Introduction to Artificial Intelligence

Chapter 2: Solving Problems by Searching (1)

In which we see how an agent can find a sequence of action that achieves its goals when no single action will do.

Outline

1. Problem-Solving Agents
2. Example Problems
3. Implement the Search

1. Problem-Solving Agents

- Goal-based Agents
- A State-space Model
- Well-defined Problems and Solutions
- Formulating Problems

Goal-based Agents

Agents that take actions in the pursuit of a goal or goals.

Goal-based Agents

- What should a goal-based agent do when none of the actions it can currently perform results in a goal state?

- Choose an action that at least leads to a state that is closer to a goal than the current one is.

Goal-based Agents

Making that work can be tricky:

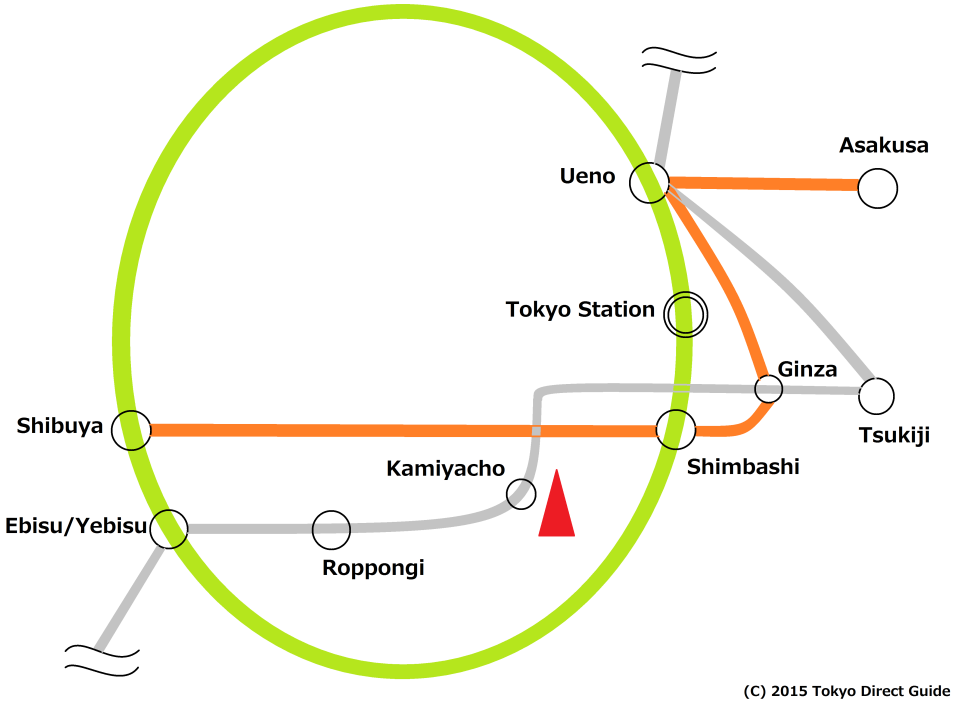
- What if one or more of the choices you make turn out not to lead to a goal?
- What if you're concerned with the best way to achieve some goal?
- What if you're under some kind of resource constraint?

Problems Solving as Search

One way to address these issues is to view goal-attainment as problem solving, and viewing that as a search through a state space.

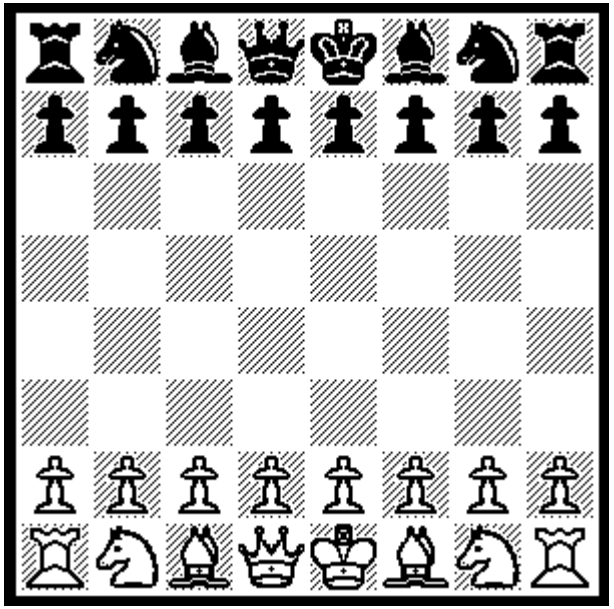
A State space Model

- State-space model:
  - The agent's model of the world
  - Usually a set of discrete states
  - E.g., in driving, the states in the model could be cities/ places visited



A State space Model

- Initial State:
  - Where we start the search
  - E.g., starting position on a chess board

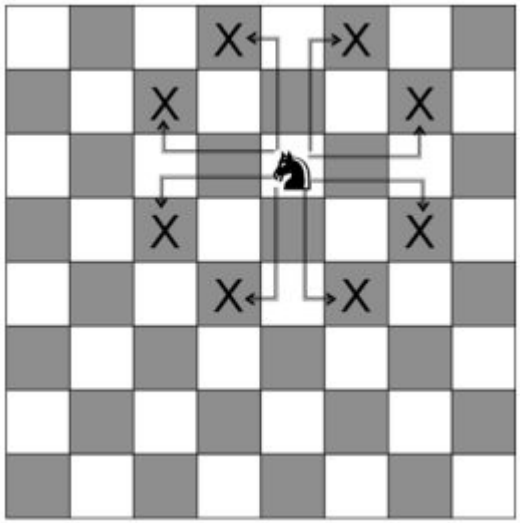


A State space Model

- Goal State(s):
  - A goal is defined as a desirable state for an agent
  - May be 1 goal
    - E.g., drive to Tokyo
  - Or many goals
    - E.g., drive to districts that have large department stores in Tokyo
    - Shinjuku? Ginza? Shibuya? ...

A State space Model

- Operators/Actions:
  - Legal actions which the agent can take to move from one state to another
  - E.g., legal move of a knight in chess board



A State space Model

- Basic search problem:
  - Find a sequence of state transitions leading from the *start state* to a *goal state*.

Formulating Problems

- Formally, a problem is characterized by:
  - A state space
    - an implicitly specified set of states
  - An initial state
  - A set of actions
    - successors: state → set of states
  - A goal test
    - goal: state → true or false
  - A path cost (optional)
    - edge/steps between states → cost

cuu duong than cong . com

What is a Solution?

- A sequence of actions that when performed will transform the initial state into a goal state
  - e.g., the sequence of actions that gets the missionaries safely across the river
- Or sometimes just the goal state
  - e.g., infer molecular structure from mass spectrographic data

A simple 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
```

cuu duong than cong . com

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

2. Example Problems

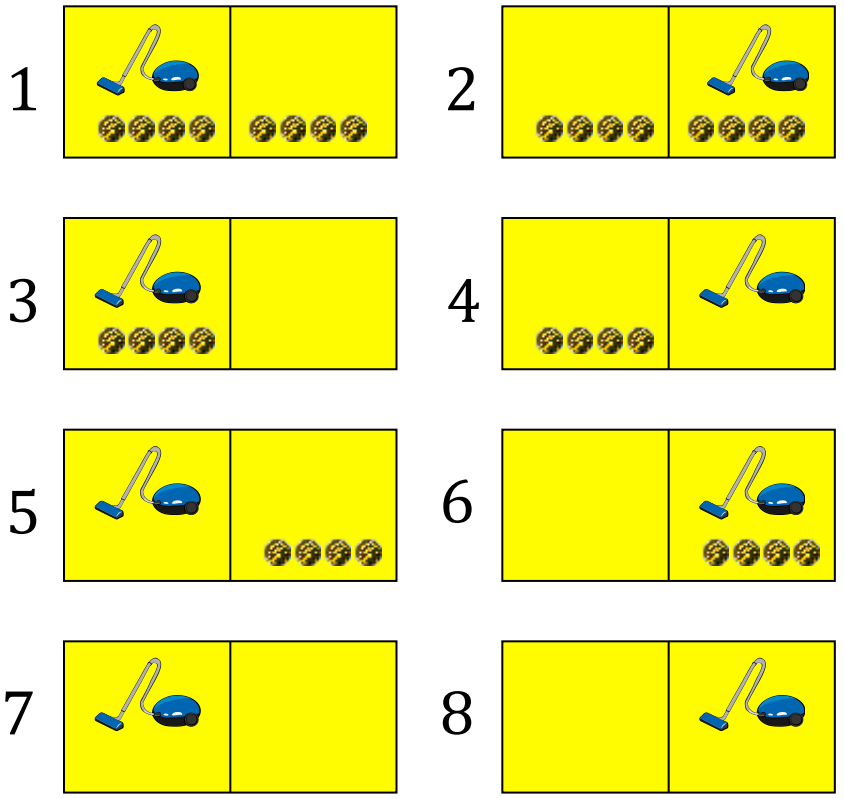
- Toy Examples
  - The Vaccum World
  - Missionaries and cannibals
  - Route Finding: Romania Holiday
  - The 8 Puzzle
- Real-world Examples

Example Problems

- Toy problems (but sometimes useful)
  - Illustrate or exercise various problem-solving methods
  - Concise, exact description
  - Can be used to compare performance
  - *Examples*: 8-puzzle, 8-queens problem, Cryptarithmetic, Vacuum world, Missionaries and cannibals, simple route finding
- Real-world problem
  - More difficult
  - No single, agreed-upon description
  - *Examples*: Route finding, Touring and traveling salesperson problems, VLSI layout, Robot navigation, Assembly sequencing

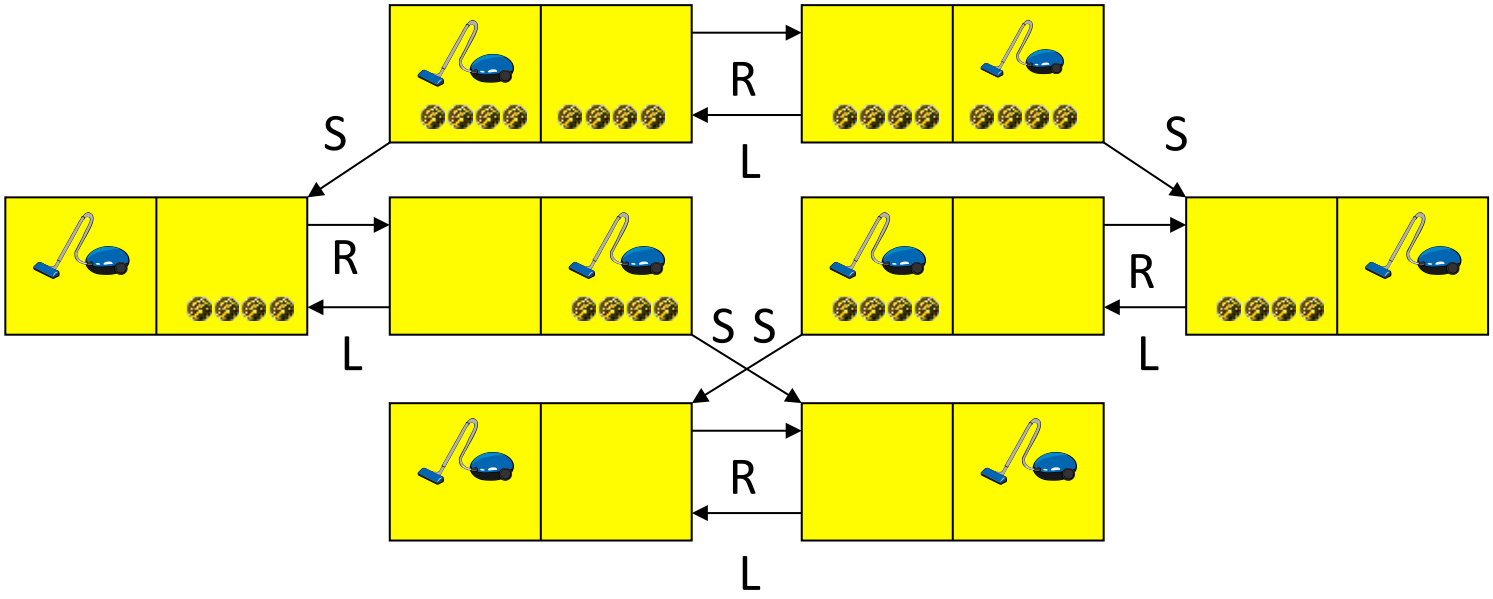
Toy Problems: *The vacuum world*

- The world has only two *locations*
- Each location may or may not contain *dirt*
- The agent may be in one location or the other
- 8 possible *world states*
- Three possible actions: *Left, Right, Suck*
- *Goal*: clean up all the dirt

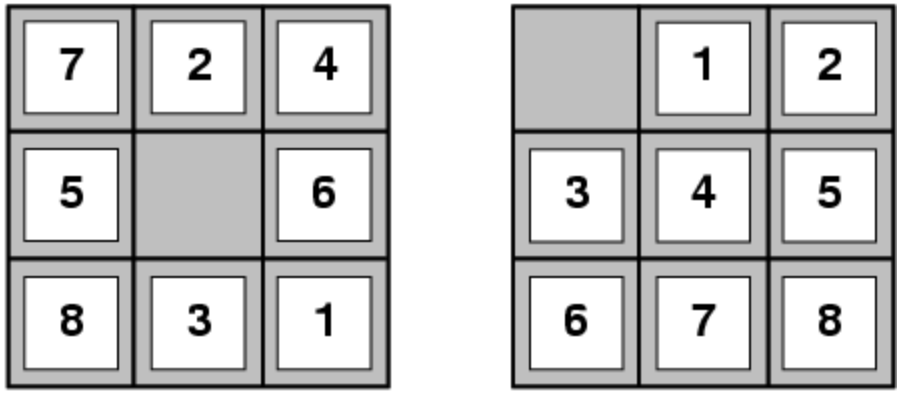


Toy Problems: *The vacuum world*

- *States*: one of the 8 states given earlier
- *Initial states:* given
- *Actions*: move left, move right, suck
- *Goal test*: no dirt left in any square
- *Path cost*: each action costs one



Example: The 8-puzzle

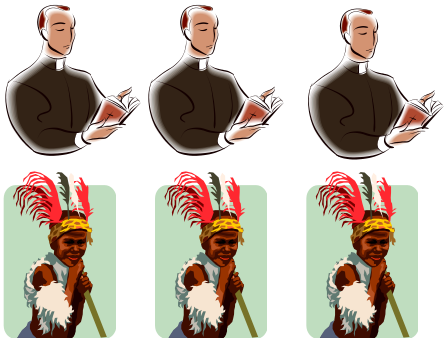


- states? locations of tiles
- initial state? given
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: Missionaries and cannibals

- Missionaries and cannibals
  - Three missionaries and three cannibals want to cross a river
  - There is a boat that can hold two people
  - Cross the river, but make sure that the missionaries are not outnumbered by the cannibals on either bank
- Needs a lot of *assumptions*
  - Crocodiles in the river, the weather and so on
  - Only the endpoints of the crossing are important
  - Only two types of people



Example: Missionaries and cannibals

- Problem formulation
  - *States*:
    - ordered sequence of three numbers representing the number of missionaries, cannibals and boats on the bank of the river from which they started.
    - The start state is (3, 3, 1)
  - *Actions*: take two missionaries, two cannibals, or one of each across in the boat
  - *Goal test*: reached state (0, 0, 0)
  - *Path cost*: number of crossings

Example: Romania Holiday

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow to Bucharest
- Formulate problem:
  - states: various cities
  - initial state: be in Arad
  - goal state: be in Bucharest
  - actions: drive between cities
- Solution:
  - sequence of cities starts from Arad to Bucharest
  - e.g., Arad, Sibiu, Fagaras, Bucharest
- Path cost:
  - Sum of distances on the route/number of cities visited

Example: Romania Holiday

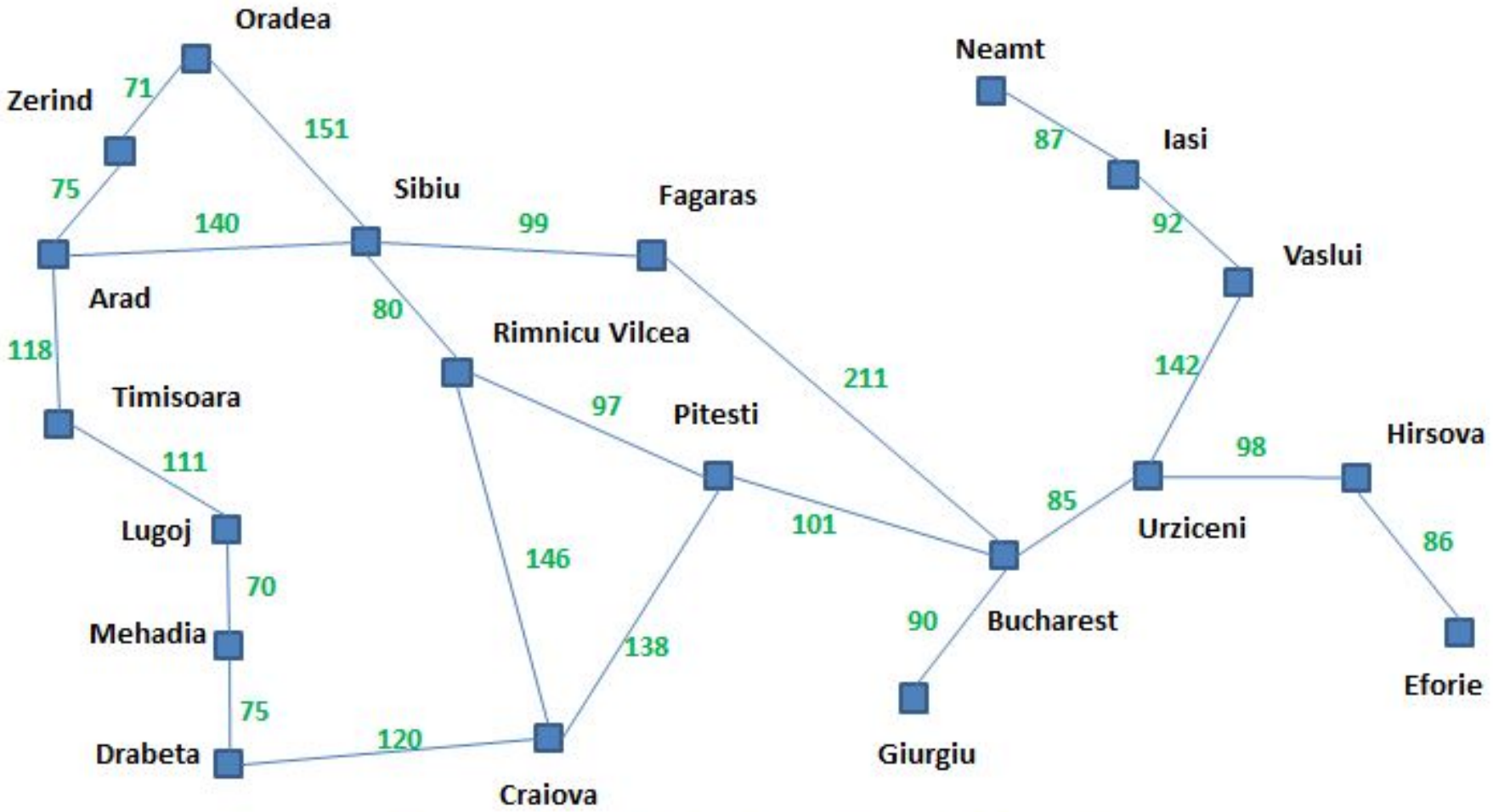


Figure A simplified road map of part of Romania.

Abstraction

- Definition of Abstraction:
  - Process of removing irrelevant detail to create an abstract representation: ``high-level", ignores irrelevant details
- Navigation Example: how do we define states and operators?
  - First step is to abstract "the big picture"
    - i.e., solve a map problem
    - nodes = cities, links = freeways/roads (a high-level description)
    - this description is an abstraction of the real problem
  - Can later worry about details like freeway onramps, refueling, etc

Abstraction

- Abstraction is critical for automated problem solving
  - must create an approximate, simplified, model of the world for the computer to deal with: real-world is too detailed to model exactly
  - good abstractions retain all important details

Initial Assumptions

- The agent knows its current state
- Only the actions of the agent will change the world
- The effects of the agent's actions are known and deterministic

All of these are defeasible… likely to be wrong in real settings.

Real-world problems

- Route finding
  - Specified locations and transition along links between them
  - *Applications*:
    - Routing in computer networks
    - Automated travel advisory systems
    - Airline travel planning systems
    - Robot navigation
    - Automatic Assembly Sequencing

Real-world problems

- Airplane travel problems by a travel-planning Web site:
  - State:
    - location, time, fare base, domestic/international flight, …
  - Initial state:
    - specified by the user's query.
  - Actions:
    - Take any flight from current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
  - Goal test:
    - Are we at the final destination specified by the user?
  - Path cost:
    - Money, waiting time, flight time, customs, immigration procedures, seat class, time of the day, ….

Real-world problems

- Touring and traveling salesperson problems (TSP)
  - "Visit every city on the map at least once and end in Bucharest"
  - Needs information about the visited cities
  - *Goal*: Find the shortest tour that visits all cities
  - *NP-hard*, but a lot of effort has been spent on improving the capabilities of TSP algorithms

3. Implement the Search

- Infrastructure for search algorithms
- Tree seach
- Graph search
- Measuring problem-solving performance

Infastructure for search algorithms

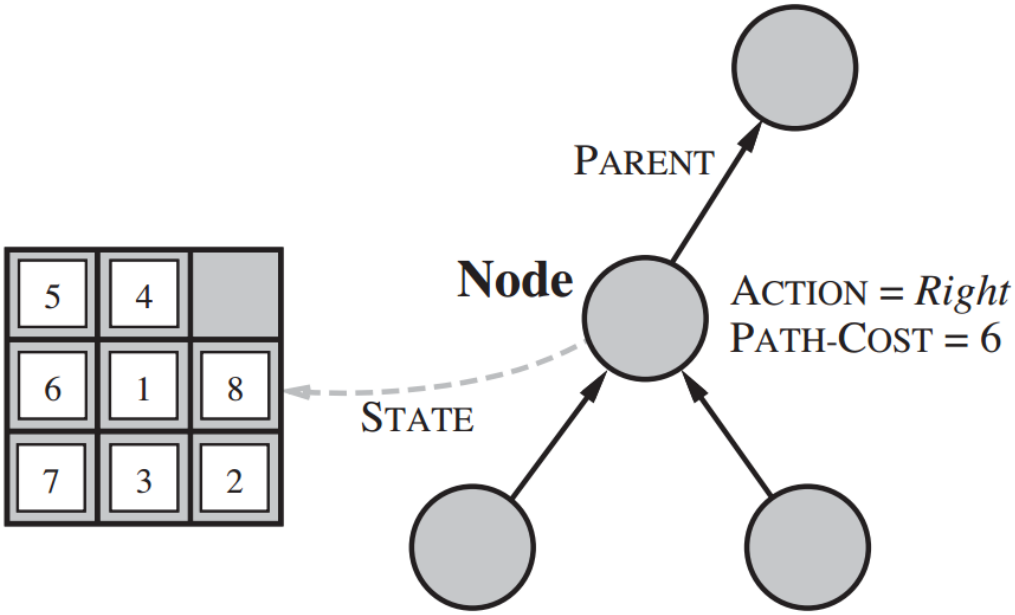
- Solution = sequence of actions = a search tree
  - Nodes = states
  - Edges (Branches) = actions

- The set of all leaf nodes available for expansion at any given point is called the frontier

Infastructure for search algorithms

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes
  - STATE: the state in the state space to which the node correspond
  - PARENT: the node in the search tree that generated this node
  - ACTION: the action that was applied to the parent to generate the node
  - PATH COST: $g(n)$, the cost from the initial state to the node

Infastructure for search algorithms



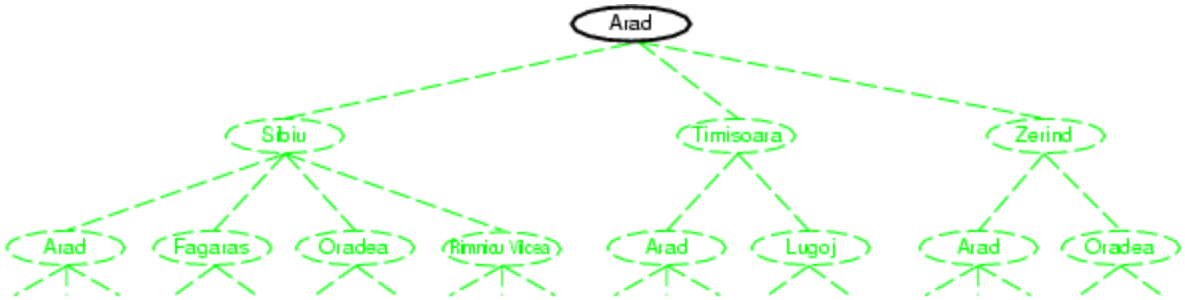
- The **`Expand`** function creates new nodes, filling in the various fields and using the **`SuccessorFn`** of the problem to create the corresponding states.

Tree search algorithms

- Basic idea:
  - Exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)
  - Every state is evaluated: is it a goal state?

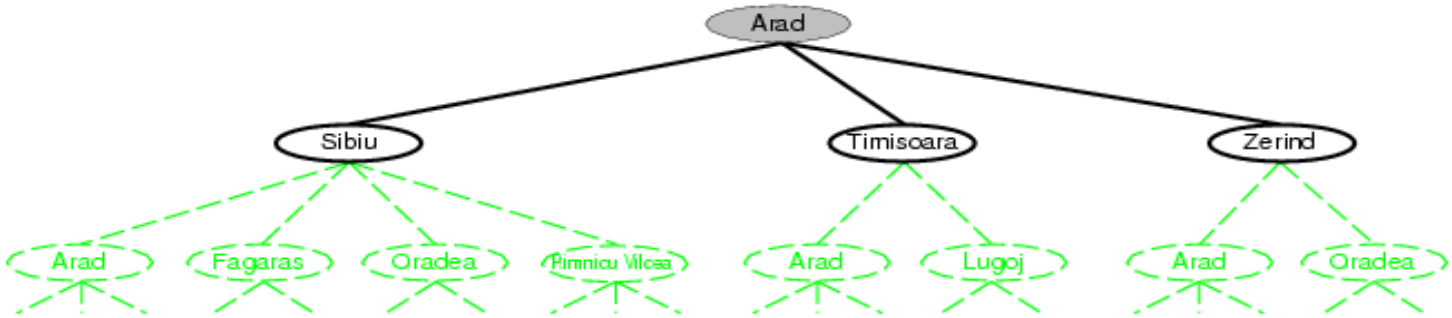
```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding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

Tree search example: Romania



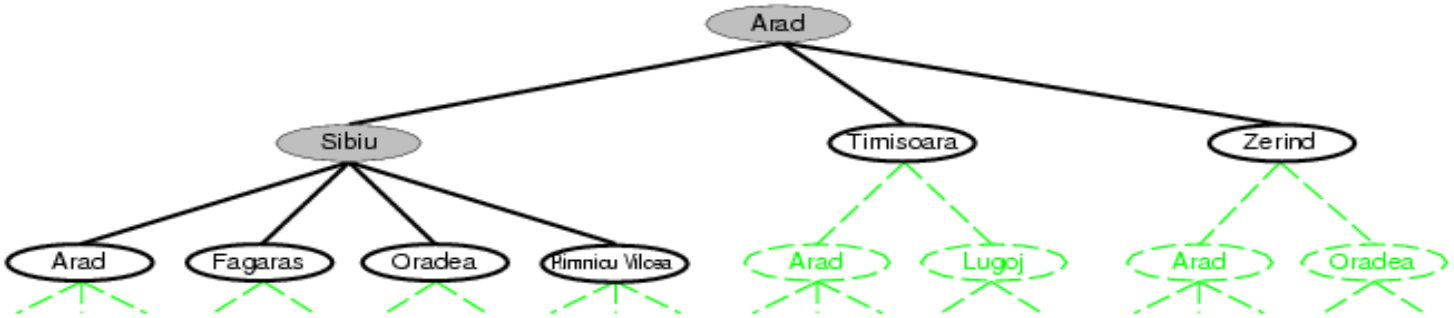
The initial state

Tree search example: Romania



After expanding Arad

Tree search example: Romania



After expanding Sibiu

Handling Repeated States

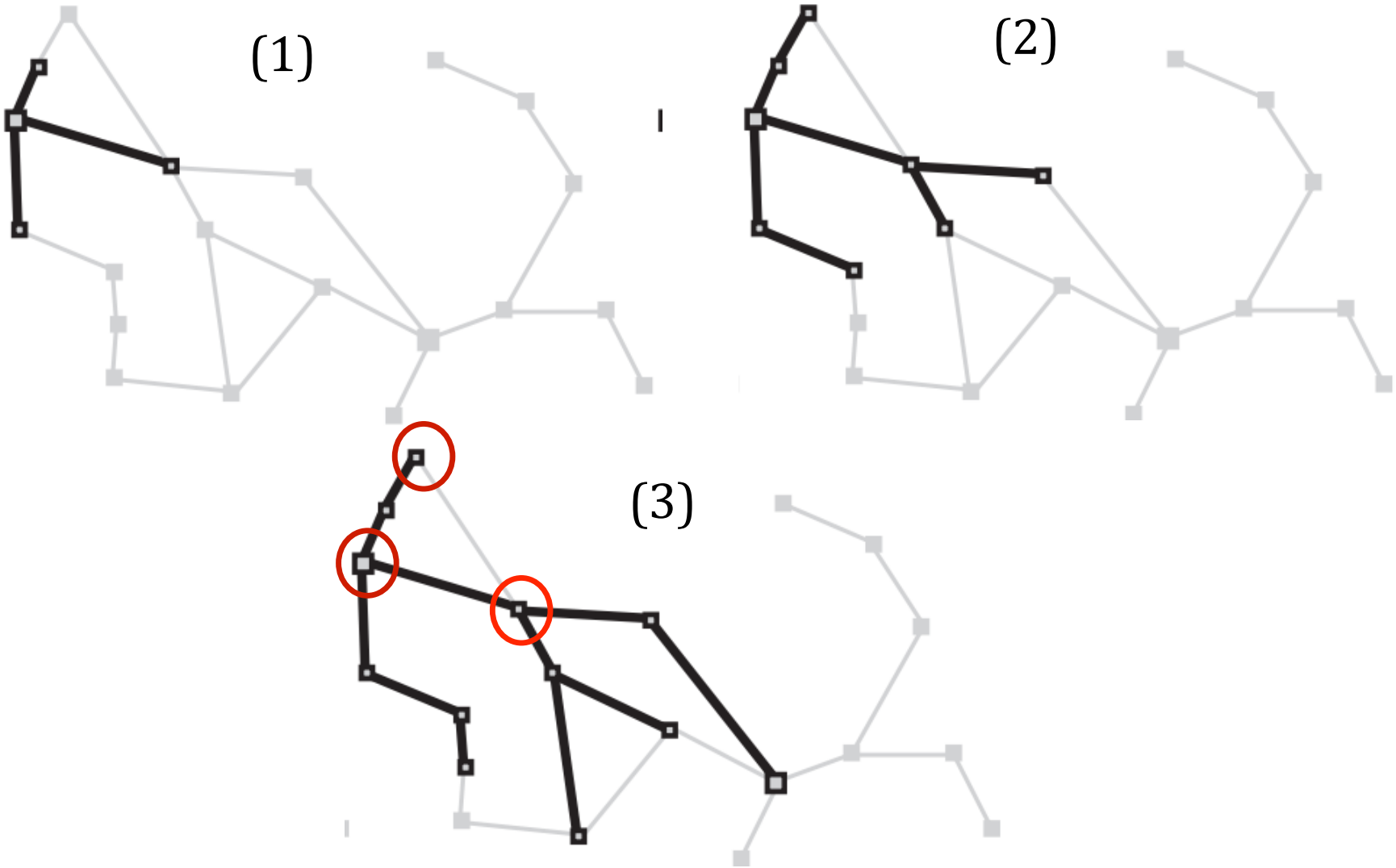
- Failure to detect repeated states (e.g., in 8 puzzle) can cause infinite loops in search
- In practice, the solution space can be a graph, not a tree
  - More general approach is graph search
  - Tree search can end up repeatedly visiting the same nodes
    - Unless it keeps track of all nodes visited
    - …but this could take vast amounts of memory
- How about redundant paths?
  - We can avoid it
  - Sometime, we cannot…

Graph Search Algorithms

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

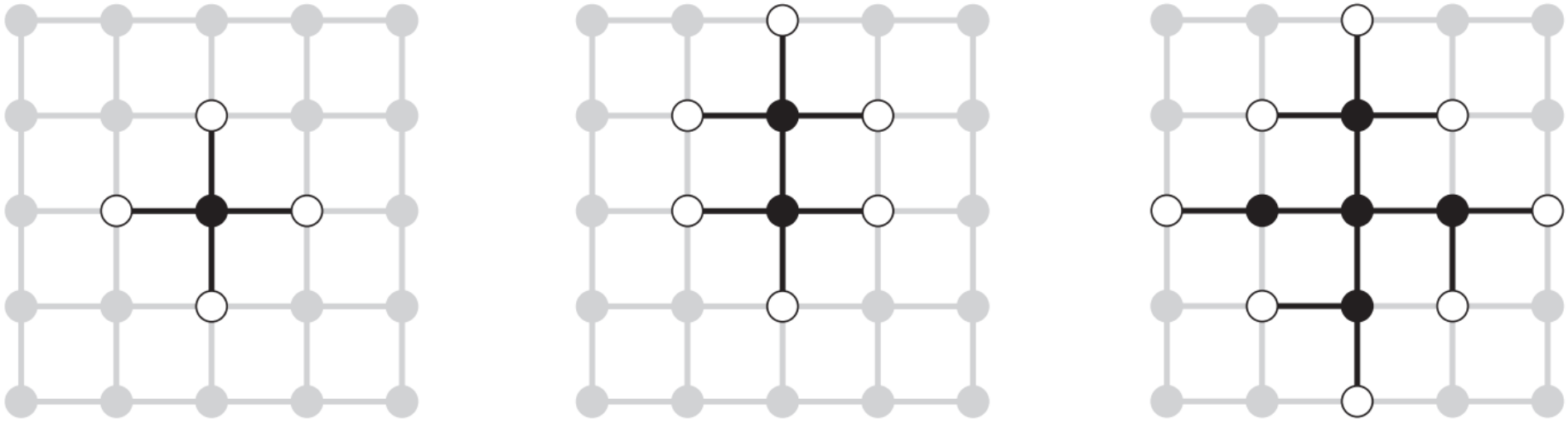
Data structures that can be used:
Queue, Stack, Priority Queue

Graph Search example: Romania



Graph Search Algorithms

- The separation property of Graph Search Algorithms:



(a) (b) (c)

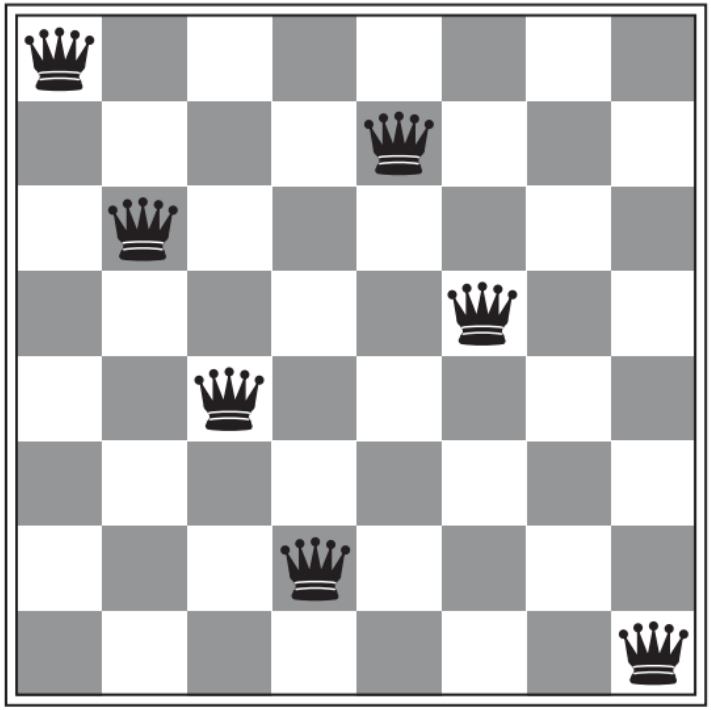
Measuring problem-solving performance

- Search Strategies are evaluated along the following dimensions:
  - Completeness: does it always find a solution if one exists?
  - Time complexity: how long does it take to find a solution?
  - Space complexity: how much memory is needed to perform the search?
  - Optimality: does it always find a least-cost solution?

- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Group Discussion

- 8-queens problem:
  - Goal: place 8 queens on a chess board such that no queen attacks any other
  - *Formulate the problem by searching*
    - States?
    - Initial State?
    - Actions?
    - Goal Test?
  - *What is the size of your state space?*



A nearly goal state

Next class

- Chapter 2: Solving Problems by Searching (cont.)
  - Uninformed Search

Group Assignment 1

- Given a graph with nodes and links, we can find the shortest path using Dijkstra's algorithm. It is not hard. We have a polynomial time algorithm to do that.
- In AI we also solving the graph search problems.
- What is the differences between these two graph search strategies? (not AI and AI)
- What is special about AI Search Algorithms? Give a specific example to explain for your ideas.
  - Hint: Why you cannot use the size of the state space graph to measure the problem difficulty (the cost of the algorithm) in AI?