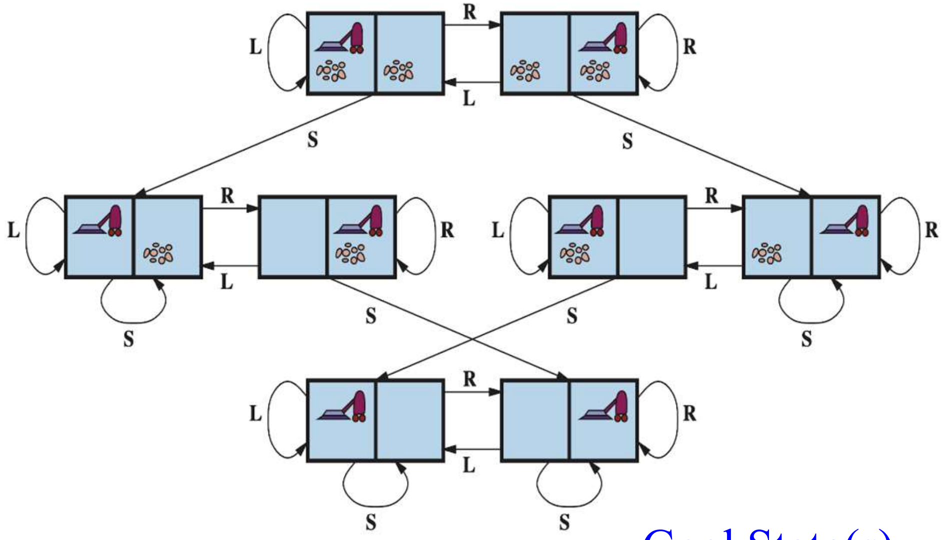
Goal-based Agent

Touring (Driving)

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

Problem-Solving Agent: Vacuum Cleaner Robot

Start State



Goal State(s)

Vacuuming

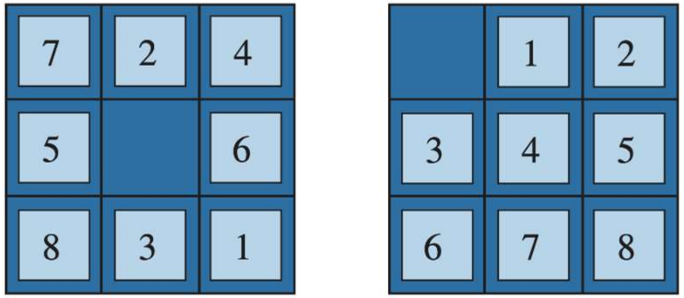
Goal: all rooms clean

Problem formulation:

- ➢ States: Boolean dirt and robot location
- ➢ Actions: Left, Right, Suck

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

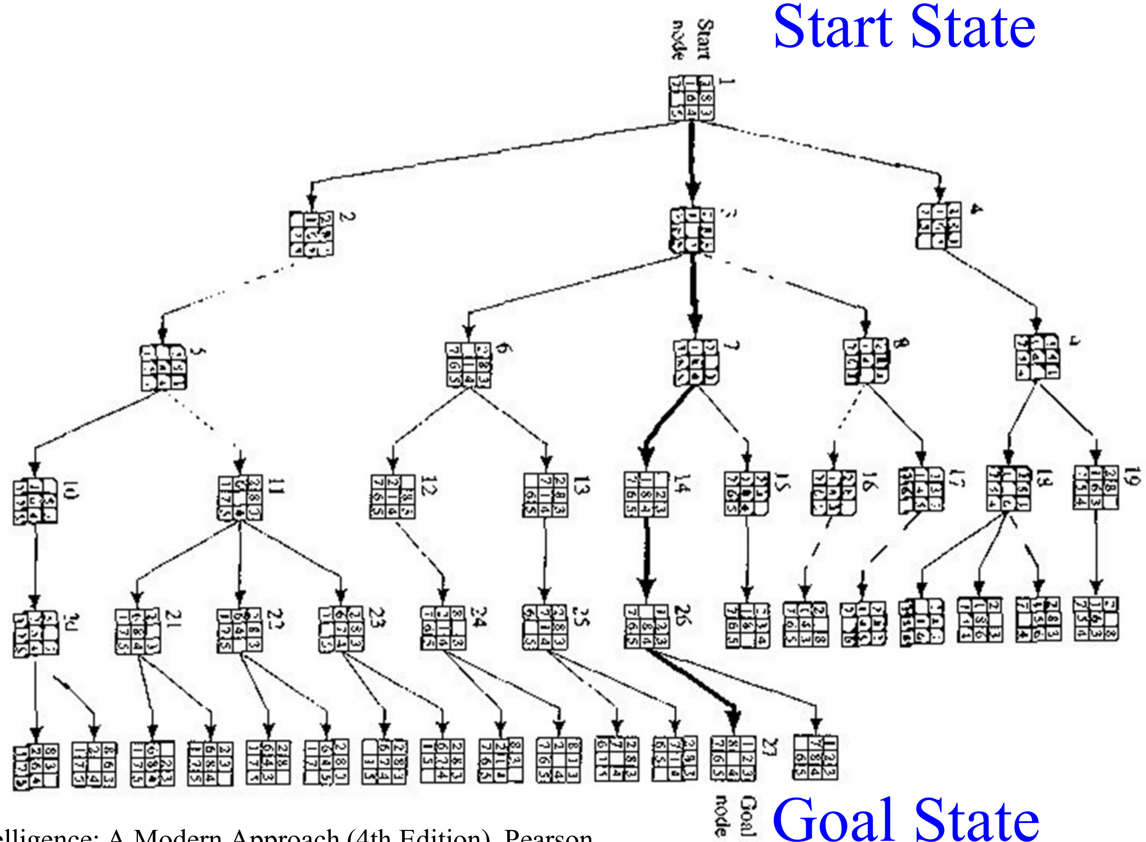
Problem Solving as Search in State Space



Start State　　Goal State

Puzzle Solving

Start State

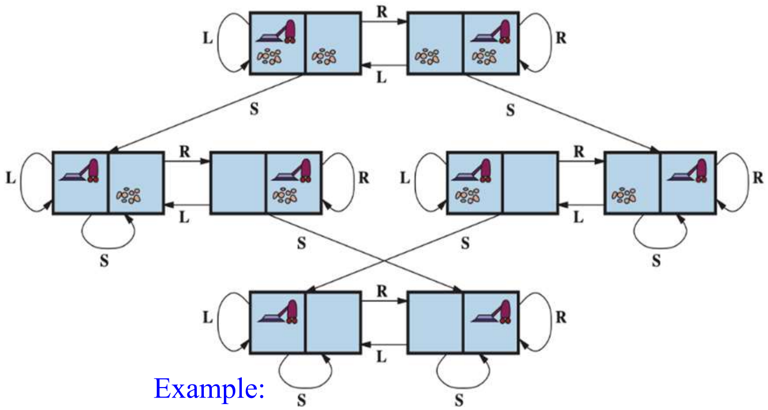


Goal State

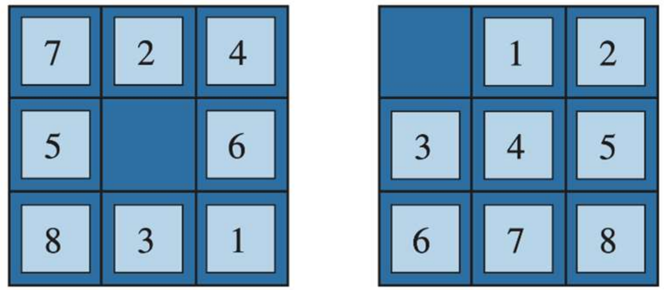
<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

Lecture 3. Problem-Solving Agents

- ➢ Problem-Solving Agents
  - ■ What are the Problem-Solving Agents?
  - ■ Examples of Problems, Goals, Elements
- ➢ Searching for Solutions
  - ■ Tree search algorithms
- ➢ Uninformed Search Strategies
  - ■ Breadth-first search, Uniform-cost search, Depth-first search, Depth-limited search, Iterative-deepening search
- ➢ Informed Search Strategies
  - ■ Greedy Search, A* Search
- ➢ Heuristic Functions
  - ■ The 8-puzzle, Relaxed problems



Example:
State-space graph for the two-cell vacuum world



Start State　　Goal State

Example:
A typical instance of the 8-puzzle

Outline (Lecture 3)

Stuart Russell & Peter Norvig (2021), Artificial Intelligence: A Modern Approach (4th Edition)

3.1. Problem-Solving Agents

3.1 Problem-Solving Agents (1/4)

Here we consider the environment:

- ➢ **Observable**: The agent always knows the current state
- ➢ **Known**: The agent knows which states are reached by each action
- ➢ **Deterministic**: Each action has exactly one outcome
- ➢ In this environment, the solution to any problem is a fixed sequence of actions. The agent ignores its percepts when choosing an action because it knows in advance what they will be. The agent is an open-loop system ("eyes closed").

Problem solving as search in state space

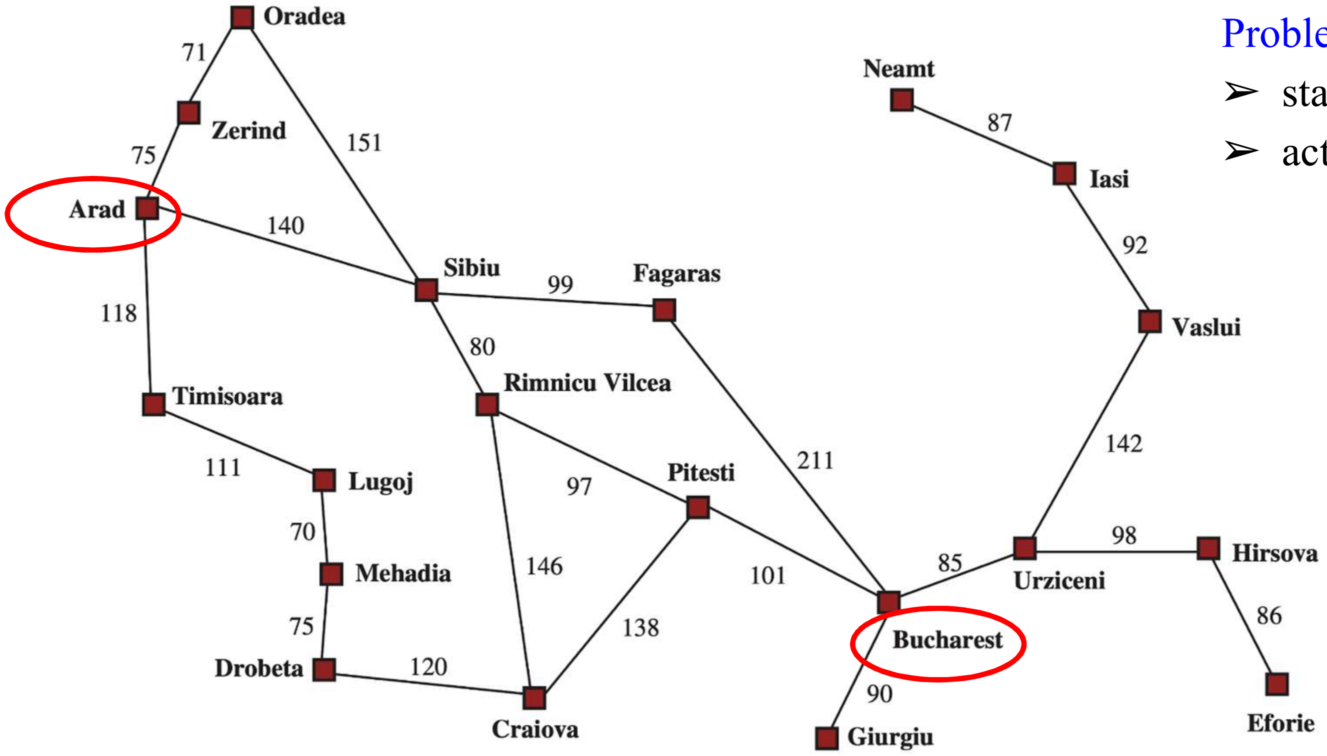
- ➢ **Goal formulation**: A goal is a set of world states to be reached.
- ➢ **Problem formulation**: The process of deciding what actions and states should be considered
- ➢ Search is the process of looking for a sequence of actions that reaches the goal in the state space.

Well-defined problems and solutions

- ➢ **Problem = <initial state, actions, transition model, goal test, path cost>**
- ➢ A solution to a problem is an action sequence that leads from the initial state to a goal state.

3.1 Problem-Solving Agents (2/4)

Example: Romania



Goal: be in Bucharest

Problem formulation:

- ➢ states = various cities
- ➢ actions = drive between cities

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.1 Problem-Solving Agents (3/4)

Simple form of problem-solving agents

```
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
```

Percept
State
Action
Goal

3.1 Problem-Solving Agents (4/4)

Problem types

➢ Deterministic, Fully observable	⇒	**Single-state**
➢ Non-observable (sensorless)	⇒	**Conformant** problem
➢ Nondeterministic and/or Partially observable	⇒	**Contingency** problem
➢ Unknown state space (online)	⇒	**Exploration** problem

This lecture

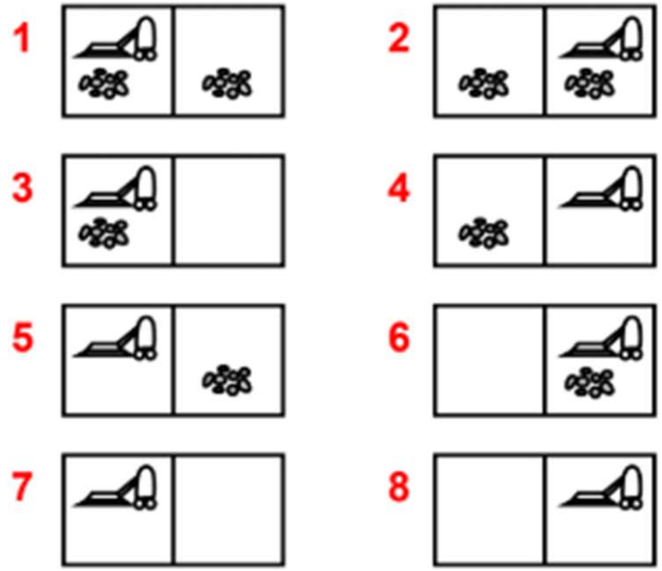
Later

3.2. Example Problems

3.2 Example Problems (1/4)

Example: vacuum world

- ➢**States:** Boolean dirt and robot location
- ➢**Actions:** Left, Right, Suck
- ➢**Transition model:** (next page)
- ➢**Goal test:** no dirt
- ➢**Path cost:** 1 per action



3.2 Example Problems (2/4)

Vacuum world

- ➢ Single-state

  start in 5

  - ■ [R, S]

- ➢ Conformant

  start in {1,2,3,4,5,6,7,8}

  - ■ [R, S, L, S]

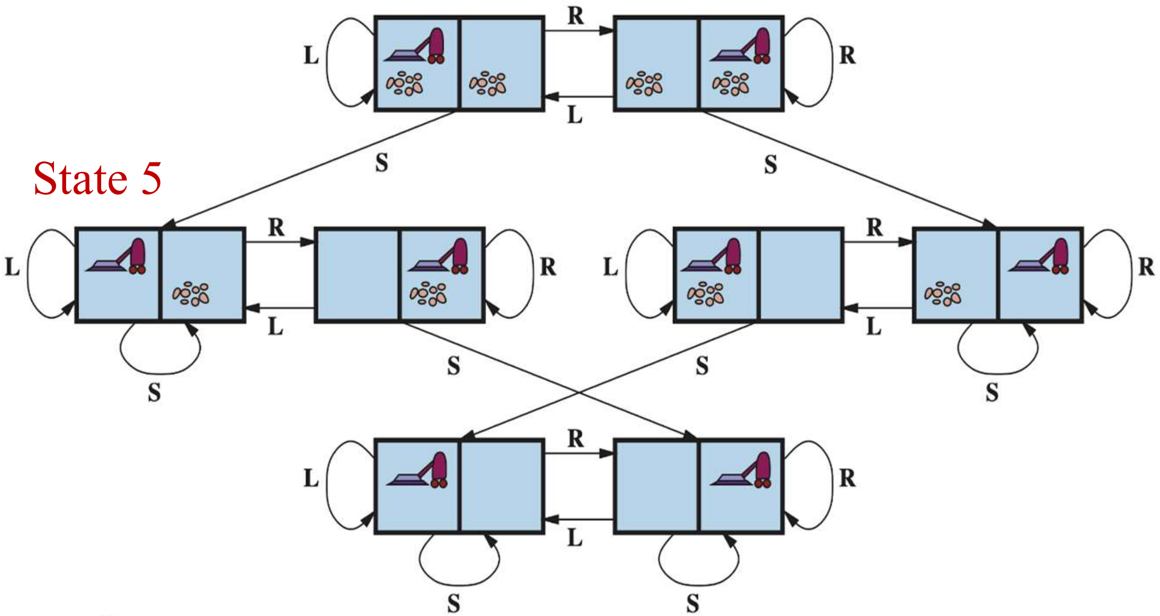
- ➢ Contingency

  start in 5

  Murphy’s Law: [S] can make clean square dırty

  - ■ [R, if dirt then S]

State 5

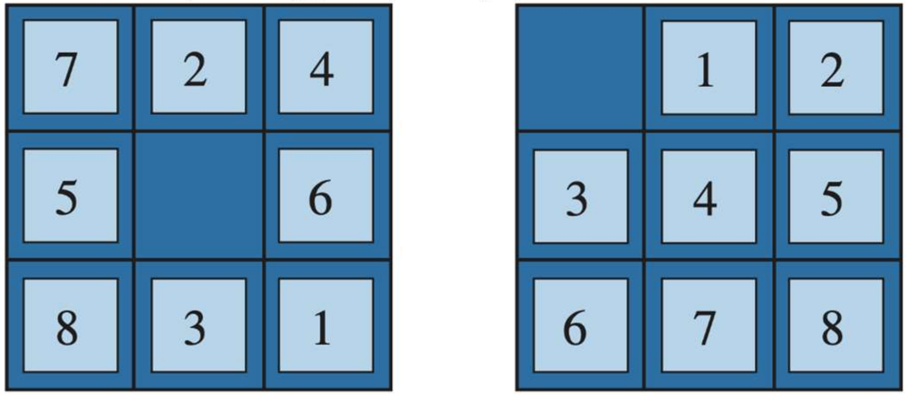


<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.2 Example Problems (3/4)

Example: The 8-puzzle (NP-hard)

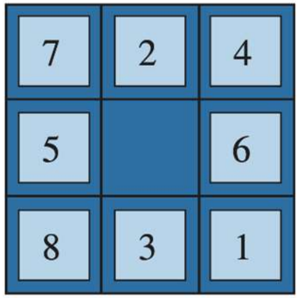
- ➢**States:** integer location of tiles
- ➢**Actions:** move blank tile Left, Right, Up, Down
- ➢**Transition model:**
- ➢**Goal test:** goal state
- ➢**Path cost:** 1 per action



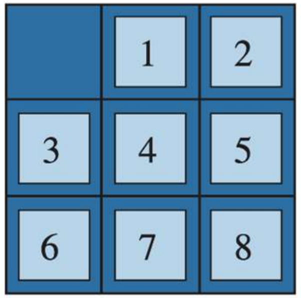
Start State　　　Goal State

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.1 Problem-Solving Agents (4/4)



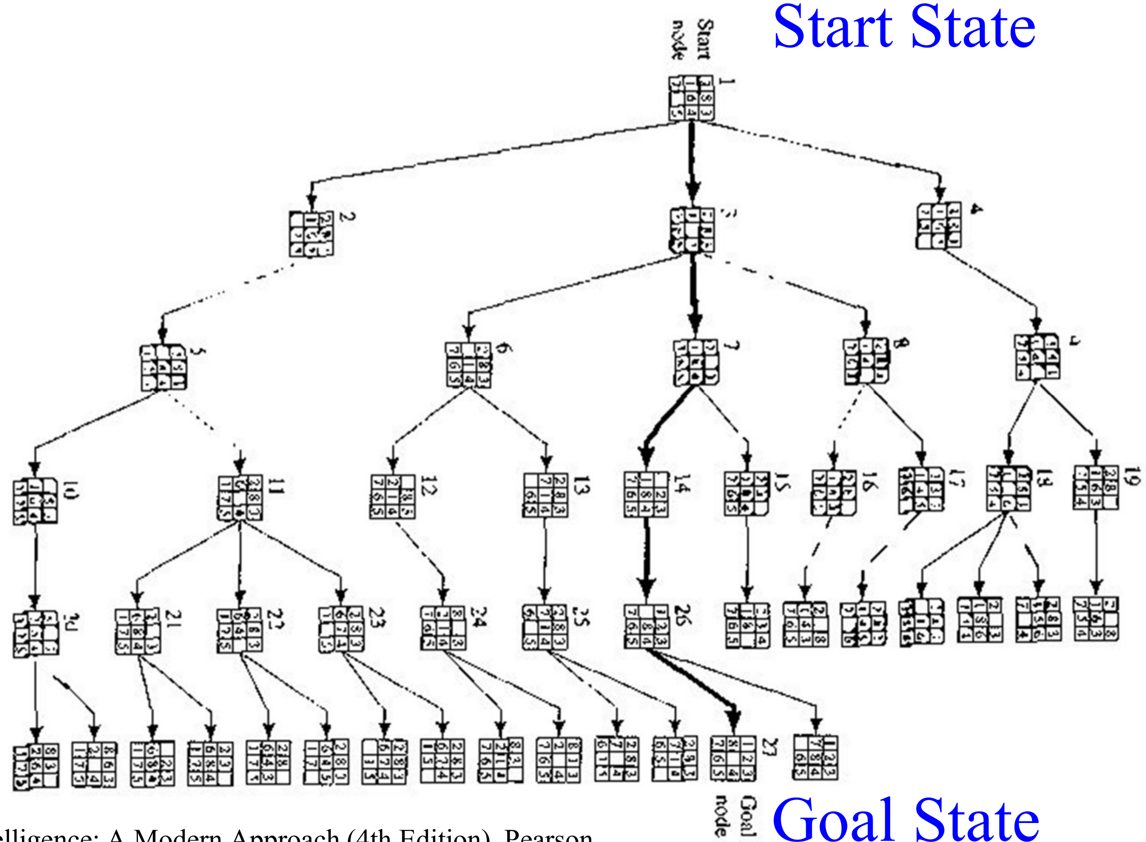
Start State



Goal State

Puzzle Solving

Start State



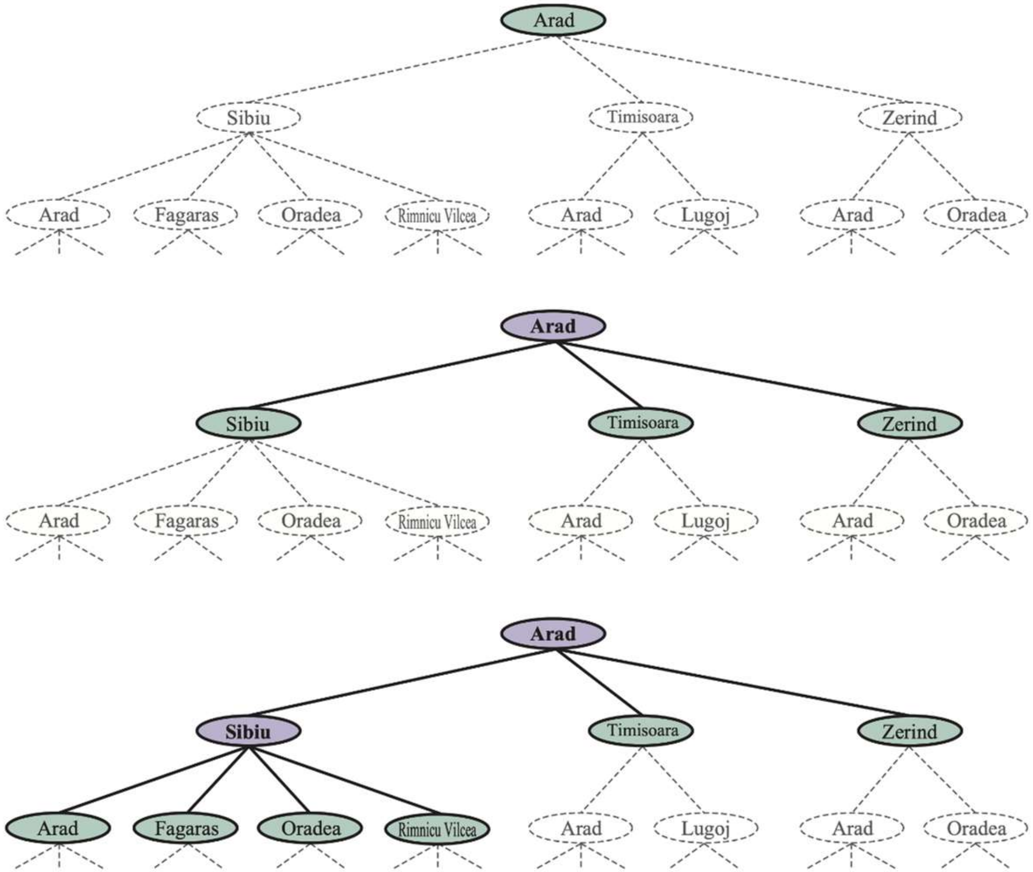
Goal State

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.3. Searching for Solutions

3.3 Searching for Solutions (1/4)

Partial search trees for finding a route from Arad to Bucharest



<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.3 Searching for Solutions (2/4)

Tree search algorithms

- Basic Idea: offline, simulated exploration of state space by generating successors of already-explored states

frontier

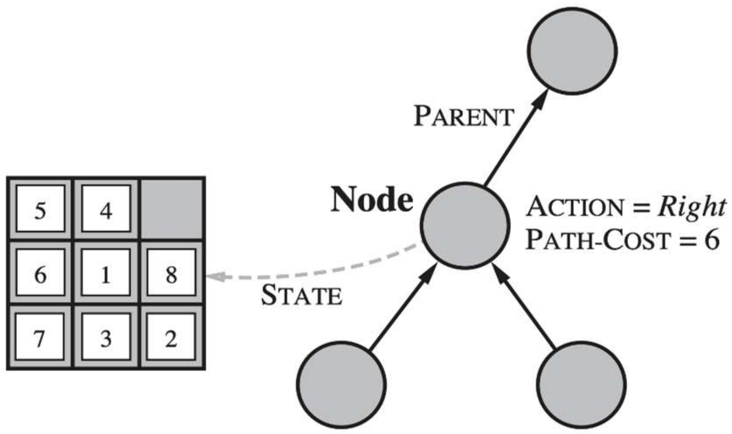
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return**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

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.3 Searching for Solutions (3/4)

Implementation: states vs nodes

- ➢ State is physical configuration
- ➢ Node is a data structure building block of a search tree
  - Parent, children, depth, path cost $g(x)$



<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.3 Searching for Solutions (4/4)

Implementation: EXPAND

- ➢ The Expand function creates new nodes, filling in the various fields
- ➢ Use successor of the problem to create corresponding states

```
function EXPAND(node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

3.4. Uninformed Search Strategies

3.4 Uninformed Search Strategies (1/11)

Evaluating for search strategies

- ➢Strategy is defined by picking the order of node expansion
- ➢Strategies are evaluated along the following metrics:
  - Completeness: Guarantee to find a solution if one exists
  - Optimality: Guarantee to find a minimum-cost path
  - Time complexity: How many nodes to expand to find a solution
  - Space complexity: How many nodes to keep to perform search

3.4 Uninformed Search Strategies (2/11)

2) Uninformed search strategies

➢Uninformed strategies use only the information available in the problem definition

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative-deepening search

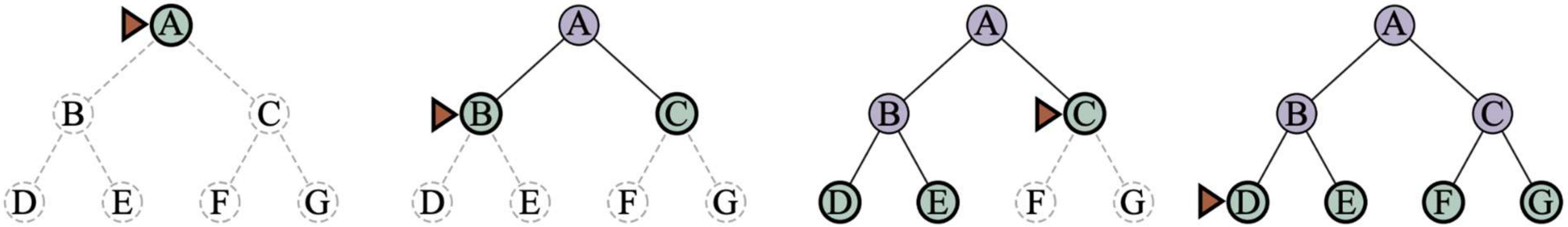
3.4 Uninformed Search Strategies (3/11)

I. Breadth-first search

- Expand shallowest unexpanded node
- Implement: fringe is a FIFO queue, i.e., new successors go at end

```
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)
```

I. Breadth-first search



<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

- Complete? Yes
- Time? $1 + b + b^2 + b^3 + \cdots + b^d + b(b^d - 1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes, Still, space is the big problem!

3.4 Uninformed Search Strategies (5/11)

II. Uniform-cost search

- ➢Expand least-cost unexpanded node
- ➢Implementation: queue ordered by path cost $g(n)$, lowest first
- ➢Complete? Yes, if step cost ≧ ε
- ➢Time? # of nodes with $g \leq$ cost of optimal solution $O(b^{\lceil C^*/\epsilon \rceil})$
- ➢Space? # of nodes with $g \leq$ cost of optimal solution $O(b^{\lceil C^*/\epsilon \rceil})$
- ➢Optimal? Yes, node expanded in order of $g(n)$

C^* is cost of the optimal solution

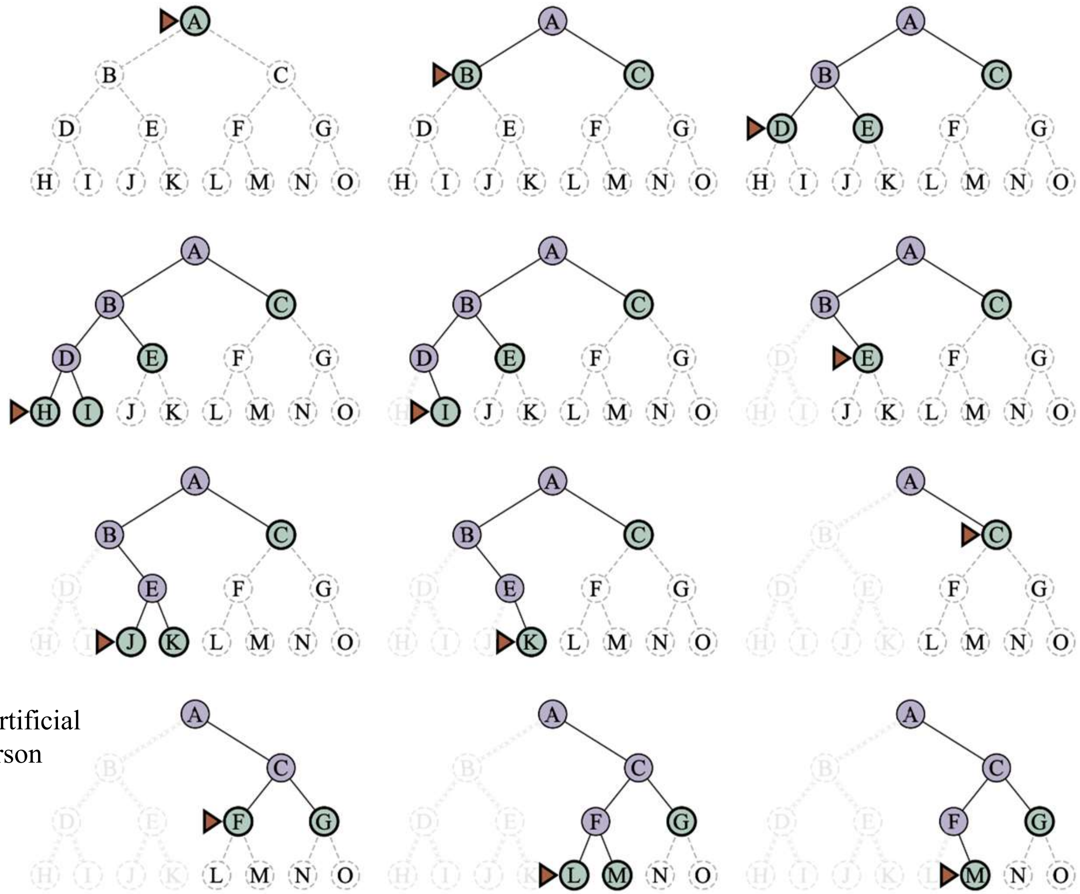
3.4 Uninformed Search Strategies (6/11)

III. Depth-first search (DFS)

- ➢Expand deepest node (depth)
- ➢Implementation: LIFO queue, i.e., put successors at front
- ➢Complete? No, fails in infinite-depth space with loops
  - ■ Complete in finite space
- ➢Time? $O(b^m)$,bad if *m* is bigger than *d*
- ➢Space? $O(bm)$
- ➢Optimal? No

3.4 Uninformed Search Strategies (7/11)

Depth-first search



<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.4 Uninformed Search Strategies (8/11)

IV. Depth-limited search (DLS)

- Depth-first search with depth limit l
  - Node at depth l, have no successor

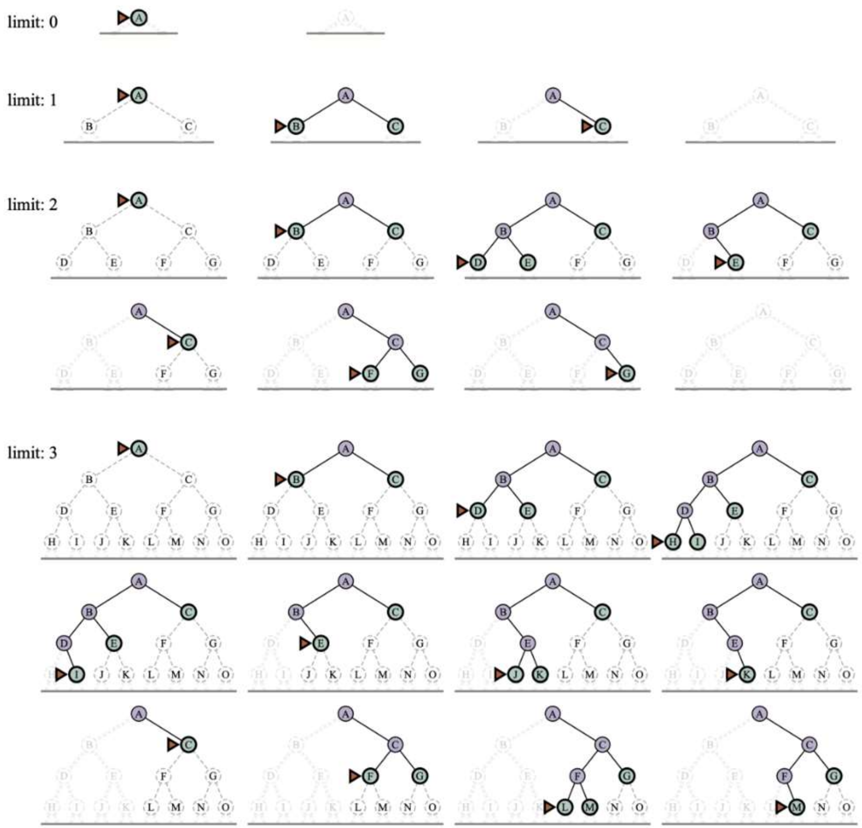
```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```


<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.4 Uninformed Search Strategies (9/11)

V. Iterative deepening search

- ➢General strategy often used in combination with DFS and find the best depth limit



<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.4 Uninformed Search Strategies (10/11)

V. Iterative deepening search

- ➢ Complete? Yes
- ➢ Time? $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d = O(b^d)$
- ➢ Space? $O(bd)$
- ➢ Optimal? Yes, if step cost = 1

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution node or *failure*
  for *depth* = 0 **to** ∞ **do**
    result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result* ≠ *cutoff* **then return** *result*

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.4 Uninformed Search Strategies (11/11)

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.5. Informed Search Strategies

3.5 Informed Search Strategies (1/7)

Best-first search

- ➢Idea: use an evaluation function for each node
  - Expand more desirable unexpanded node
- ➢Implement: queue sorted in decreasing order of desirability
- ➢Special cases:
  - **Greedy search**
  - **A* search**

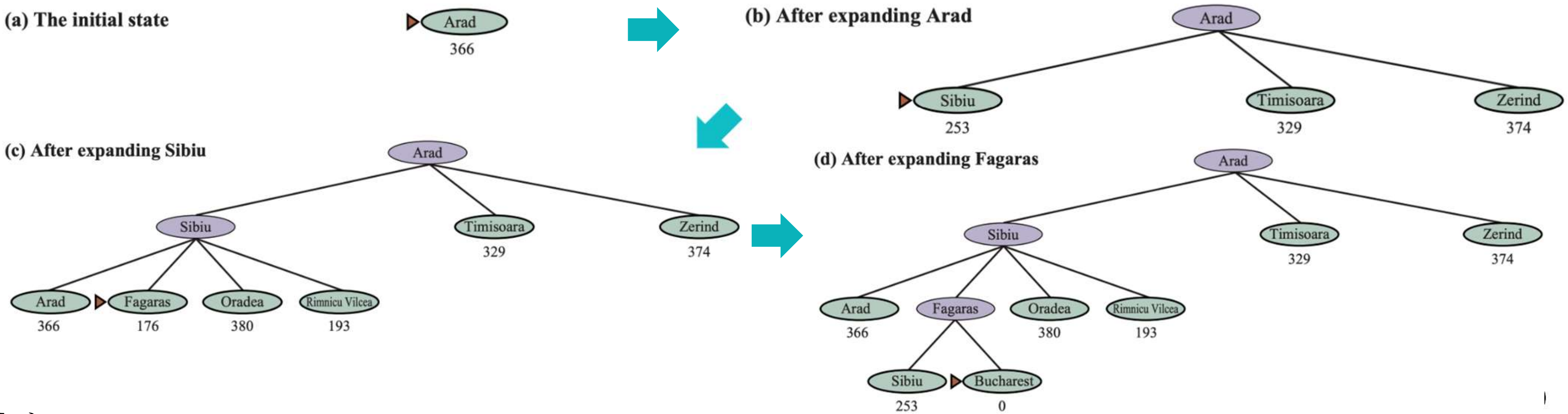
3.5 Informed Search Strategies (2/7)

Greedy search

➢ Evaluation function $h(n)$

■ Estimate of cost from n to the closest goal

➢ Greedy search expands the node that appears to be closest to goal



<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

3.5 Informed Search Strategies (3/7)

Greedy search

- Complete? No–can get stuck in loops,

  e.g., Iasi → Neamt → Iasi → Neamt → …
- Complete? in finite space with repeated-state checking
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$, keeps all nodes in memory
- Optimal? No

3.5 Informed Search Strategies (4/7)

A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost to goal from n
  - $f(n)$ = estimated total cost of path through n to goal
- A* search uses an admissible heuristic $h(n) \le h^*(n)$

$$\text{Start}\quad \overbrace{n_0 \qquad\qquad\qquad n}^{g(n)}\overbrace{\qquad\qquad n_g}^{h(n)}\quad \text{Goal}$$

3.5 Informed Search Strategies (5/7)

A* search

- Complete? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time? Exponential in [relative error in h × length of soln.]
- Space? Keeps all nodes in memory
- Optimal? Yes—cannot expand f_{i+1} until f_i is finished

3.5 Informed Search Strategies (6/7)

Conditions for optimality: Admissibility and consistency

- The first condition for optimality is that $h(n)$ be an **admissible heuristic**, which *never overestimates* the cost to reach the goal.
  - $h(n) \leq h^*(n)$
  - Admissible heuristics are by nature *optimistic* because they think the cost of solving the problem is less than it actually is.
- A second, slightly stronger condition is **consistency** (or **monotonicity**) is required for applications of A* to graph search.
  - $h(n) \leq c(n, a, n') + h(n')$
  - A heuristic $h(n)$ is consistent if the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'.

3.5 Informed Search Strategies (7/7)

Optimality of A*

$h(n) \leq c(n, a, n') + h(n')$

- A* has the following properties:
  - The tree search version of A* is optimal if $h(n)$ is admissible, while the graph-search version is optimal if $h(n)$ is consistent.
- We show the consistency of A* in two steps.
- Step 1: If $h(n)$ is consistent, then the value of $f(n)$ along any path are nondecreasing.
  - $g(n') = g(n) + c(n, a, n')$
  - $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geqq g(n) + h(n) = f(n)$
- Step 2: Whenever A* selects a node n for expansion, the optimal path to that node has been found.
  - Were this not the case, there would have to be another frontier node n' on the optimal path from the start node to n, (because f is nondecreasing along any path) n' would have lower f-cost than n and would have been selected first.

3.6. Heuristic Functions

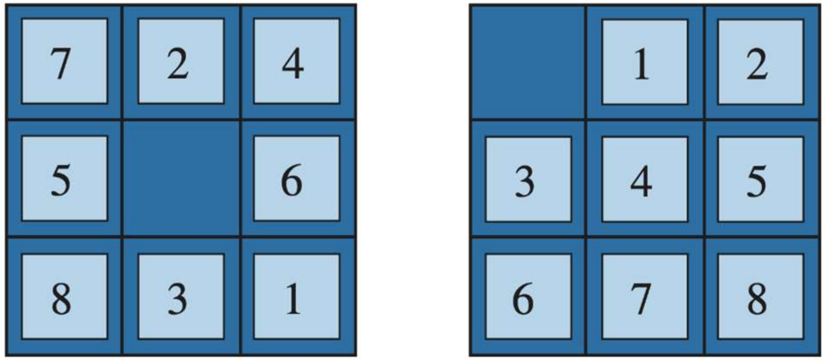
3.6 Heuristic Functions (1/3)

Example: the 8-puzzle

➢ Two heuristics:

$h_1(n)$ =number of misplaced tiles

$h_2(n)$ = total Manhattan distance



Start State Goal State

➢ $h_1(S) = 8$

➢ $h_2(n) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$

3.6 Heuristic Functions (2/3)

Example: the 8-puzzle

- Dominance
  - If $h_2(n) > h_1(n)$ for all n, then h_2 dominates h_1
- Typical search costs:

$d = 14$ IDS = 3,473,941 nodes
A$^*(h_1)$ = 539 nodes
A$^*(h_2)$ = 113 nodes
$d = 24$ IDS $\approx$ 54,000,000,000 nodes
A$^*(h_1)$ = 39,135 nodes
A$^*(h_2)$ = 1,641 nodes

- Given any admissible heuristics h_a, h_b,

  $h(n) = \max(h_a(n), h_b(n))(n))$ is also admissible and dominates

3.6 Heuristic Functions (3/3)

Example: the 8-puzzle

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem
- If a tile can move anywhere (in 8-puzzle)
  - then $h_1(n)$ gives the shortest solution
- If a tile can move to any adjacent square (in 8-puzzle)
  - then $h_2(n)$ gives the shortest solution
- Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem
- Given any admissible heuristics h_a, h_b,

  $h(n) = \max(h_a(n), h_b(n))$ is also admissible and dominates h_a, h_b

Summary

1. **Heuristic functions estimate costs of shortest paths**
2. **Good heuristics can dramatically reduce search cost**
3. **Greedy best-first search expands lowest h**
   - incomplete and not always optimal
4. **A* search expands lowest $g + h$**
   - Complete and optimal
   - Also optimally efficient (up to tie-breaks, for forward search)
5. **Admissible heuristics can be derived from exact solution of relaxed problems**