Lecture 6
Intro to search (solving problems by search)

CS 180 – Intelligent Systems

Dr. Victor Chen

Spring 2021

Solving problems by search

Review: PEAS model: on which most of AI agents work

PEAS: Performance measure, Environment states, Actuators, Sensors

P: a function the agent is maximizing (or minimizing)

E: *A state = a group of* **variables**

A: actions that the agent takes to move from one state to another according to a *transition model*

S: observations/sensors that allow the agent to infer the state

Maze problems

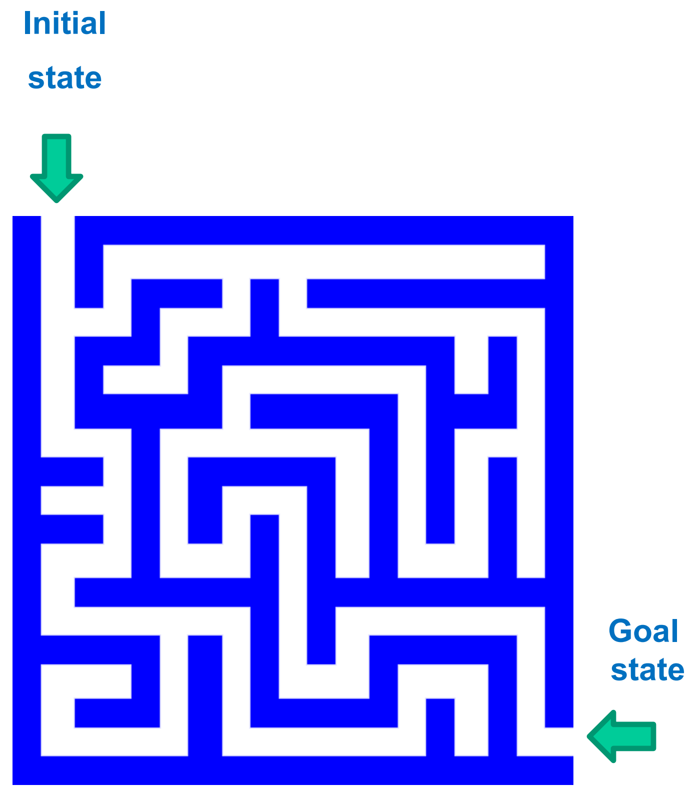
Initial state

Actions

Goal state

Objective function

- Shortest path (a sum of *step costs)* from the initial state to a goal state



Example: Romania Traveling

- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest
- The **optimal solution** is <u>the shortest path from Arad to Bucharest</u>

Initial state

- Arad

Actions

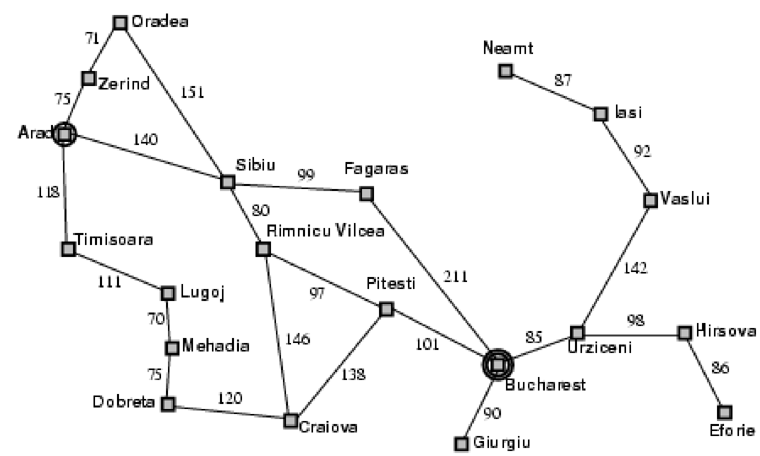
- Go from one city to another

Goal state

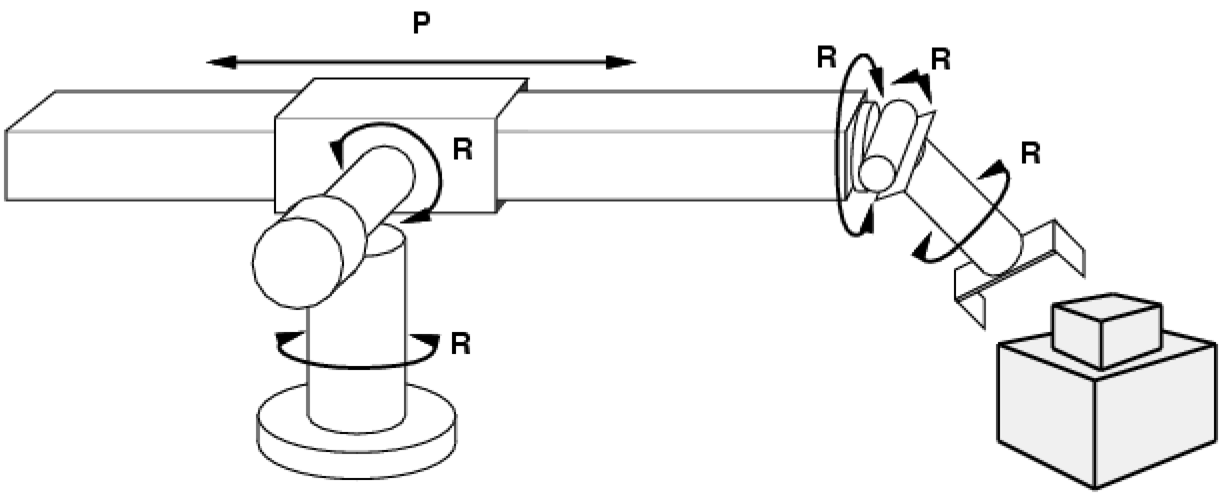
- Bucharest

Objective function

- Minimax the sum of edge costs (total distance traveled)



Example: Robot motion planning



States

- Real-valued robot joint parameters (angles, displacements)

Actions

- Continuous motions of robot joints

Goal state

- Configuration in which object is grasped

Path cost

- Time to execute, smoothness of path, etc.

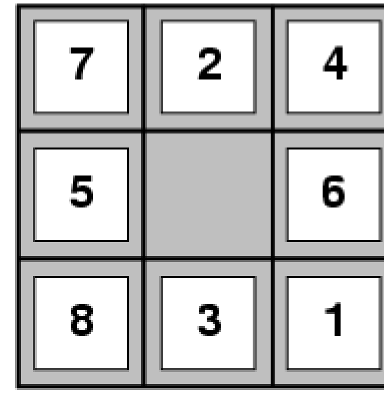
State space

Given an AI problem, state space refers to all the states reachable from initial state

How large the **state space** can be?

Example: The 8-puzzle

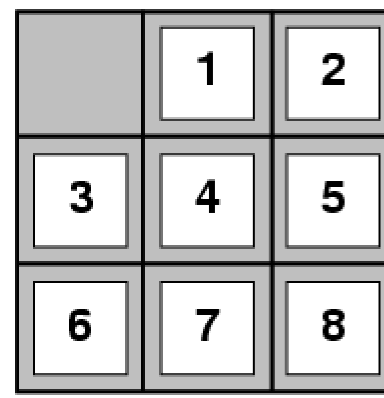
- 3x3 board with 8 tiles. Each tile represents a number
- A tile can only slide into the blank space .



Start State

States

- Locations of tiles
  - 8-puzzle: 181,440 states (9!/2)
  - 15-puzzle: ~10 trillion states
  - 24-puzzle: ~10^{25} states



Goal State

Actions

- Move blank space left, right, up, down

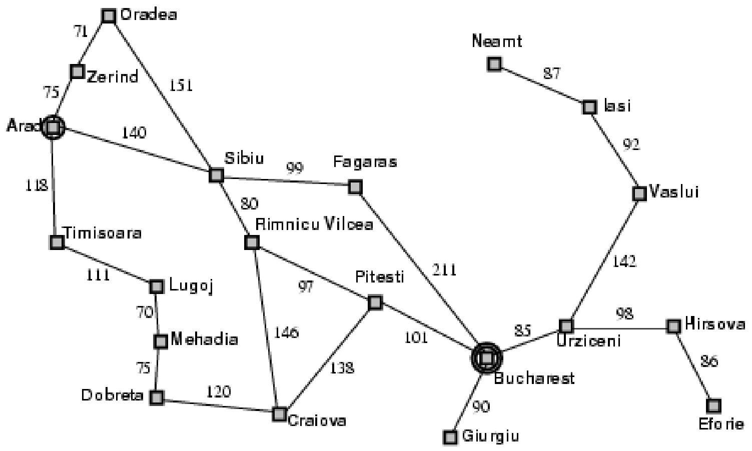
Path cost

- Each step costs 1.

What is Search?

Given:

- **Initial state**
- **Goal state**
- **Actions**
- **Transition model**
- **Path cost function**



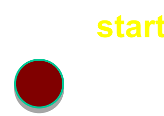
find <u>a sequence of actions that minimize the cost</u>

- Can we use Dijkstra's shortest path algorithm?
  - Cost of Dijkstra's is $O(E + V \log V)$, where V is the size of the state space
  - Unaffordable cost because V may be huge!

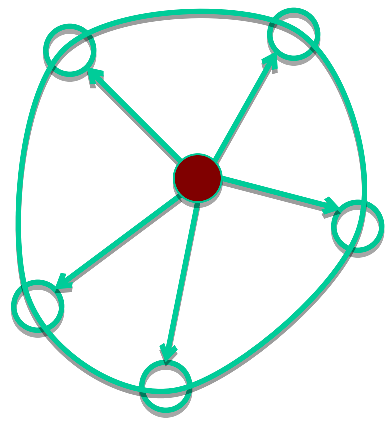
Search: Basic idea

- Begin at the initial state and **expand** it to all possible successor states
- Maintain a **frontier** (a list of unexpanded states)
- At each step, pick a state from the frontier to expand
- Keep going until you reach a goal state
- Goal: Try to <u>expand as few states as possible</u>

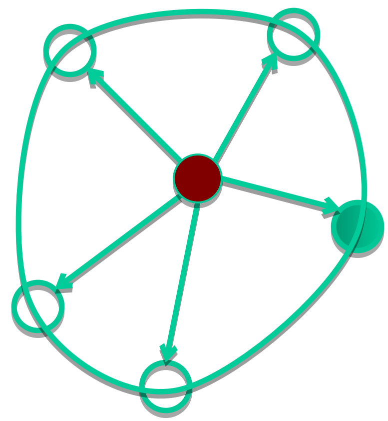
Search: Basic idea



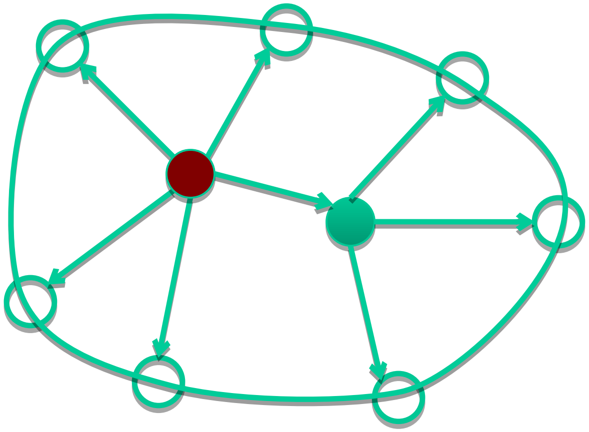
Search: Basic idea



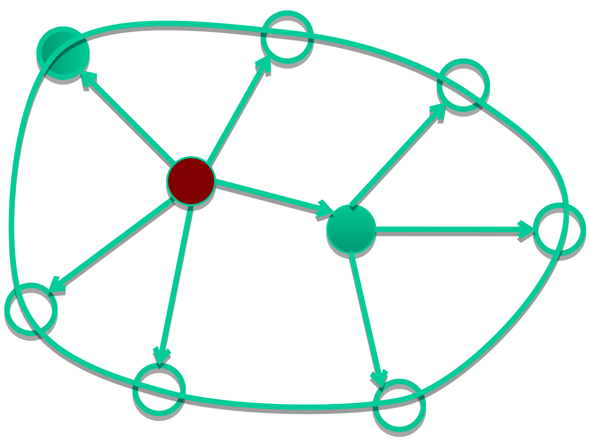
Search: Basic idea



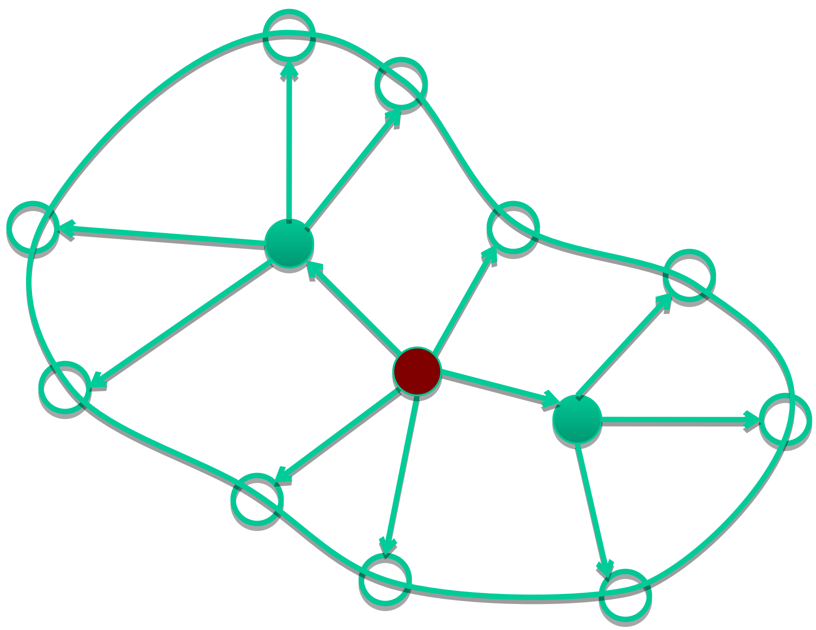
Search: Basic idea



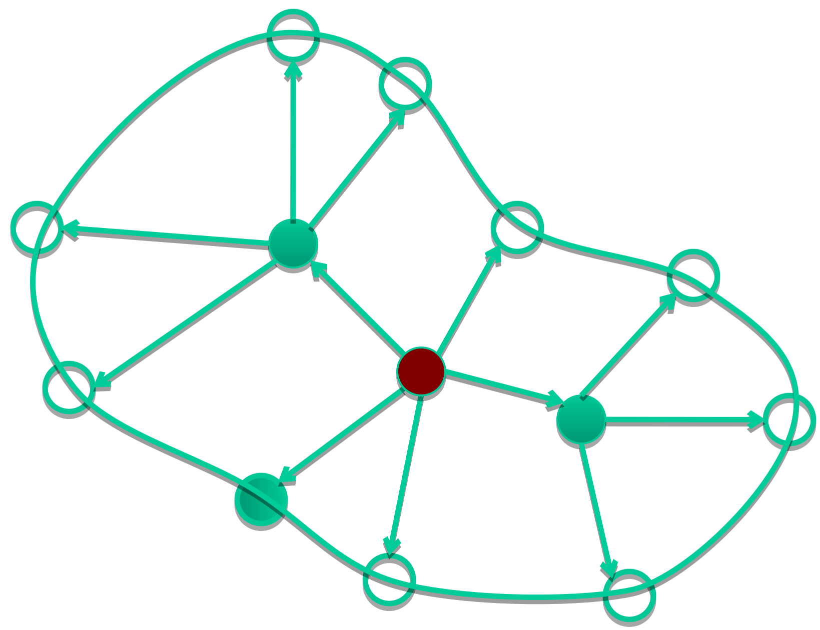
Search: Basic idea



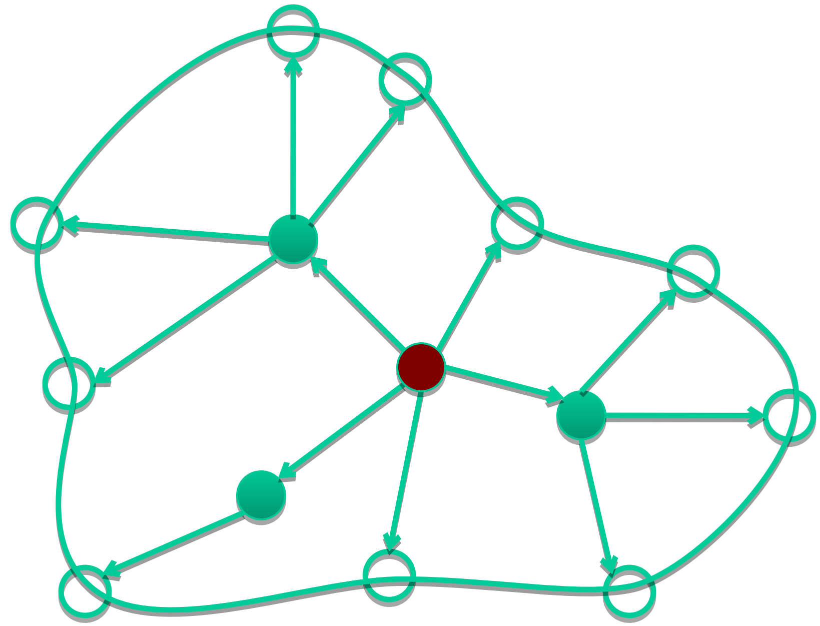
Search: Basic idea



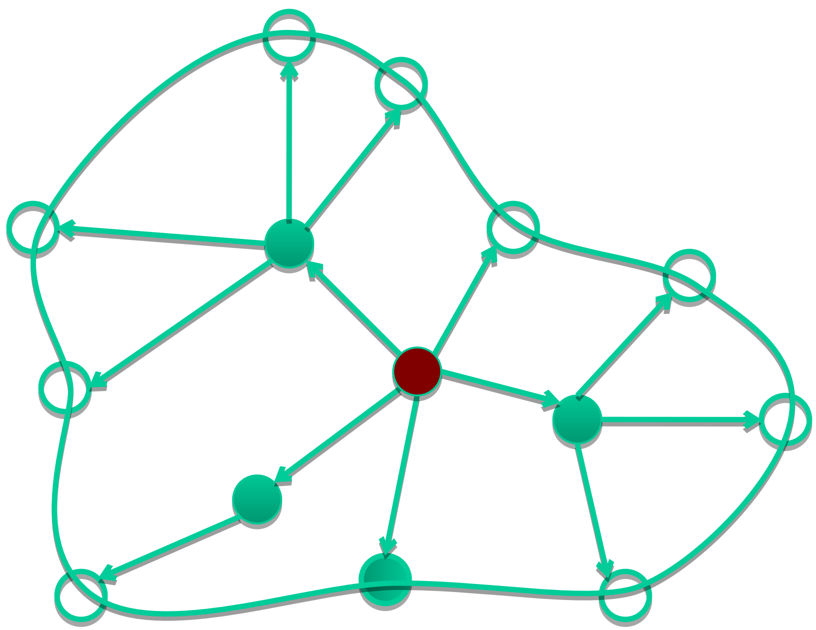
Search: Basic idea



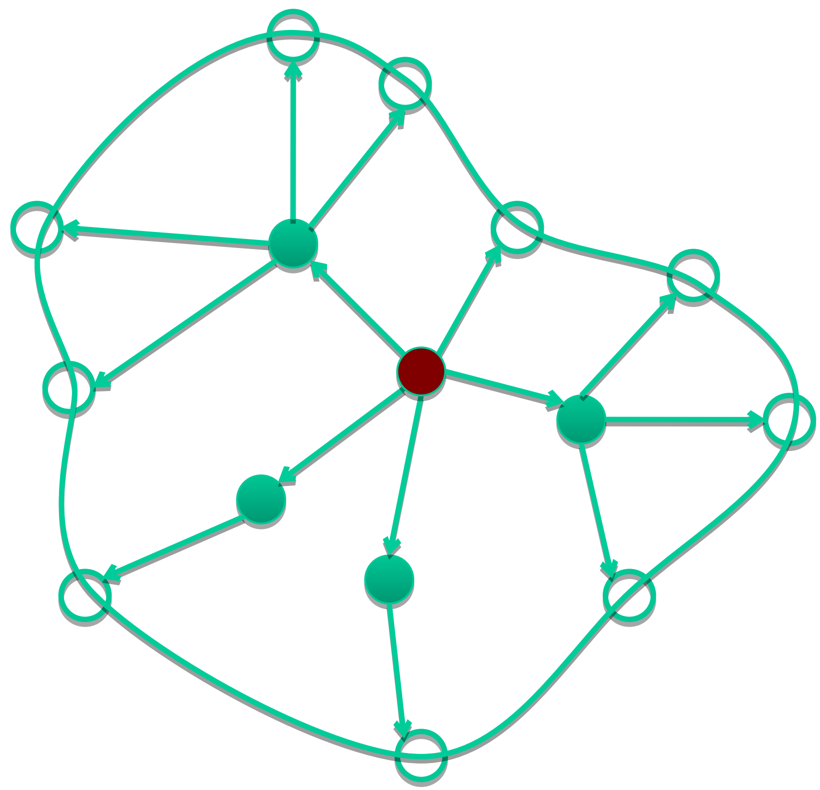
Search: Basic idea



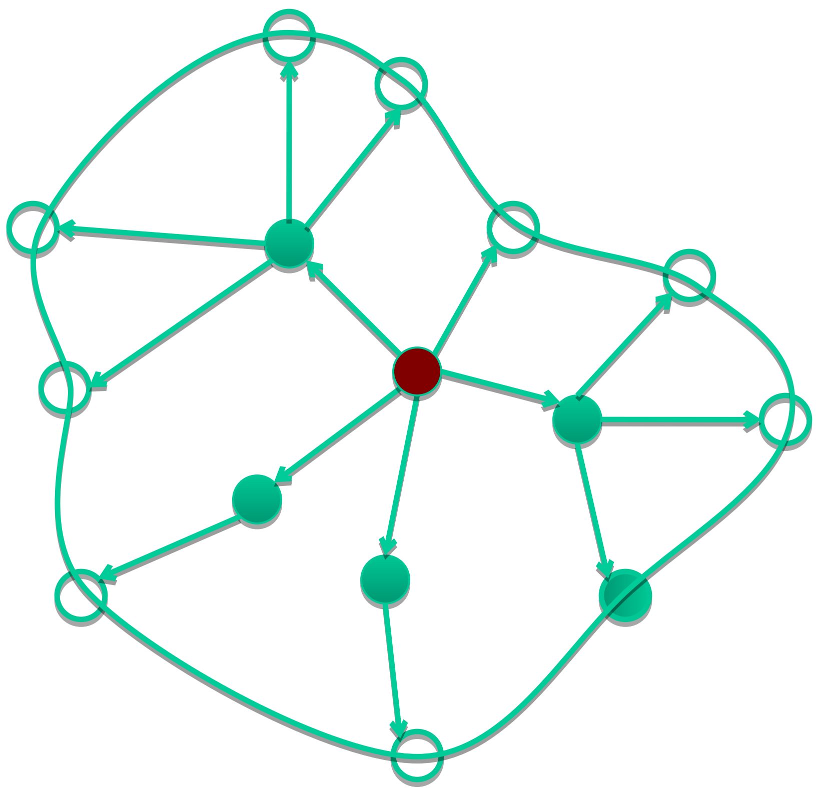
Search: Basic idea



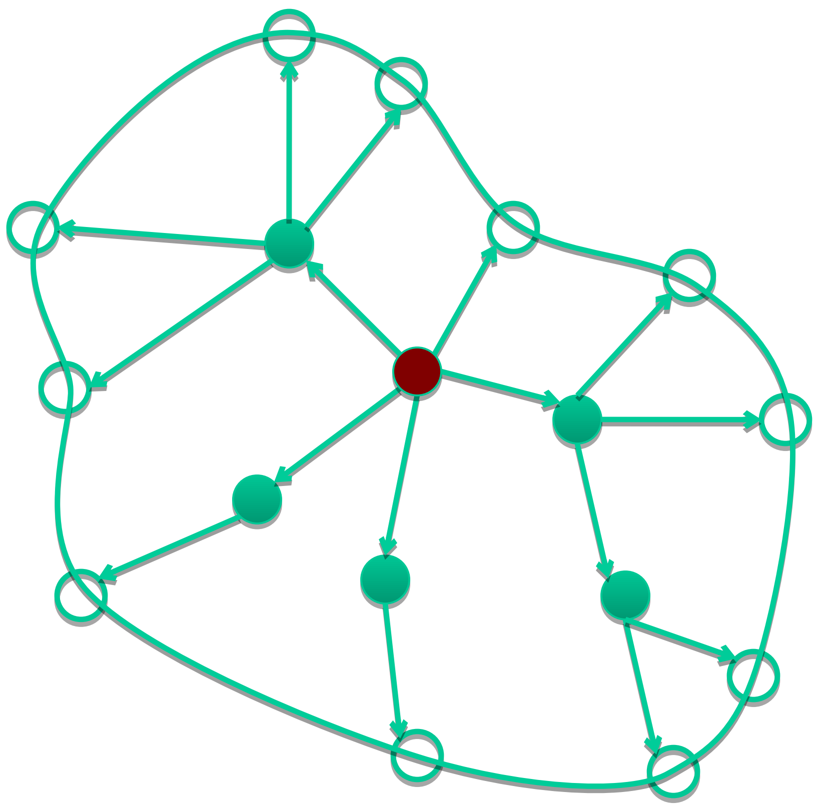
Search: Basic idea



Search: Basic idea



Search: Basic idea



Search: Basic idea

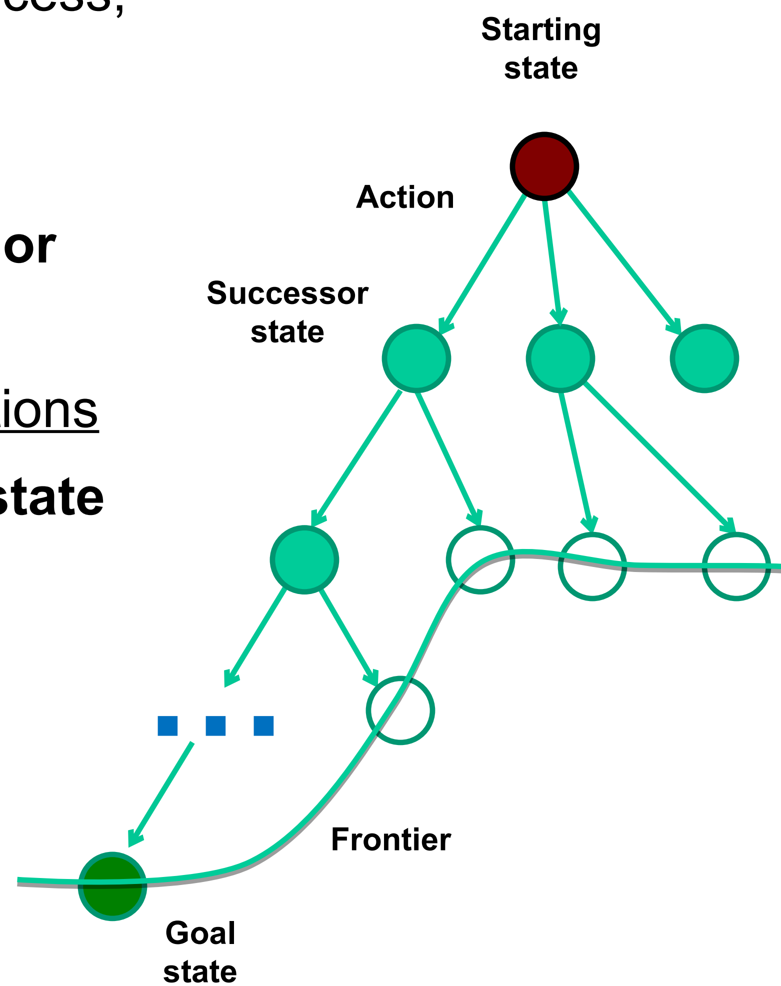
All search algorithms share the basic idea above; they vary according to which state we will pick from the frontier to expand next (the so-called **search strategy**)

Pseudocode of All Search Algorithm

1. Initialize the **frontier** using the **initial state**
2. While the frontier != empty
   - Choose a frontier node according to **search strategy** and take it off the frontier
   - If the node is **goal state**, return solution
   - Else **expand the node** and add its successor states to the frontier

Visualization by Search tree

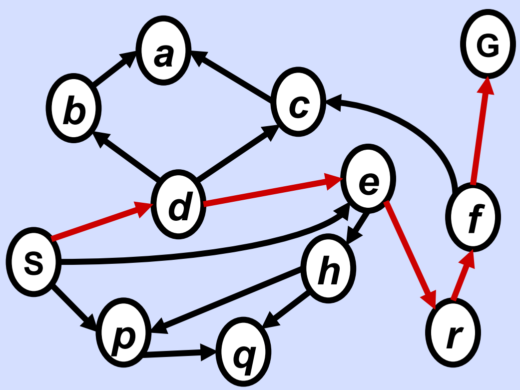
- We can build a search tree for whole process, where each nodes represents a state
- The root node is the initial state
- The **children** of a node are the **successor states** of that node
- A path corresponds to a sequence of actions
- **Goal: Find a path ending in the goal state**



Two ways to show a search process:

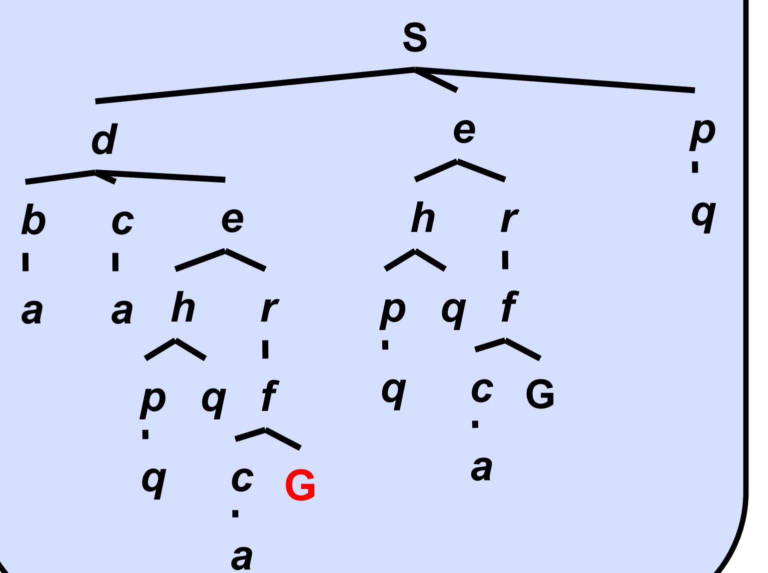
State Space Graphs vs. Search Trees

State Space Graph

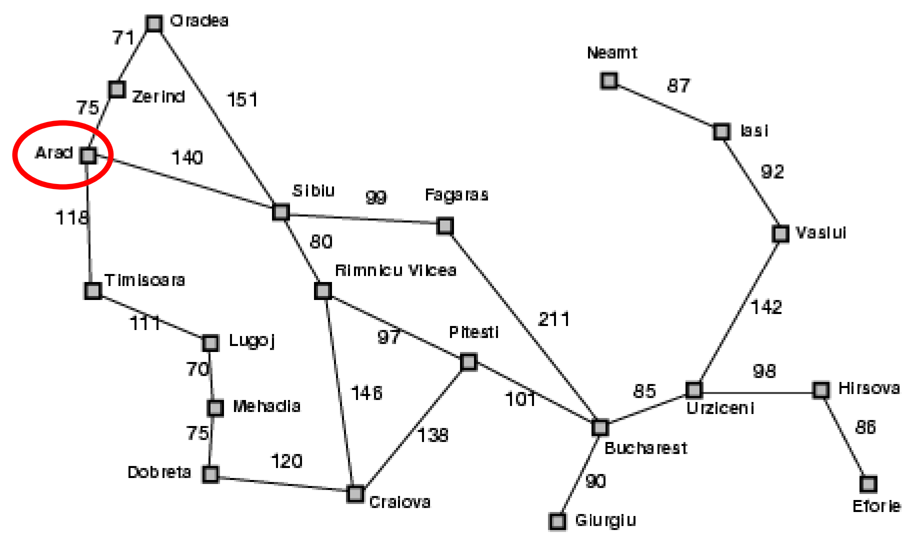


Each NODE in search tree is a PATH in state space graph.

Search Tree



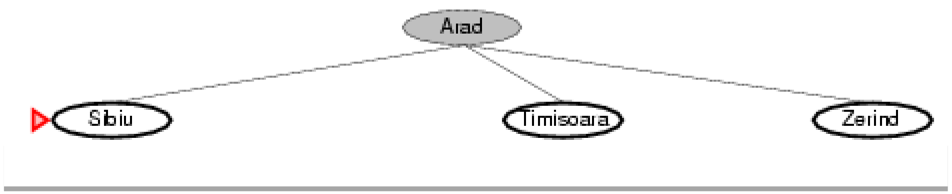
Tree search example (Romania)

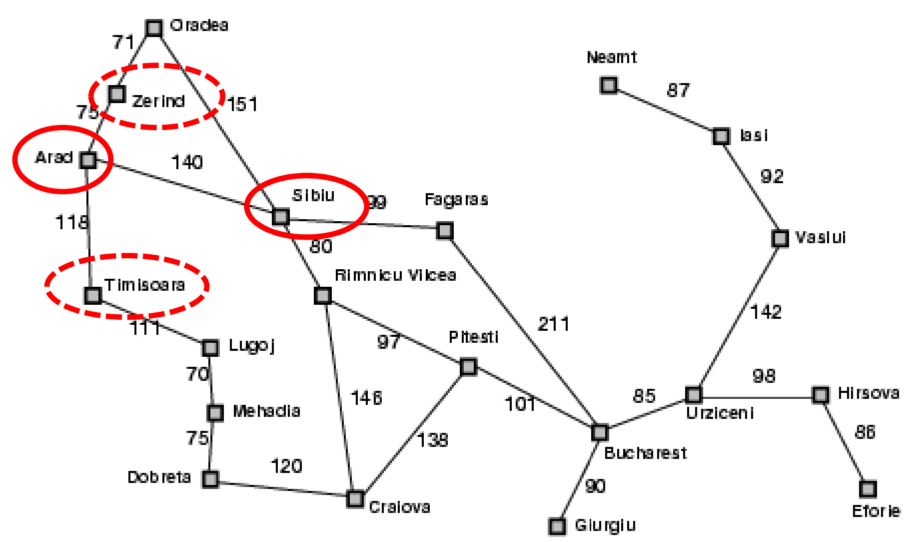


Start: Arad

Goal: Bucharest

Tree search example

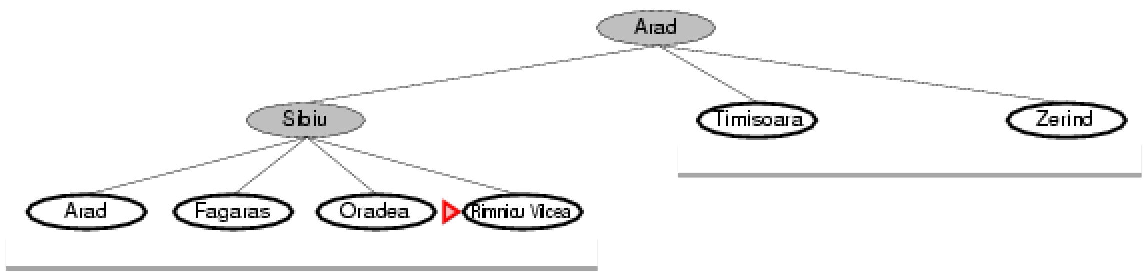


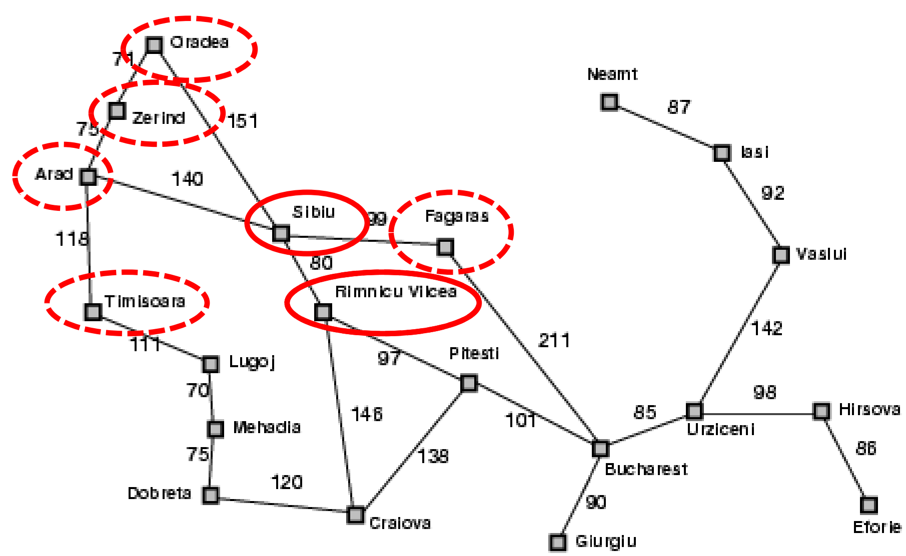


Start: Arad

Goal: Bucharest

Tree search example

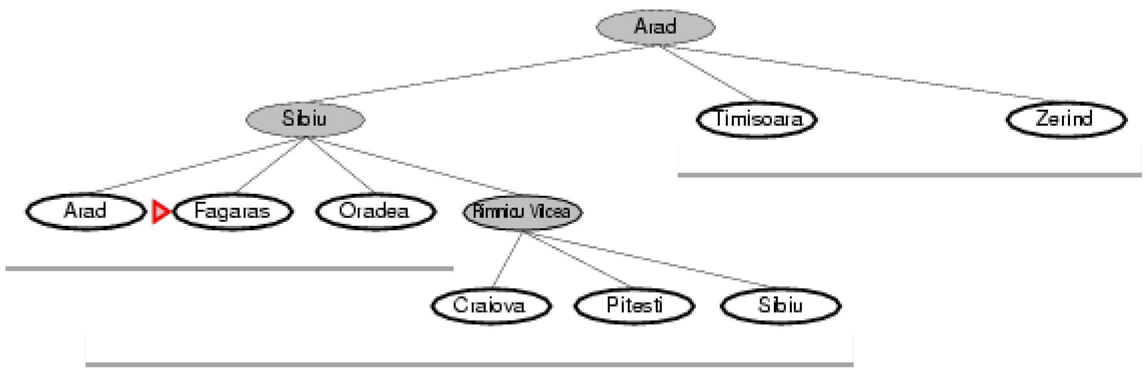




Start: Arad

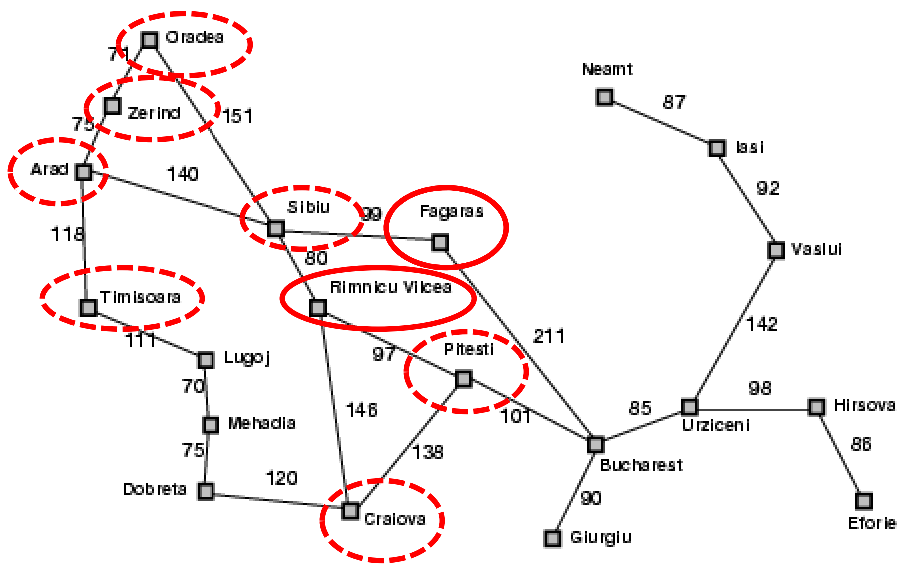
Goal: Bucharest

Tree search example

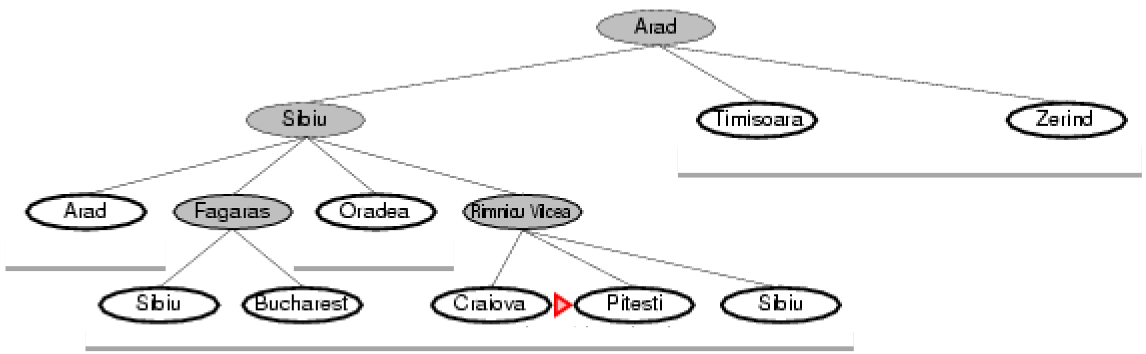


Start: Arad

Goal: Bucharest

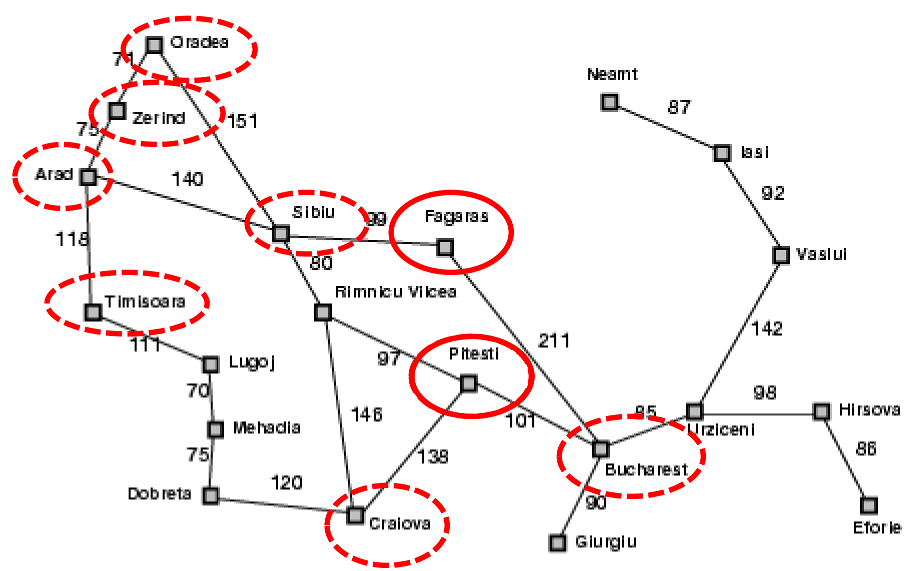


Tree search example

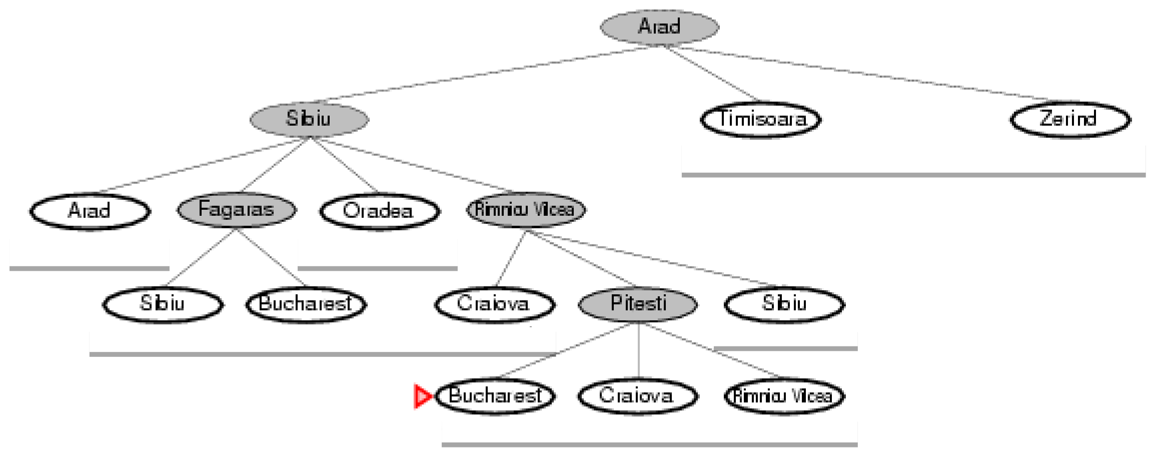


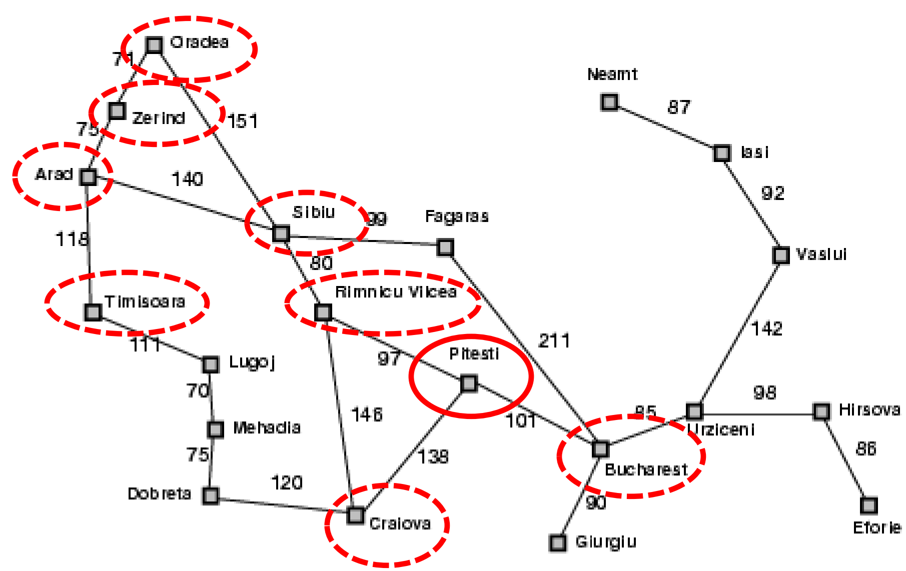
Start: Arad

Goal: Bucharest



Tree search example

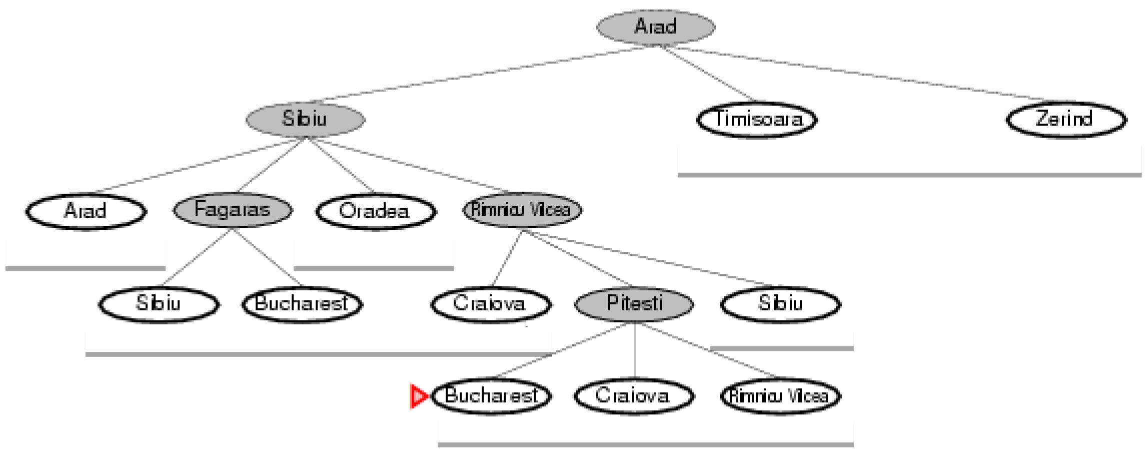




Start: Arad

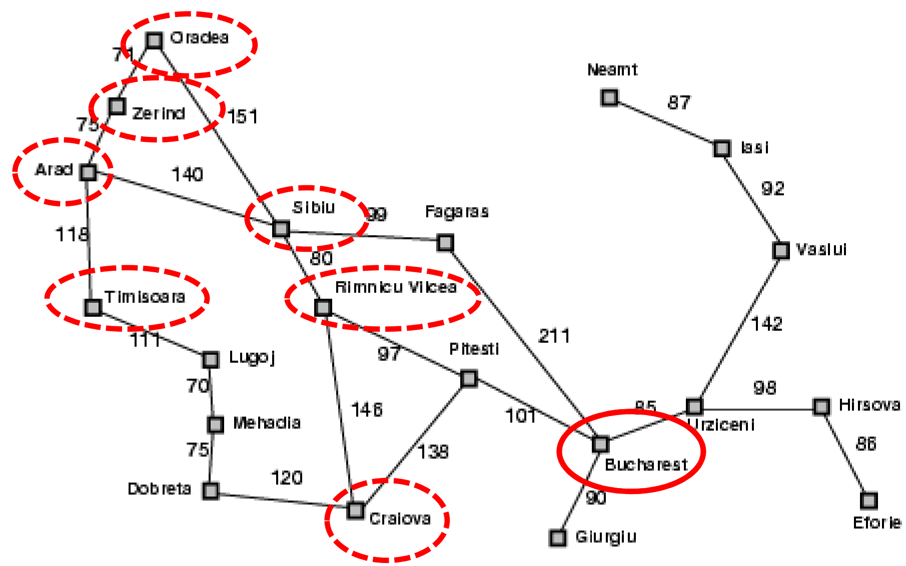
Goal: Bucharest

Tree search example



Start: Arad

Goal: Bucharest



Improve the search above by handling repeated states

In the example above, when we **expand the node,** we add all its successor states to the frontier

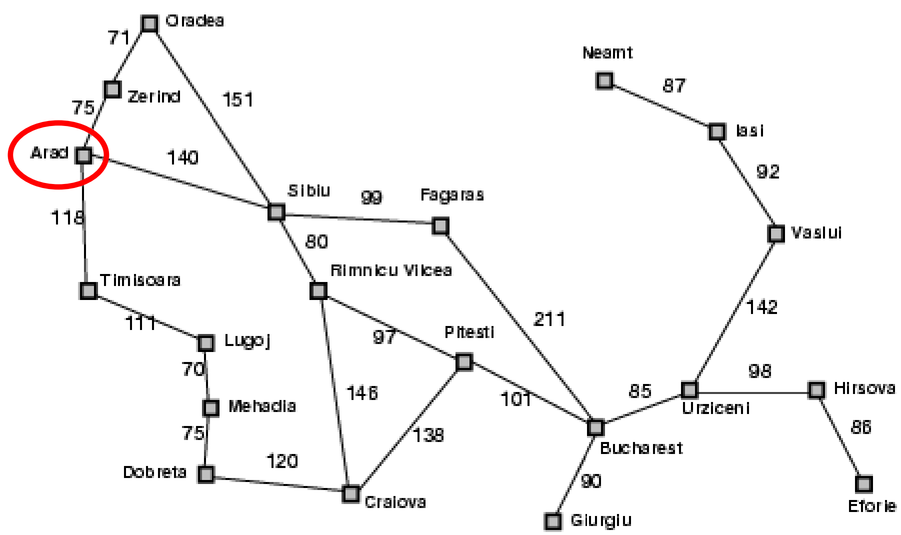
In order to eliminate repeated states:

- Every time you expand a node, add that node to a **explored set**; do not add any explored states to the frontier again
- Every time you add a node to the frontier, check if it already exists in the frontier with a higher path cost, and if yes, replace that node with the new one

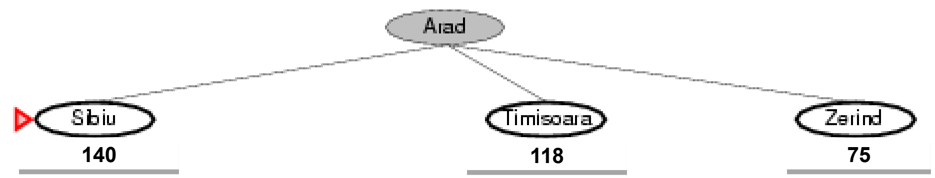
Search without repeated states

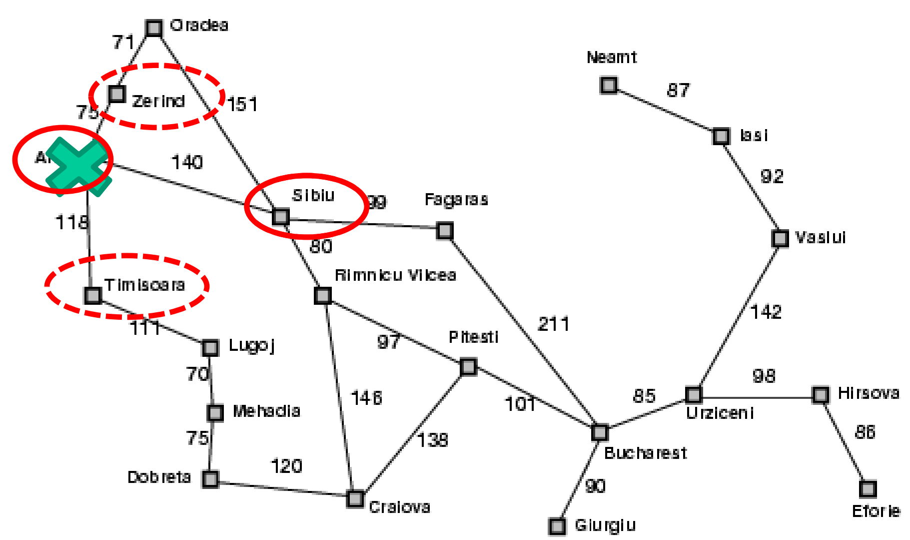
Start: Arad

Goal: Bucharest



Search without repeated states

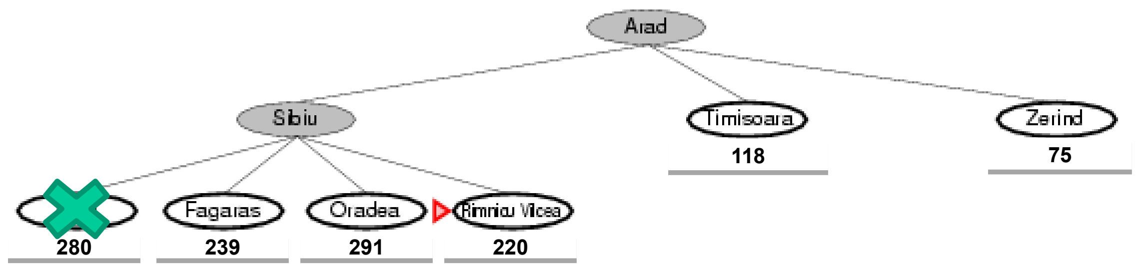


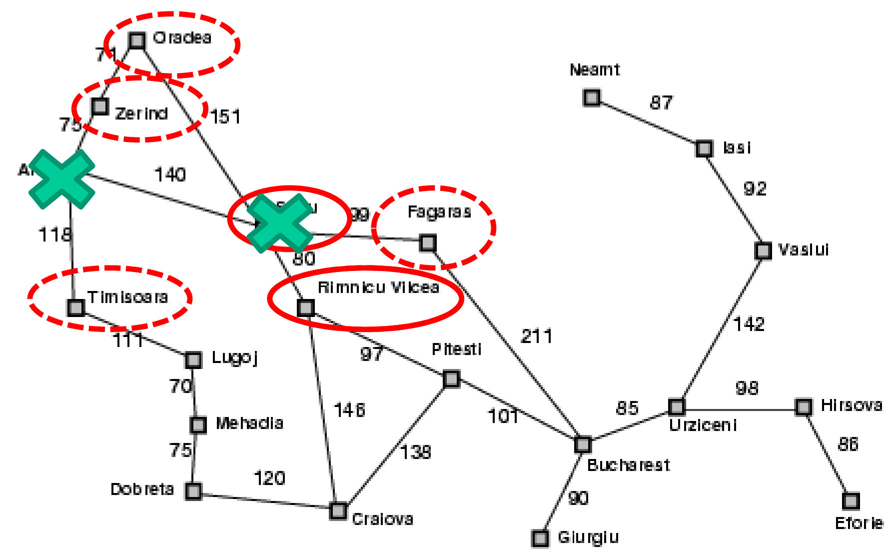


Start: Arad

Goal: Bucharest

Search without repeated states

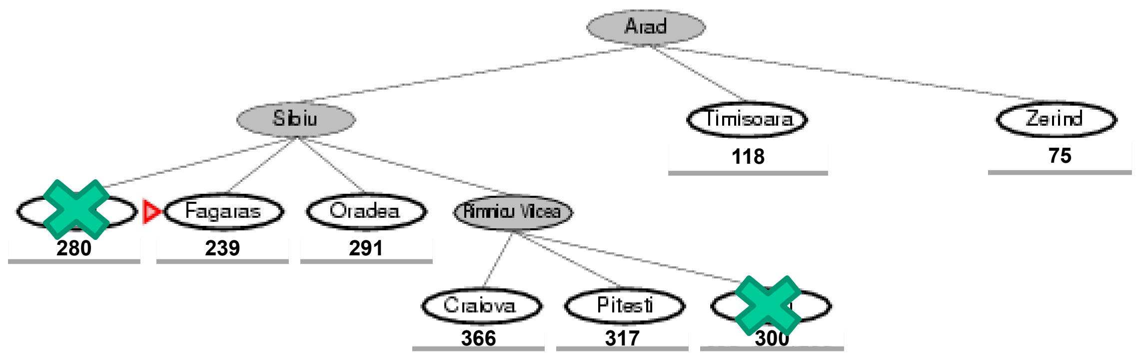




Start: Arad

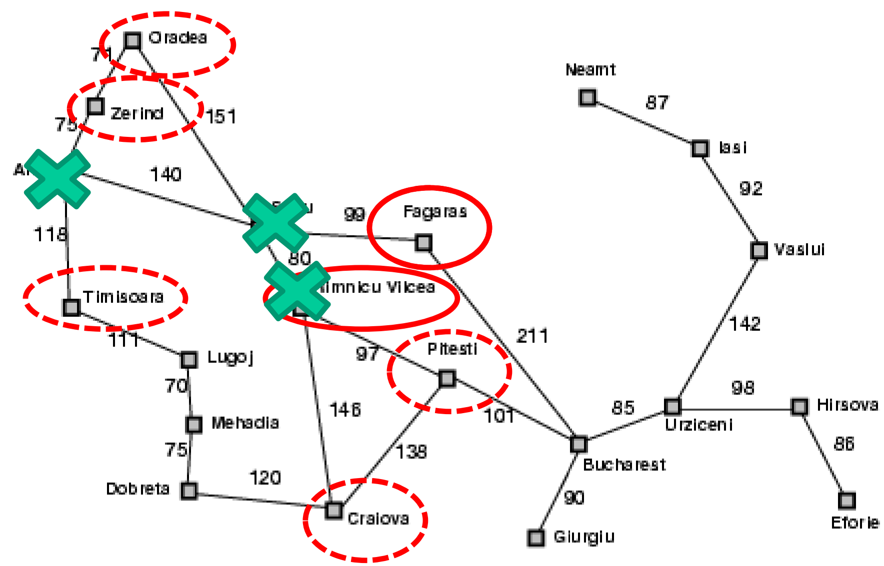
Goal: Bucharest

Search without repeated states

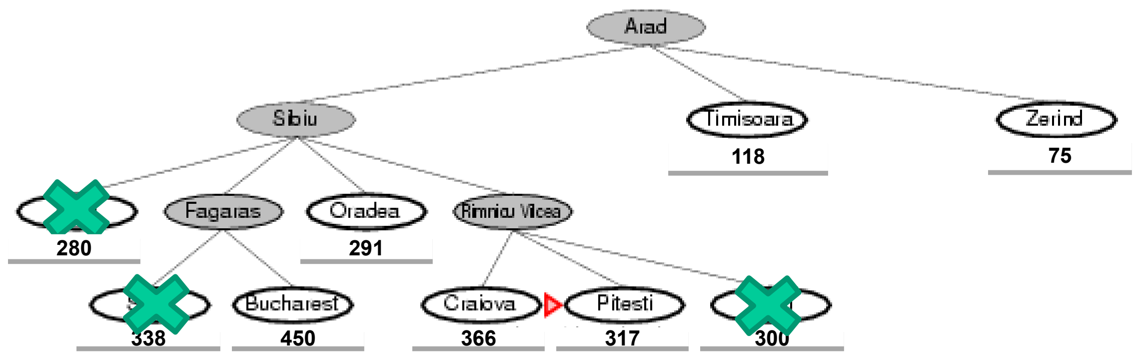


Start: Arad

Goal: Bucharest

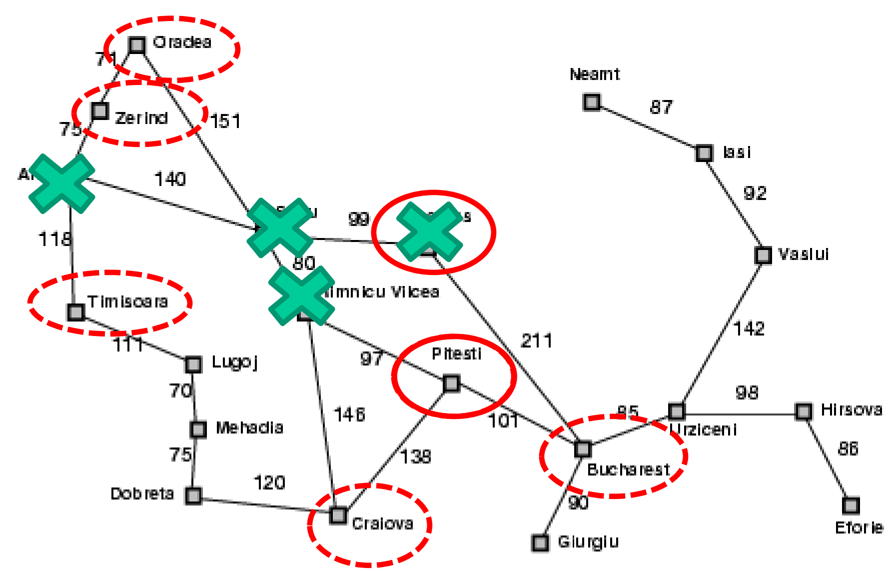


Search without repeated states

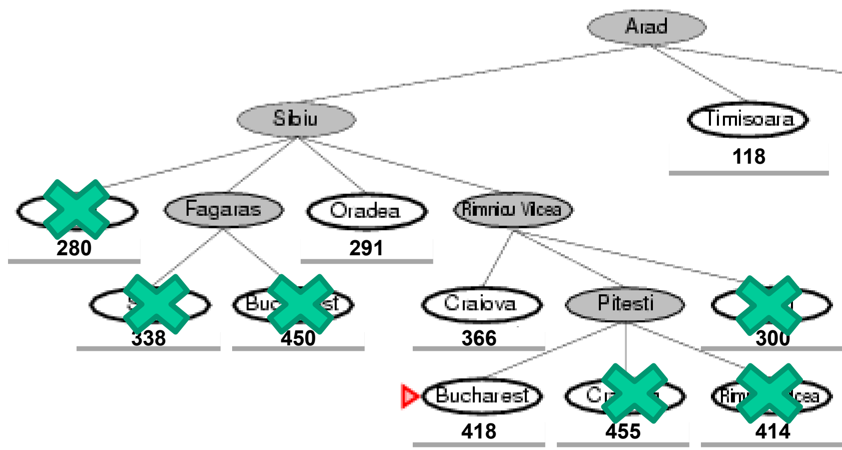


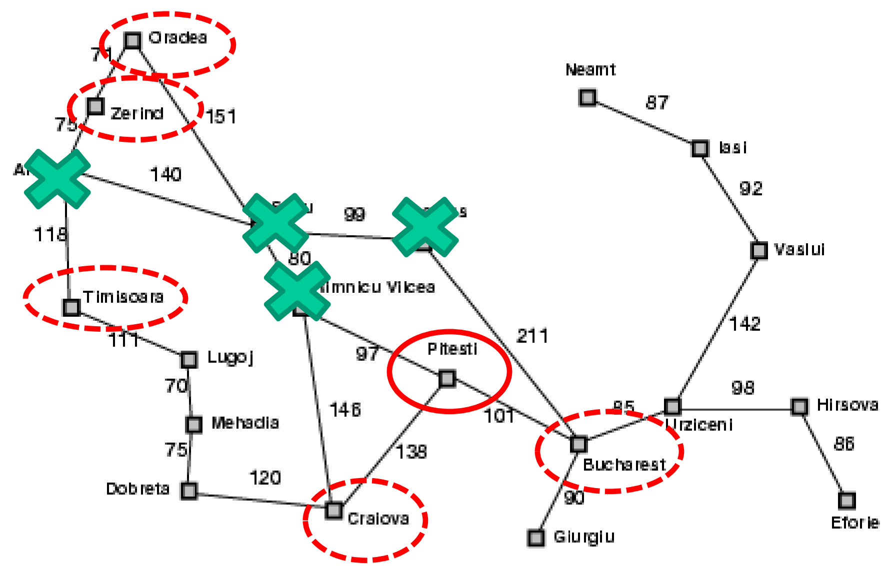
Start: Arad

Goal: Bucharest



Search without repeated states

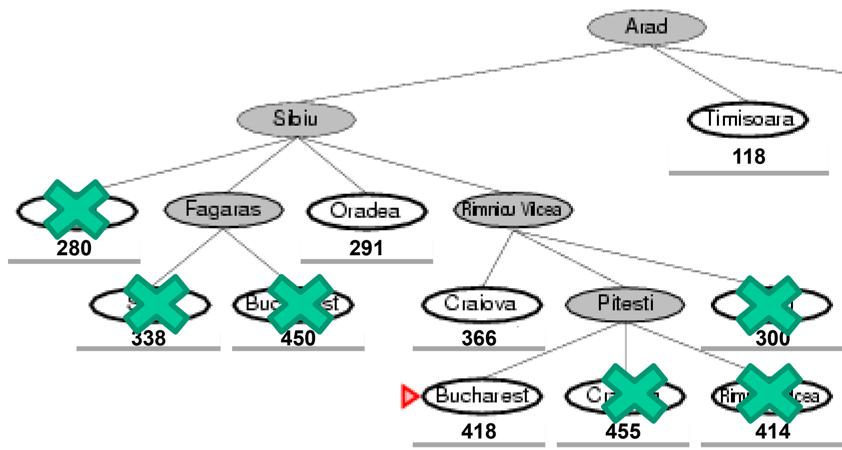




Start: Arad

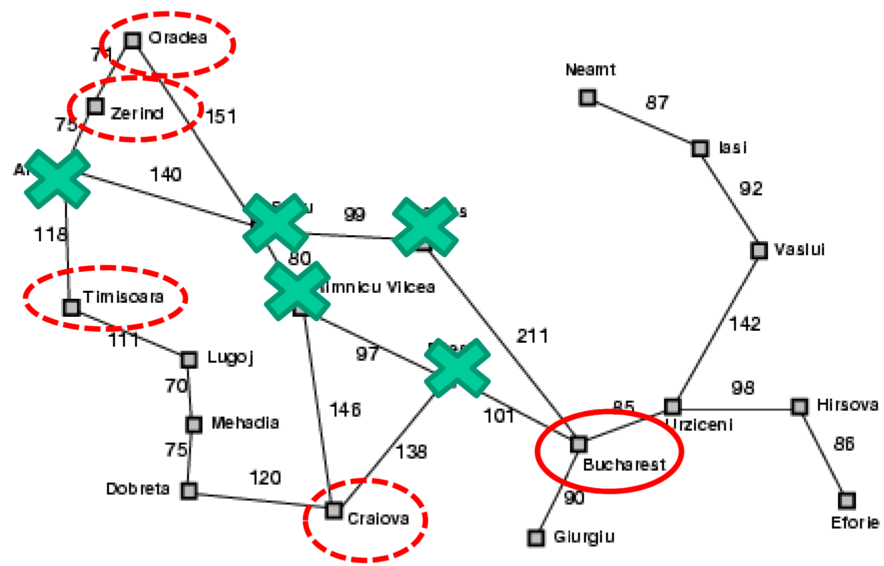
Goal: Bucharest

Search without repeated states



Start: Arad

Goal: Bucharest



Analysis of search strategies

All the search algorithms we will learn follow the idea above.

In the following two lectures, we will examine search performance using the following criteria:

- **Completeness:** does it always find a solution if one exists?
- **Optimality:** does it always find a least-cost solution?
- **Time complexity:** maximum number of nodes expanded
- **Space complexity:** maximum number of nodes kept in memory

Time and space cost are measured in terms of

- ***b**:* maximum branching factor of the search tree
- ***d**:* depth of the optimal solution
- ***m***: the depth of the entire search tree