Solving problems by searching

Problem Solving Agents

- When the correct action to take is not immediately obvious, an agent may need to plan ahead: to consider a sequence of actions that form a path to a goal state.
- A kind of “goal based” agent
- Finds sequences of actions that lead to desirable states.

The Romania problem

- An agent is in the city of Arad, Romania, enjoying a touring holiday
- The agent wants to
  - Improve its Romanian
  - Enjoy sightseeing
  - Improve its suntan
- The agent has a nonrefundable ticket to fly out of Bucharest the following day

Goal: Reach Bucharest on time

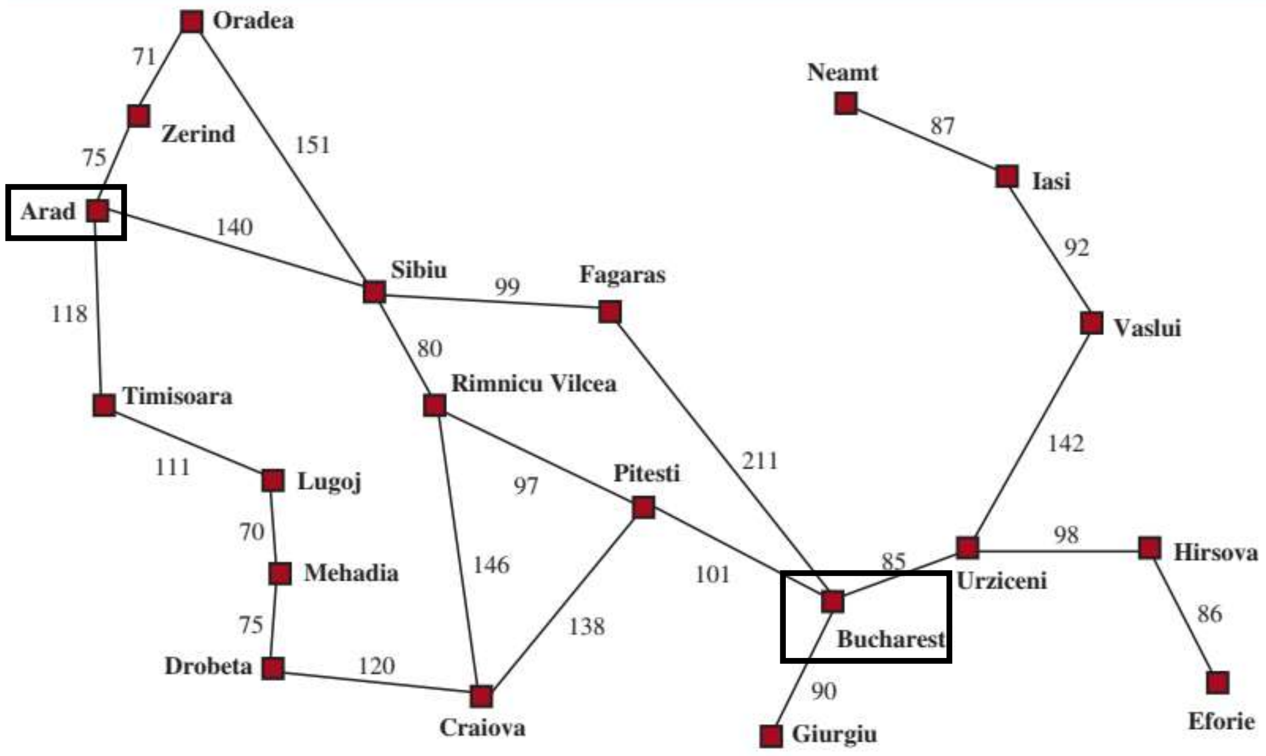


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Four phase problem solving process

- **Goal formulation**: The agent adopts the goal of reaching Bucharest. Goals organize behavior by limiting the objectives and hence the actions to be considered.
- **Problem formulation**: The agent plans a description of the states and actions necessary to reach the goal (going from one city to adjacent city)
- **Search**: Before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a **solution.**
- **Execution:** The agent can now execute the actions in the solution, one at a time.

Well defined problems and solutions

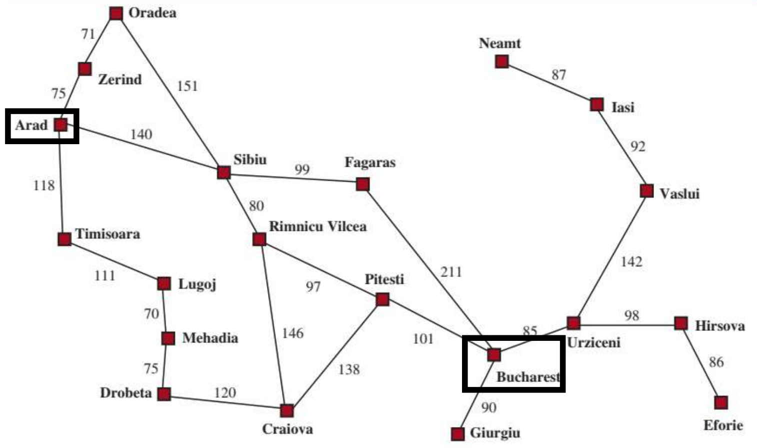


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

- A problem can be defined formally by five components:
  - The **initial state** that the agent starts in
    - Initial state for our agent = In (Arad)
- A description of possible actions available to the agent
  - From the state In (Arad) the applicable actions are:

  {Go(Sibiu), Go (Timisoara), Go(Zerind)}
- A description of what each action does (transition model)
  - Successor = any state reachable from a given state by a single action
  - RESULT (In(Arad), Go (Zerind)) = In (Zerind)
  - The initial state, actions and transition model implicitly define the **state space of the problem** – the set of all states reachable from the initial state by any sequence of actions
  - **The state space forms a directed network (graph)** in which the nodes are states and the links are actions
- The **goal test**, which determines whether a given state is a goal state
  - Goal States = {In(Bucharest)}
- A path cost function that assigns **numeric cost to each path**

Optimal solution

- A solution to a problem is:
  - An action sequence that leads from the initial state to goal state
- Solution quality is measured by the path cost function.
- An optimal solution has the lowest path cost among all solutions.

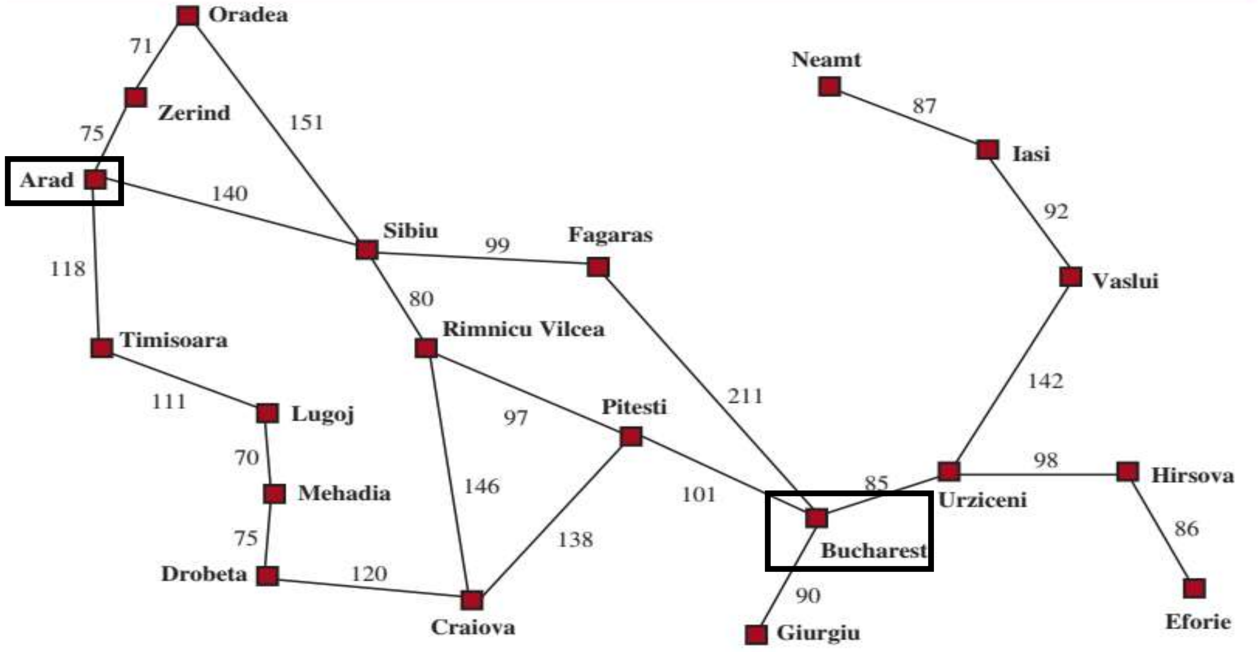


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Abstraction when formulating problems

- There are many details in the actual world!
- **Actual World State** = the travelling companions, the scenery out of the window, the condition of the road, the weather, etc.
- **Abstract Mathematical State** = In (Arad)
- We left out all other considerations in the state description because they are irrelevant to the problem of finding a route to Bucharest.
- Is weather irrelevant? Why?
- The process of removing details from a representation is called abstraction
- Choosing the level of abstraction
  - How many details do we want to consider?
  - Do we want to consider everything such as stopping in red light, looking out of the window?

The Vacuum world problem (state space graph)

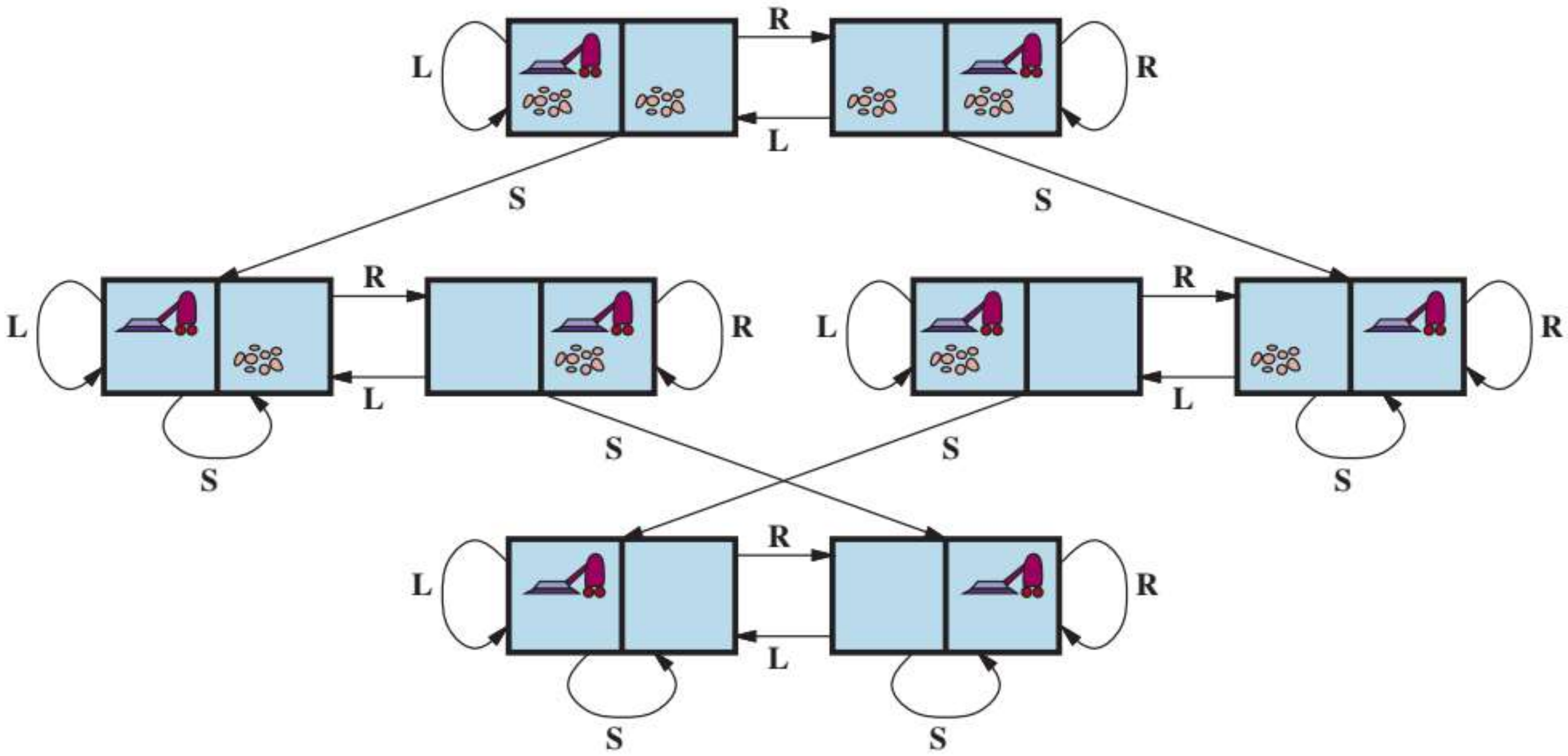
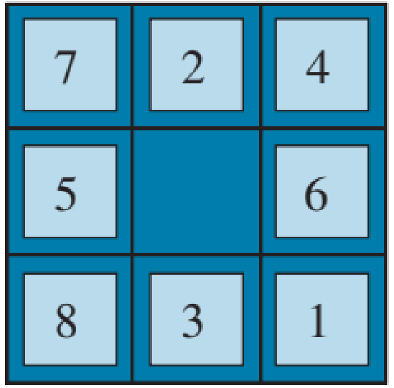


Figure 3.2 The state-space graph for the two-cell vacuum world. There are 8 states and three actions for each state: L = *Left*, R = *Right*, S = *Suck*.

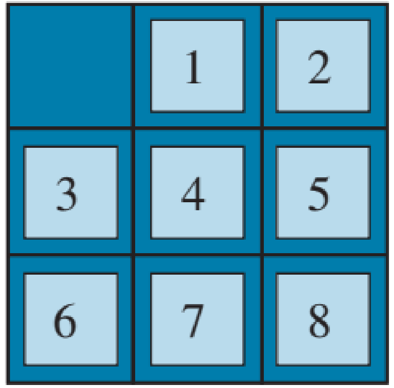
Formulating the vacuum world problem

- States: The state is determined by both the agent location and dirt locations
  - The agent is in one of the two locations, each of which might or might not contain dirt
  - Possible world states = $2 * 2^2 = 8$ ($n * 2^n$ is generalized form where n represent number of rooms)
- Initial State: Any state may be designated as initial state
- Actions: Three actions: left, right, and suck
- Transition model: the actions have expected effects, except that
  - Moving left in leftmost square
  - Moving right in rightmost square
  - Sucking in a clean square has no effect
- Goal test: check whether all the squares are clean
- Path cost: Each step cost 1, so the path cost is the number of step in the path

8-Puzzle (**sokoban puzzle**)



Start State



Goal State

- 3×3 grid with eight numbered tiles and one blank space.
- A tile adjacent to the blank space can slide into the space.
- The objective is to read a specified goal state
- 8 puzzle with picture block https://murhafsousli.github.io/8puzzle/#/

- States: A state description specifies the location of each of the tiles.
- Initial state: Any state can be designated as the initial state.
- Actions: While in the physical world it is a tile that slides, the simplest way of describing an action is to think of the blank space moving Left, Right, Up, or Down.
- Transition model: Maps a state and action to a resulting state;
- Goal state: a state with the numbers in order
- Action cost: Each action costs 1.

Real-world problems

- Route-finding problems are common in real world. Consider the airline travel problem that must be solved by a travel-planning website:
- **States**: Each state included a location (e.g. an airport) and the current time
- **Initial State**: Specified by the user's query
- **Actions**: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- **Transition model**: The state resulting from taking a flight will have the flight's destination as the new location and the flight's arrival time as the new time.
- **Goal state**: Are we at the destination city?
- **Action cost**: A combination of monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer reward points, and so on.

quiz

- What kind of environment is crossword puzzle? Justify your answer
- Draw a block diagram for a goal based agent.

Example real world problems (Assignment 1)

Each Group prepare a 2 page document detailing the problem using the 5 components learned in the lecture. Each component must be clear, easy to understand and should describe the purpose. A short presentation should be prepared by the group. (deadline 1^{st} Week of October 2022)

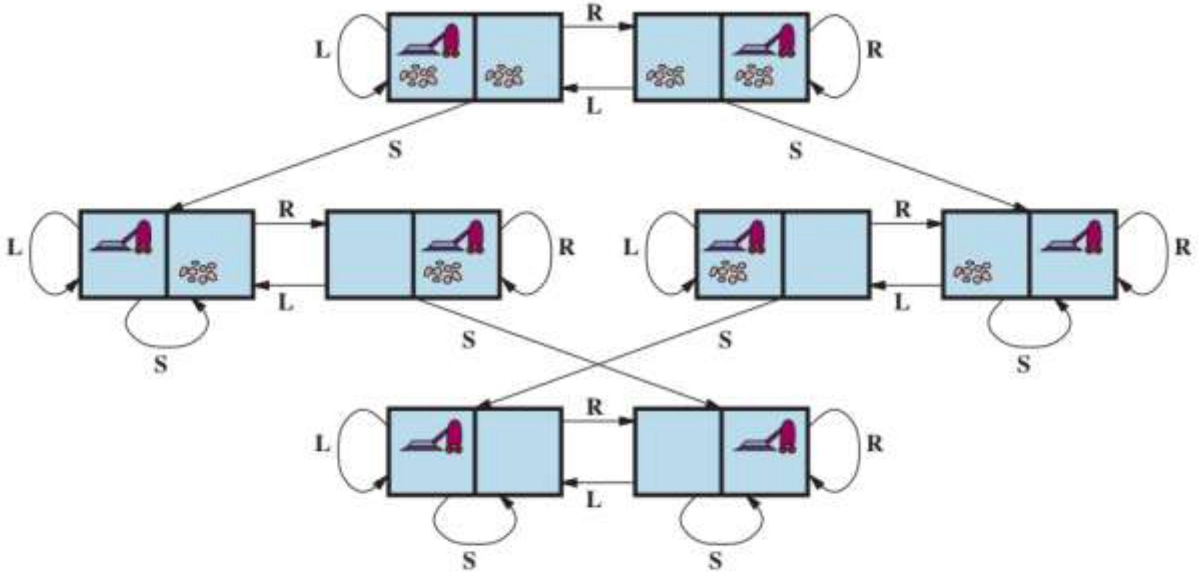
- **Touring problem (Group 1)**
- **Very Large Scale Integration (VLSI) layout problem (Group 2)**
- **Robot Navigation (Group 3)**
- **Automatic Assembly Sequencing (Group 4)**

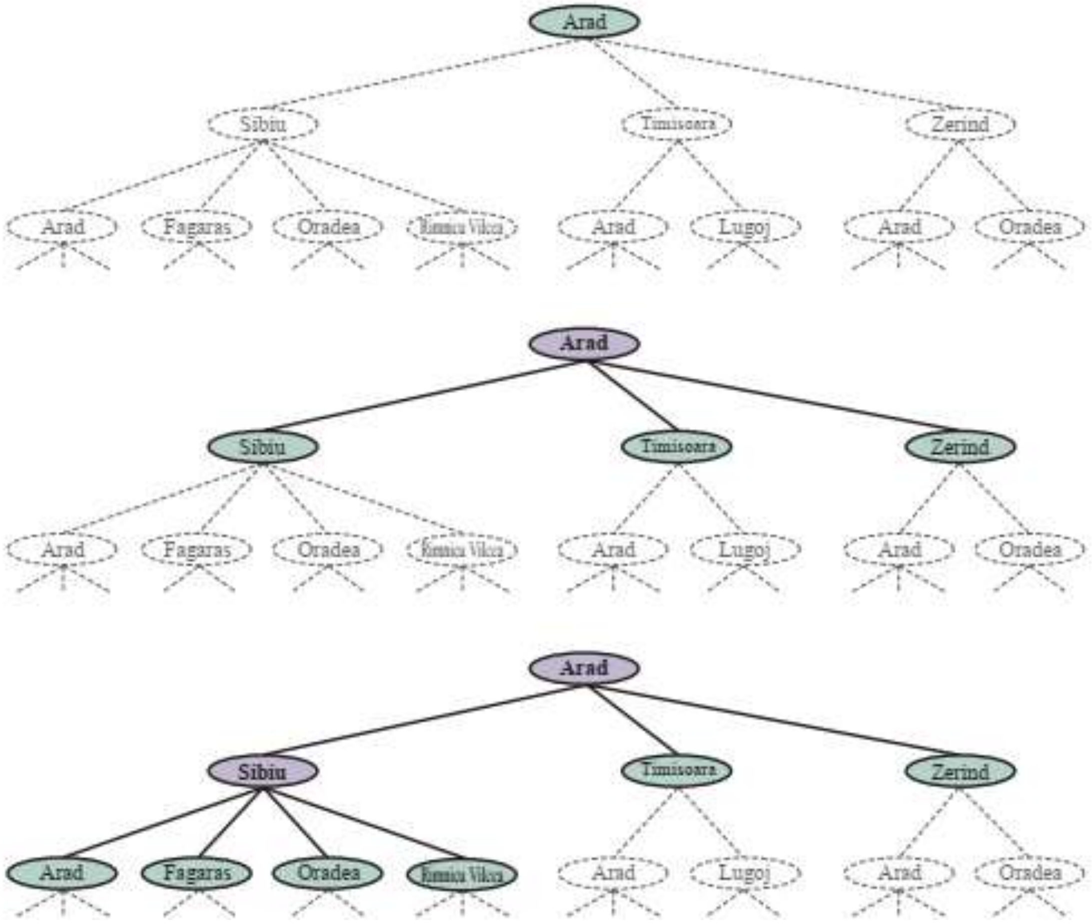
Concepts for searching algorithms

- State Space
- What Exactly is “Searching for Solution”?
- A general Tree-Search Algorithm
- Tree Search vs Graph Search
- Graph Search Algorithm (General Tree-Search)
- Nodes vs States
- The queue data structure
- Measuring Problem-solving performance

The Concept of state space

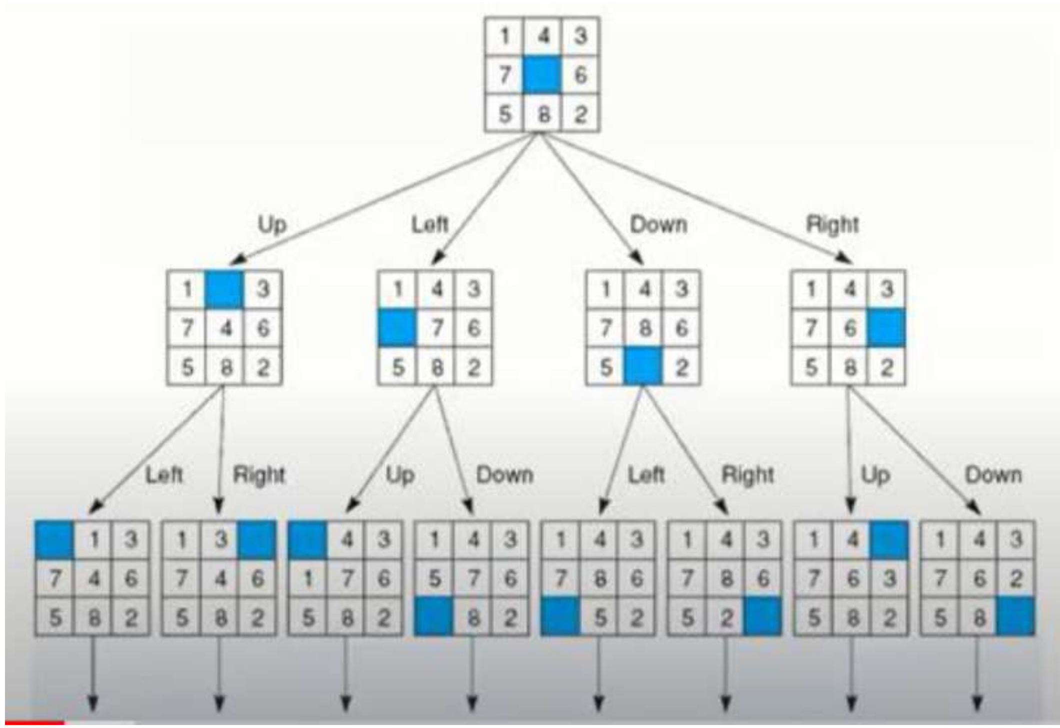
State Space = {“All possible Configuration}





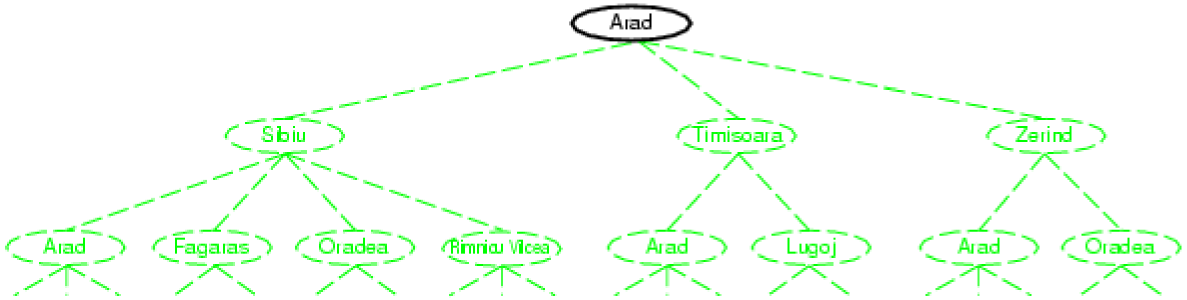
What exactly is 'Searching for Solution'?

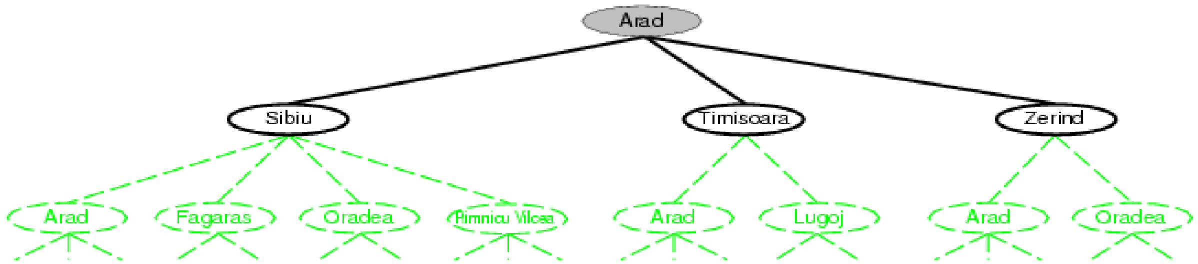
- A solution is an action sequence
  - Search algorithms work by considering various action sequences
- The possible action sequences, starting at the initial state form a **search tree** with
  - The initial state at the **root**
  - the **branches** are the actions and
  - The **nodes** correspond to states in the state space of the problem
- The essence of search:
  - Following up one option now and putting the others aside for later, in case the first choice does not lead to a solution



Example

- The root node of the tree corresponds to the initial state, In(Arad).
- The first step is to test whether this is a goal state
- Then we need to consider taking various actions
  - We do this by expanding the current state
  - i.e. applying each legal action to the current state, thereby generating a new set of states (Transition model)
- We add three branches from the parent node In (Arad) leading to three new child nodes
  - In (Sibiu)
  - In (Timisoara), and
  - In (Zerind)
- Now we must choose which of these three possibilities to consider further

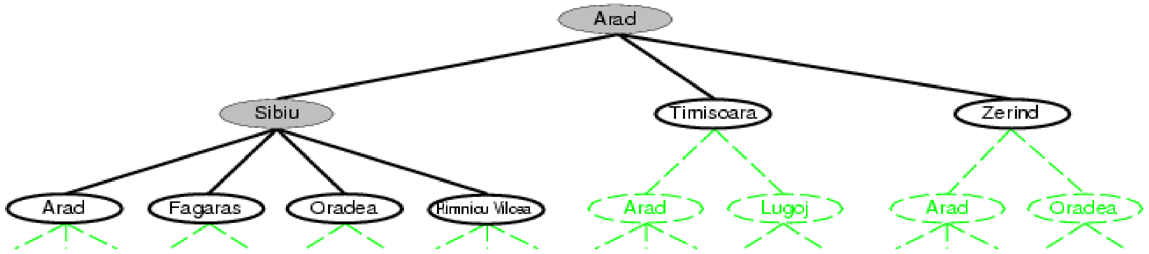




Cont…

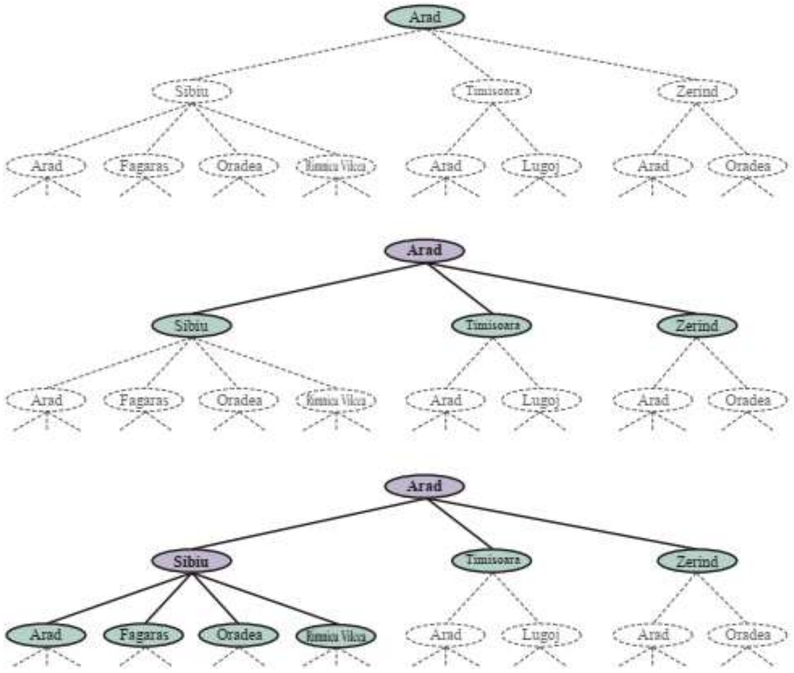
- Suppose we choose Sibiu first.
  - We check to see whether it is a goal sate (it is not) and then expand it to get In (Arad), In (Fagaras), In(Oradea), and In (RminicuVilcea)
  - We can then choose any of these four or go back and choose Timisoara or Zerind
- Each of these six node is a leaf node
  - A node with no children in the tree
- The set of all leaf nodes available for expansion at any given point is called the frontier
  - Tree consists of those nodes with bold outline

- What is the difference between a leaf node and frontier?



A general tree-search algorithm

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



Problem with the general tree search algorithm

- “algorithms that forget their history are doomed to repeat it”
- The way to avoid exploring redundant paths is to remember where one has been
- To achieve this, TREE-SEARCH is augmented with a data structure called **explored set** (which remembers every expanded node)
- Newly generate node, that match previously generated nodes-ones in the explored set or the frontier-can be discarded instead of being added to the frontier

Graph Search Algorithm

```
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
```

Data structures to keep track of the search tree

- Search algorithms require a data structure to keep track of the search tree that is being constructed
- 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

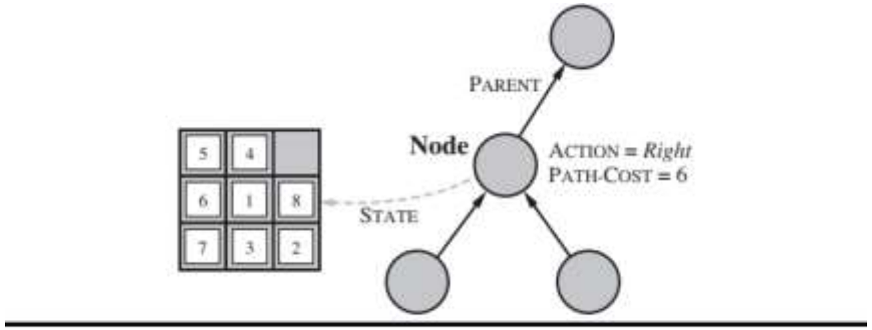


Figure 3.10 Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

Nodes vs states

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*

- *Nodes are on particular paths, as defined by parent pointers whereas states are not*

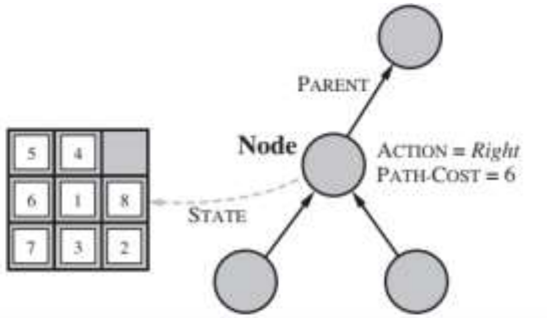


Figure 3.10 Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

The queue data structure

- The operations on a queue data structure are as follows:
  - EMPTY?(queue) returns true only if there are no more elements in the queue.
  - POP(queue) removes the first element of the queue and returns it.
  - INSERT(element, queue) inserts an element and returns the resulting queue.
- Queues are characterized by the *order* in which they store the inserted nodes.
- Three common variants are:
  - first-in, first-out or **FIFO queue**, which pops the *oldest* element of the queue;
  - last-in, first-out or **LIFO queue** (also known as a **stack**), which pops the *newest* element
  - **priority queue**, which pops the element of the queue with the highest priority according to some ordering function

Measuring problem solving performance

- Four ways to measure algorithm's performance
  - 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?
  - Time and Space complexity is measured using the size of the state space graph i.e.
    - |V | + |E|, where V is the set of vertices (nodes) of the graph and E is the set of edges (links)

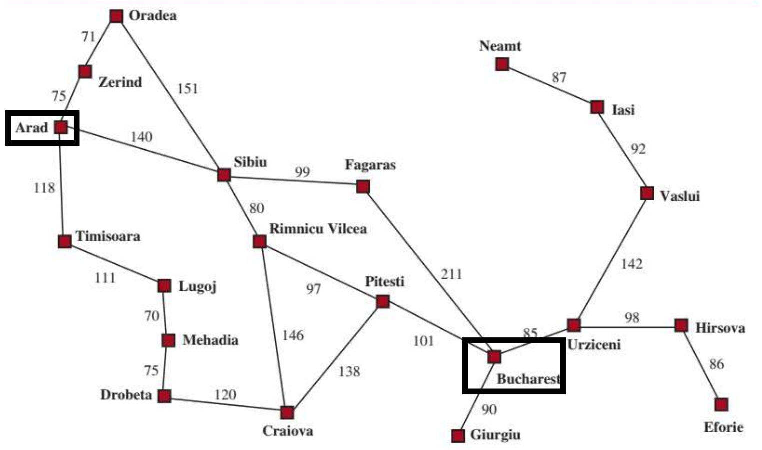


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.