Title: Solving Problems by Searching

AIMA: Chapter 3 (Sections 3.1, 3.2 and 3.3)

Introduction to Artificial Intelligence

CSCE 476-876, Spring 2012

URL: `www.cse.unl.edu/~choueiry/S12-476-876`

Berthe Y. Choueiry (Shu-we-ri)

(402)472-5444

Summary

Intelligent Agents

- Designing intelligent agents: PAES
- Types of Intelligent Agents
  1. Self Reflex
  2. ?
  3. ?
  4. ?
- Types of environments: observable (fully or partially), deterministic or stochastic, episodic or sequential, static vs. dynamic, discrete vs. continuous, single agent vs. multiagent

Outline

- Problem-solving agents
- Formulating problems
  - Problem components
  - Importance of modeling
- Search
  - basic elements/components
  - Uninformed search (Section 3.4)
  - Informed (heuristic) search (Section 3.5)

Simple reflex agent unable to plan ahead

- actions limited by current percepts
- no knowledge of what actions do
- no knowledge of what they are trying to achieve

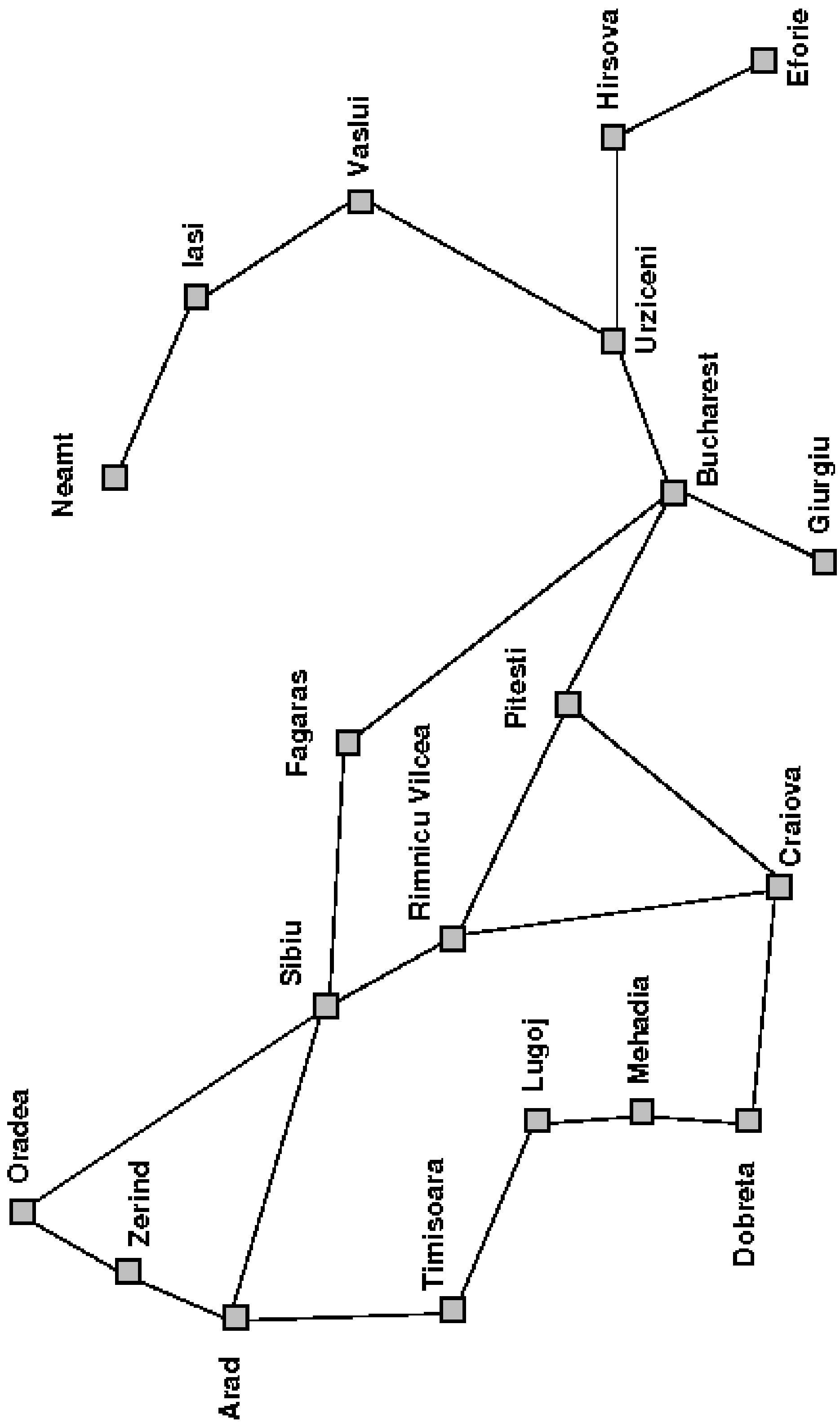
Problem-solving agent: goal-based agent

Given:

- a problem formulation: a set of states and a set of actions
- a goal to reach/accomplish

Find:

- a sequence of actions leading to goal


Arad
Zerind
Oradea
Timisoara
Sibiu
Lugoj
Mehadia
Dobreta
Rimnicu Vilcea
Craiova
Fagaras
Pitesti
Bucharest
Giurgiu
Neamt
Iasi
Vaslui
Urziceni
Hirsova
Eforie

Example: Holiday in Romania

On holiday in Romania, currently in Arad, want to go to Bucharest

Formulate goal:

be in Bucharest

Formulate problem:

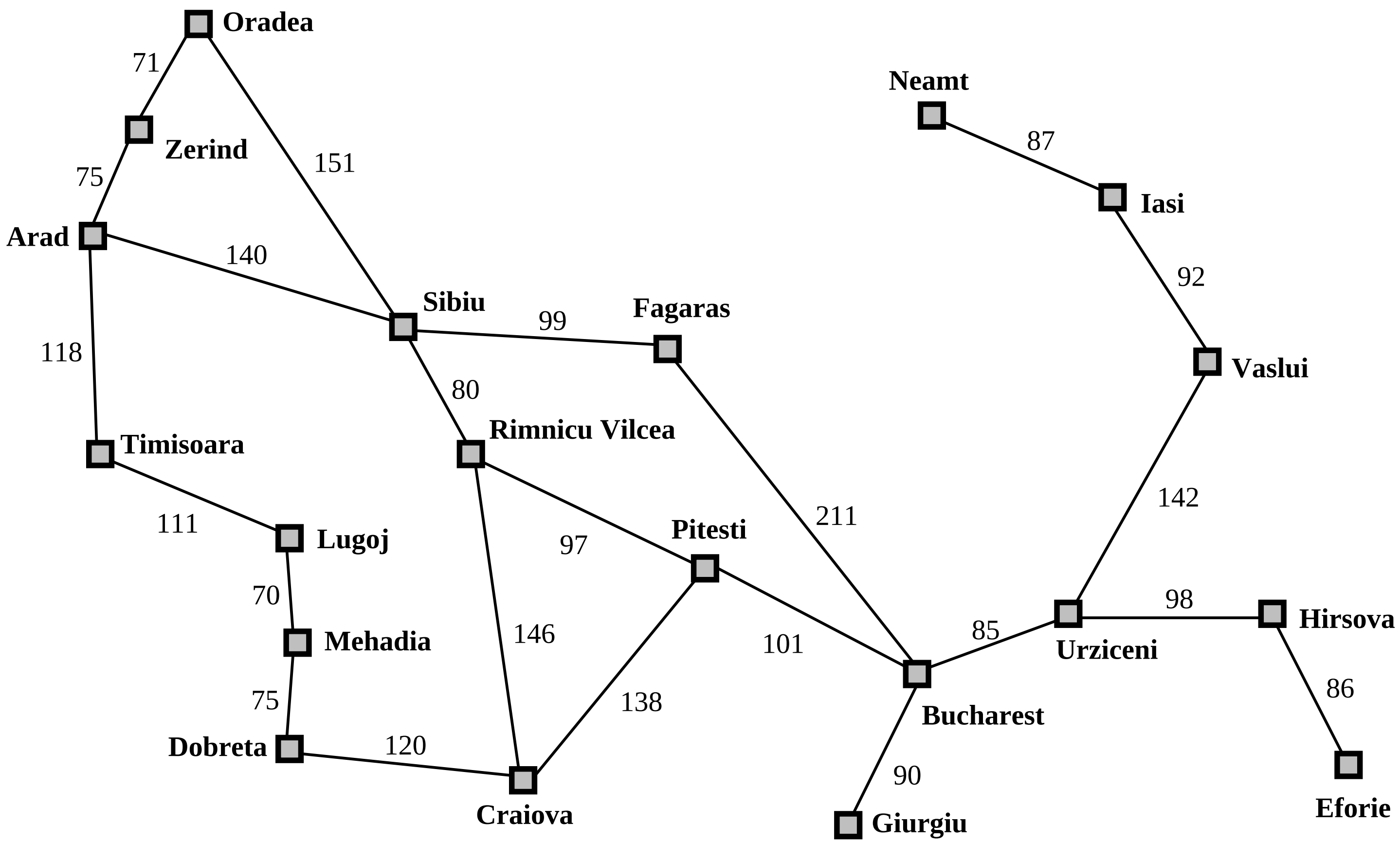
states: various cities

actions: (operators, successor function) drive between cities

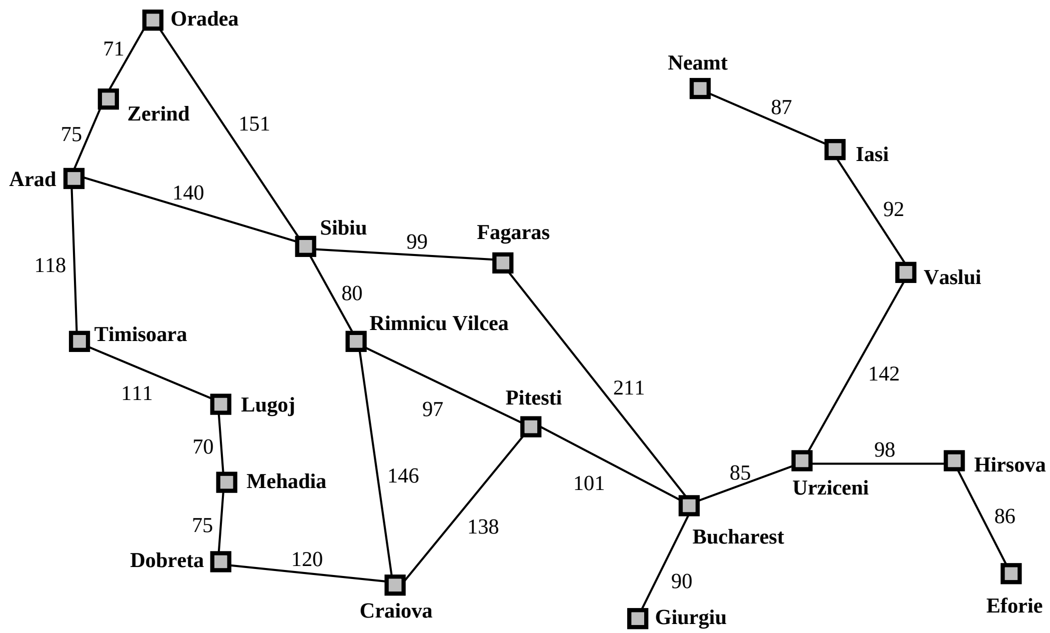
Find solution:

sequence of cities, *e.g.* Arad, Sibiu, Fagaras, Bucharest

Drive to Bucharest... how many roads out of Arad?



Use map to consider hypothetical journeys through each road until reaching Bucharest



Looking for a sequence of actions $\longrightarrow$ search

Sequence of actions to goal $\longrightarrow$ solution

Carrying out actions $\longrightarrow$ execution phase

Formulate, search, execute

Formulate, search, execute

```
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
```

$\times$	Update-State	$\times$	Formulate-goal
$\surd$	Formulate-Problem	$\surd$	Search

Recommendation = first, and Remainder = rest

Assumptions for environment: observable, static, discrete, deterministic sequential, single-agent

Problem formulation

A *problem* is defined by the following items:

1. *initial state*: $In(Arad)$
2. *successor function* $S(x)$ (operators, actions)
   Example, $S(In(Arad)) = \{\langle Go(Sibiu), In(Sibiu)\rangle, \langle Go(Timisoara), In(Timisoara)\rangle, \langle Go(Zerind), In(Zerind)\rangle\}$
3. *goal test*, can be explicit, e.g., $x = In(Bucharest)$
   or a property $NoDirt(x)$
4. *step cost*: assumed non-negative
5. *path cost* (additive)
   e.g., sum of distances, number of operators executed, etc.

A *solution* is a sequence of operators leading from the initial state to a goal state.
Solution quality, optimal solutions.

Importance of modeling (for problem formulation)

Real art of problem solving is modeling,

$$\text{deciding what's in} \begin{cases} \text{state description} \\ \text{action description} \end{cases}$$

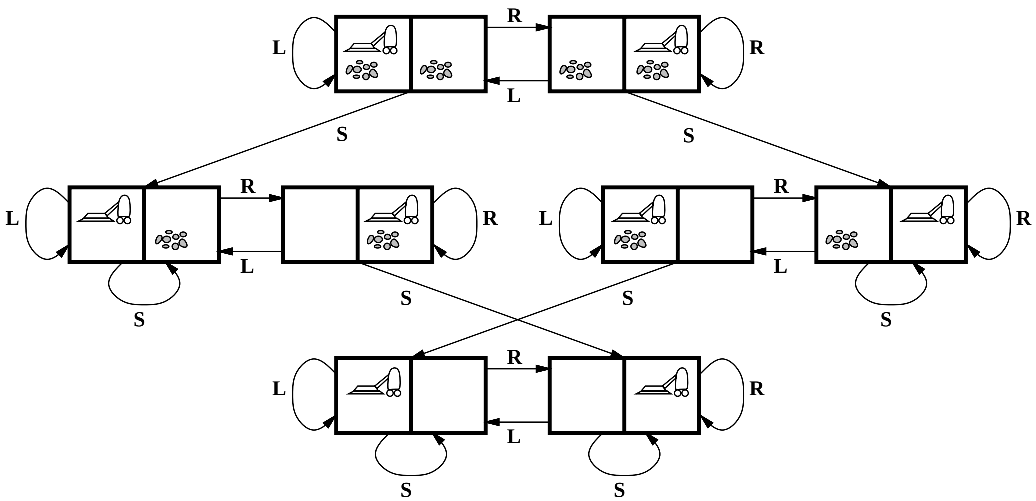
choosing the right level of abstraction

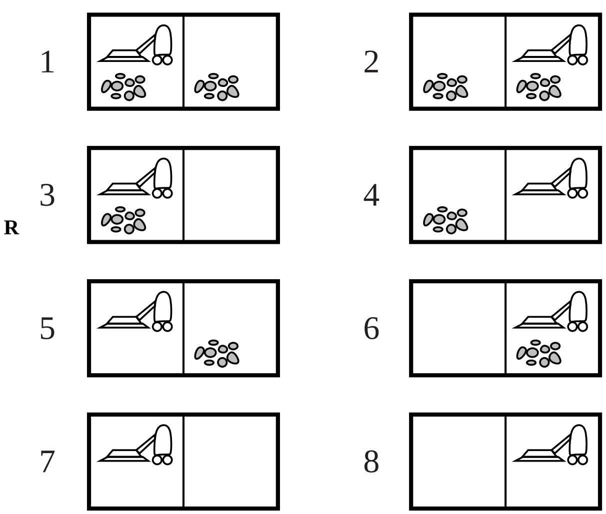
State abstraction: road maps, weather forecast, traveling companions, scenery, radio programs, ...

Action abstraction: generate pollution, slowing down/speeding up, time duration, turning on the radio, ..

Combinatorial explosion. Abstraction by removing irrelevant detail make the task easier to handle

State space vs. state set





Example problems

Toy Problems:

$\rightarrow$ intended to illustrate or exercise $\begin{cases} \text{concepts} \\ \text{problem-solving methods} \end{cases}$

$\surd$ can be give concise, exact description

$\surd$ researchers can compare performance of algorithms

$\times$ yield methods that rarely scale-up

$\times$ may reflect reality inaccurately (or not at all)

Real-world Problems:

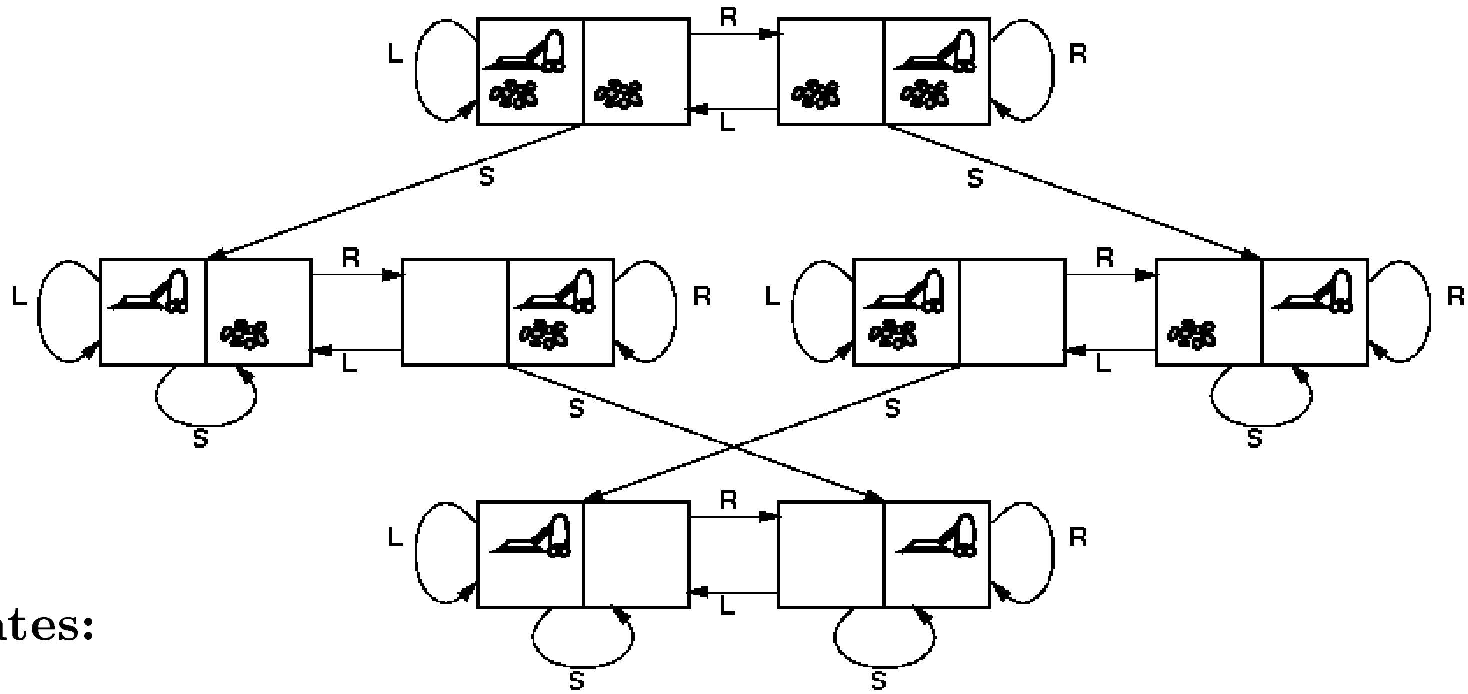
$\rightarrow$ more difficult but whose solutions people actually care about

$\surd$ more credible, useful for practical settings

$\times$ difficult to model, rarely agreed-upon descriptions

Toy problem: vacuum

Single state case



States:

Initial State:

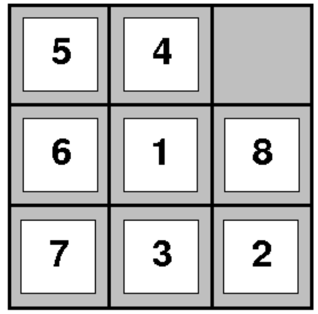
Successor function:

Goal test:

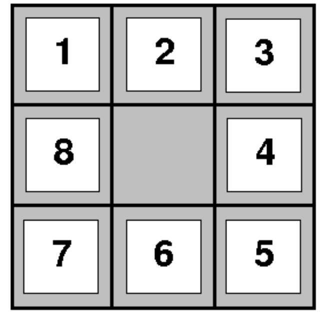
Path cost:

With 2 locations: 2.2^2 states. With n locations: $n.2^n$ states

Toy problem: 8-puzzle



Start State



Goal State

States:

Initial state:

Successor function:

Goal test:

Path cost:

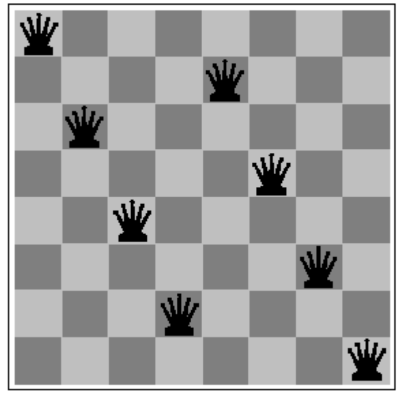
→ instance of sliding-block puzzles, known to be **NP**-complete

→ Optimal solution of n-puzzle **NP**-hard

→ so far, nothing better than search

→ 8-puzzle, 15-puzzle traditionally used to test search algorithms

Toy problem: n-Queens



$\rightarrow$ Formulation: incremental vs. complete-state

States: Any arrangement of $x \leq 8$ queens on board

Initial state:

Successor function: add a queen (alt., move a queen)

Goal test: 8 queens not attacking one another

Path cost: irrelevant (only final state matters)

$\rightarrow$ 64^8 possible states, but $\exists$ other more effective formulations

Toy problems: requiring search

$\surd$	8 puzzles
$\surd$	n-queens
$\surd$	vacuum
Others:	Missionaries & cannibals, farmer's dilemma, etc.

Real-world problems: requiring search

- Route finding: state = locations, actions = transitions
  routing computer networks, travel advisory, etc.
- Touring: start in Bucharest, visit every city at least once
  Traveling salesperson problem (TSP) (exactly once, shortest tour)
- VLSI layout: cell layout, channel layout
  minimize area and connection lengths to maximize speed
- Robot navigation (continuous space, 2D, 3D, $ldots$)
- Assembly by robot-arm
  States: robot joint angles, robot and parts coordinates
  Successor function: continuous motions of the robot joins
  goal test: complete assembly
  path cost: time to execute
- + protein design, internet search, etc. (check AIMA)

Problem solving performance

Measures for effectiveness of search:

1. Does it find a solution? complete
2. Is it a good solution? path cost low
3. Search cost? time & space

Total cost = Search cost + Path cost
$\longrightarrow$ problem?

Example: Arad to Bucharest
Path cost: total mileage, fuel, tire wear f(route), etc.
Search cost: time, computer at hand, etc.

So far

- Problem-solving agents

  Formulate, Search, Execute

- Formulating problems
  - Problem components: States, Initial state, Successor function, Goal test, Step cost, Path cost

    Solution: sequence of actions from initial state to goal state
  - Importance of modeling

Now, search

- Terminology: tree, node, expansion, fringe, leaf, queue, strategy
- Implementation: data structures
- Four evaluation criteria.. ?

Search: generate action sequences

partial solution: sequence yielding a (non goal) intermediate state

$$\text{Search} \left\{ \begin{array}{c} \text{generate} \\ \text{maintain} \end{array} \right\} \text{a set of sequences of partial solutions}$$

Two aspects:

1. how to generate sequences
2. which data structures to keep track of them

Search generate action sequences

Basic idea:

offline, simulated exploration of state space

by generating successors of already-explored states

$\rightarrow$ *expanding* states

Start from a state, test if it is a goal state

If it is, we are done

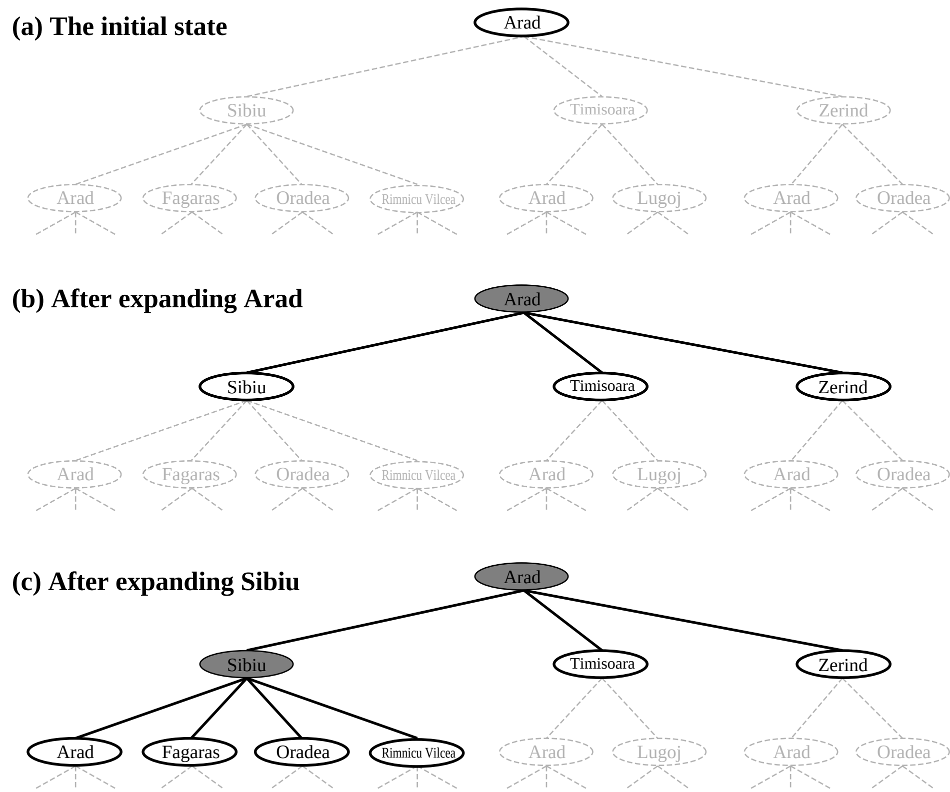
If it is not: *expand state*

Apply all operators applicable to current state to

generate all possible sequences of future states

now we have set of partial solutions

...



$$\text{Search tree, nodes} \begin{cases} \text{root: initial state} \\ \text{leaves: states that can/should not be expanded} \end{cases}$$

Data structure

LHW Chapter 13

A node x has a parent, children, depth, path cost $g(x)$

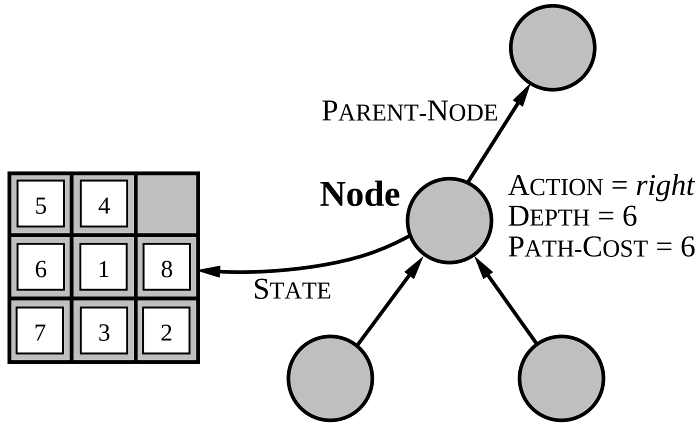
A data structure for a search node

$$\begin{cases} State[x] & \text{state in space state} \\ Parent-Node[x] & \text{parent node} \\ Action[x] & \text{operator used to generate node} \\ Path-Cost[x] & \text{path cost of parent+cost step, path cost } g(x) \\ Depth[x] & \textbf{depth}\text{: \# nodes from root (path length)} \end{cases}$$

Nodes to be expanded

$$\begin{cases} \text{constitute a fringe (frontier)} \\ \text{managed in a queue,} \\ \text{order of node expansion determines search strategy} \end{cases}$$

Warning:



Do not confuse: State space and Search (tree) space

Holiday in Romania:
- What is a state?
- What is the state space?
- What is the size of state space?
- What is the size of search tree ?

A node has a parent, children, depth, path cost $g(x)$

A state has no parent, children, depth, etc..

Types of Search

Uninformed: use only information available in problem definition

Heuristic: exploits some knowledge of the domain

Uninformed search strategies:

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

Search strategies

Criteria for evaluating search:

1. Completeness: does it always find a solution if one exists?
2. Time complexity: number of nodes generated/expanded
3. Space complexity: maximum number of nodes in memory
4. Optimality: does it always find a least-cost solution?

Time/space complexity measured in terms of:

- b: maximum branching factor of the search tree
- d: depth of the least-cost solution
- m: maximum depth of the search space (may be ∞)