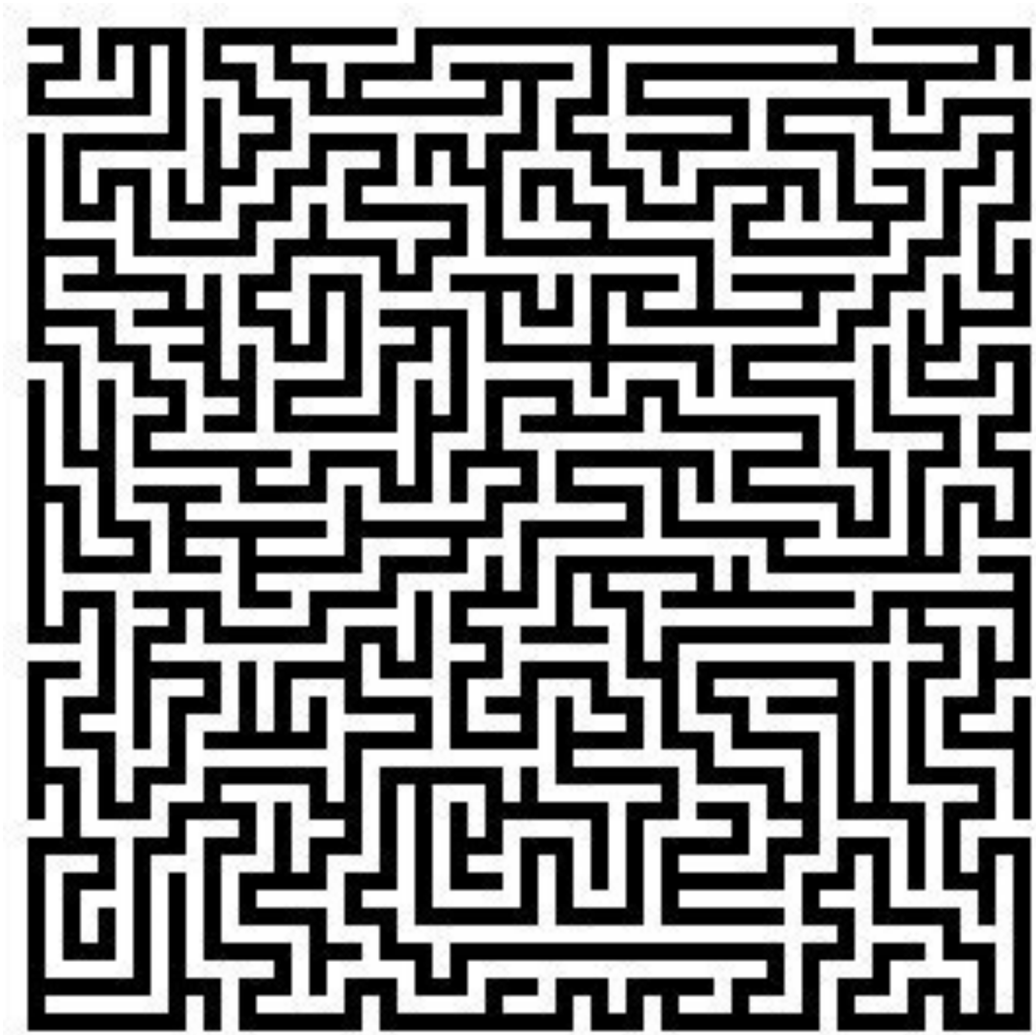
Solving problems by searching



Sanja Lazarova-Molnar, PhD

Search as Problem-Solving Strategy

- many problems: reach goal state from a given starting point

Examples

- getting from home to SDU
  - start: home
  - goal: SDU
  - operators: move one block, turn
- loading a moving truck
  - start: apartment full of boxes and furniture
  - goal: empty apartment, all boxes and furniture in the truck
  - operators: select item, carry item from apartment to truck, load item
- getting settled
  - start: items randomly distributed over the place
  - goal: satisfactory arrangement of items
  - operators: select item, move item

Motivation

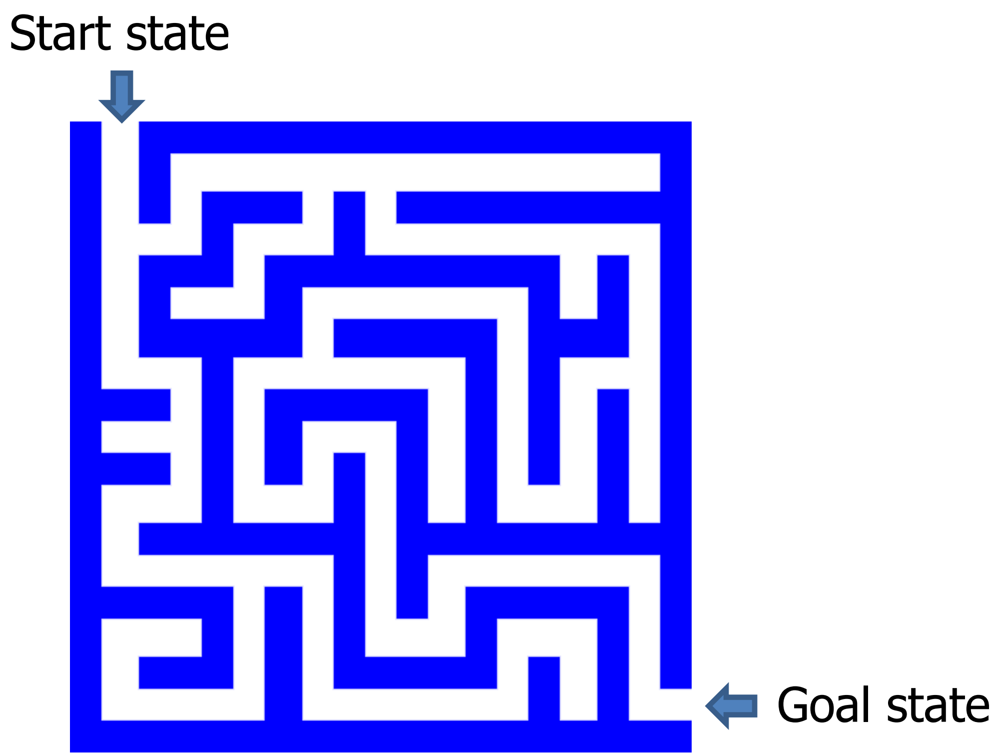
- search strategies - important methods for many approaches to problem-solving
- to use search - abstract formulation of the problem and the available steps
- search algorithms - basis for many optimization and planning methods

Objectives

- formulate appropriate problems as search tasks
  - states, initial state, goal state, successor functions (operators), cost
- know the fundamental search strategies and algorithms
  - uninformed search
    - breadth-first, depth-first, uniform-cost, iterative deepening
  - informed search
    - best-first (greedy, A*), heuristics
- evaluate the suitability of a search strategy for a problem
  - completeness, optimality

Search

- We will consider the problem of designing **goal-based agents** in **observable, deterministic, discrete, known** environments
- Example:

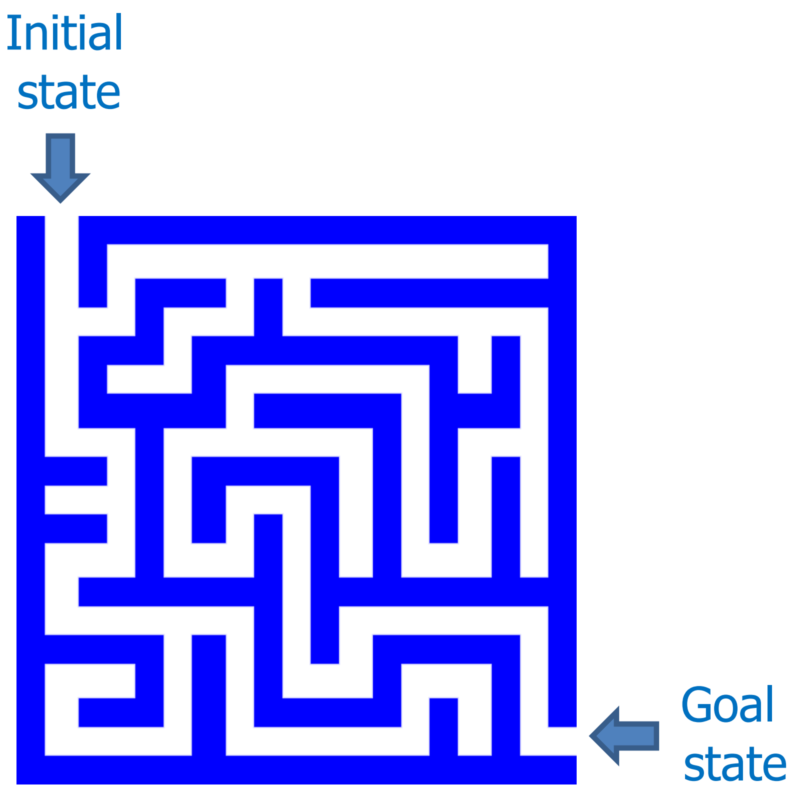


Search

- We will consider the problem of designing **goal-based agents** in **observable**, **deterministic**, **discrete**, **known** environments
  - Solution: fixed sequence of actions
  - Search: process of looking for the sequence of actions that reaches the goal
  - Agent can ignore percepts during execution

Search problem components

- **Initial state**
- **Successor Function (Operator)**
  - Result of doing an action in a state
- **Goal state**
- **Path cost**
  - Assume that it is a sum of nonnegative *step costs*

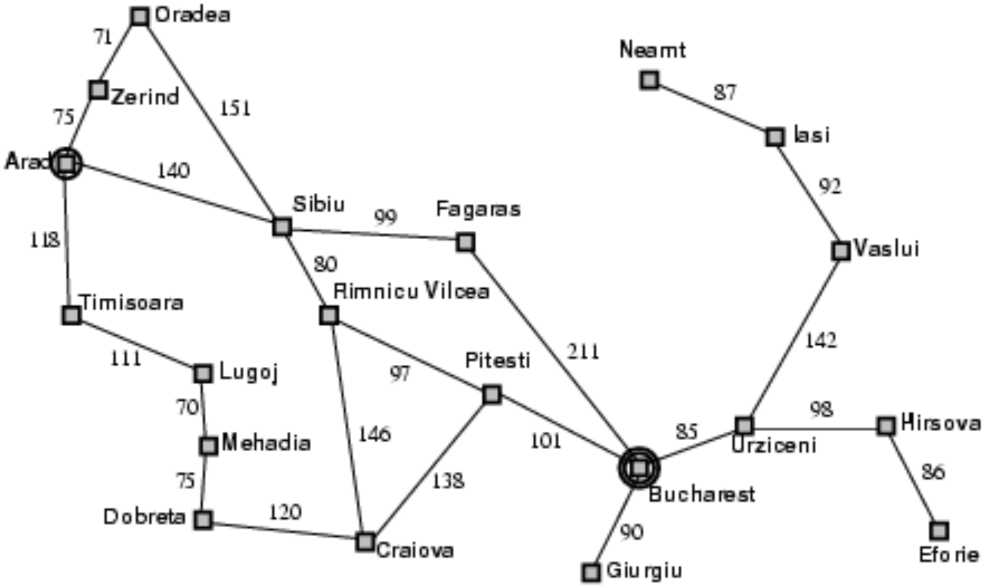


- **Optimal solution:** sequence of actions with the **lowest path cost** for reaching the goal

Example: Romania

- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

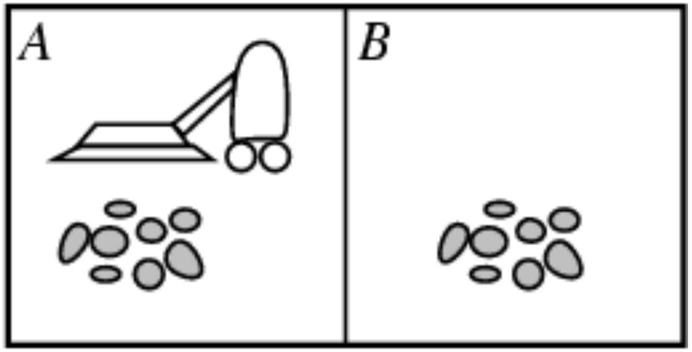
- **Initial state**
  - E.g. Arad
- **Successor Function**
  - S(Arad) = {Zerind, Timisoara, Sibiu}
- **Goal state**
  - Bucharest
- **Path Cost**
  - Sum of edge costs



State Space

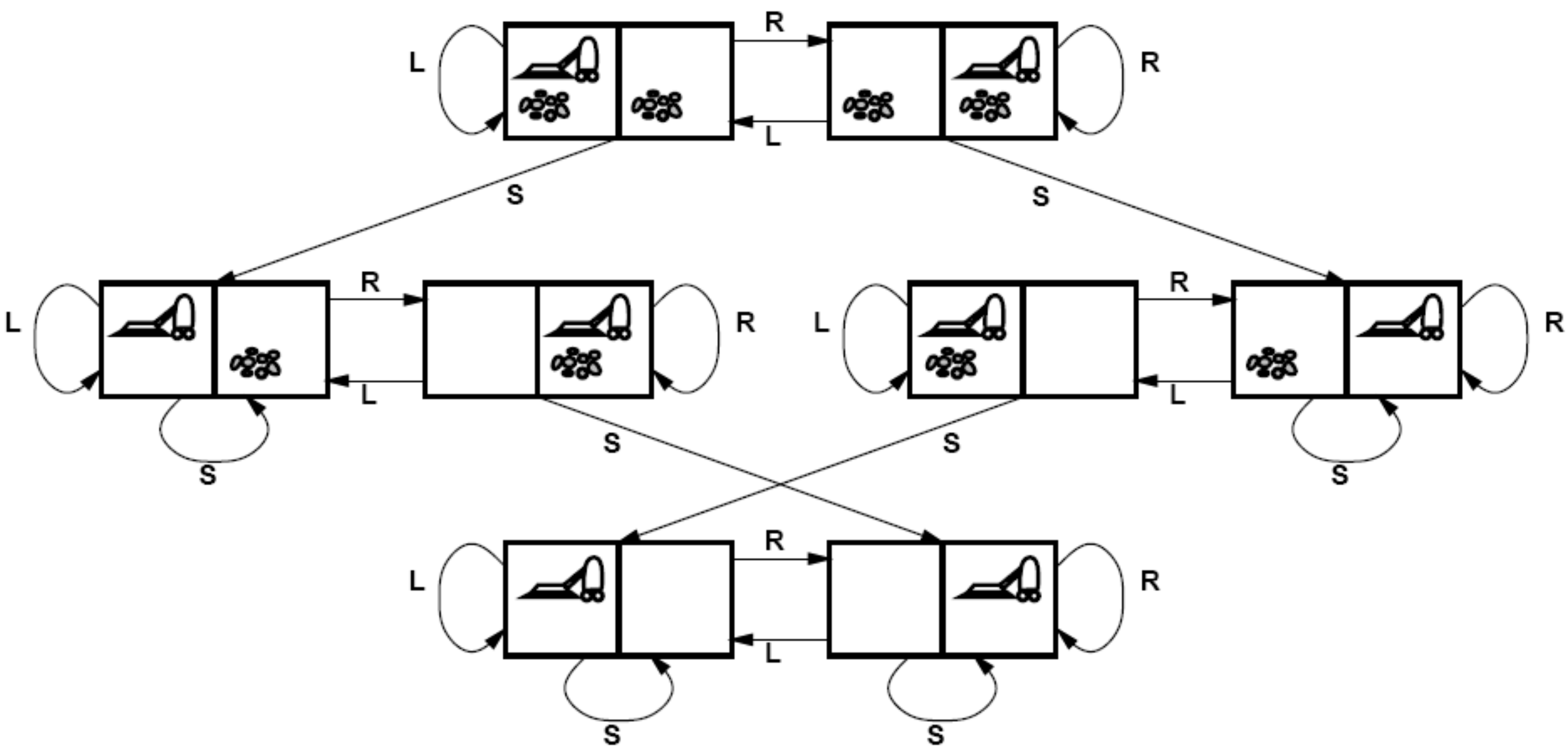
- The initial state and the successor function define the **state space** of the problem
  - The set of all states reachable from initial state by any sequence of actions
  - Can be represented as a **directed graph** (nodes are states and links between nodes are actions)
- What is the state space for the Romania problem?

Example: Vacuum world

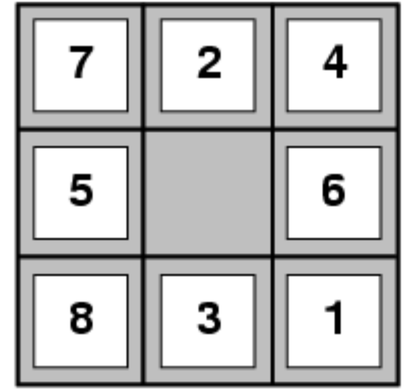


- **Initial State**
  - Any
- **Goal State**
  - All clean
- **Successor function**
  - Described by state space (next slide)
- **Path Cost**
  - Could be the sum of the amounts of electricity consumed with each move

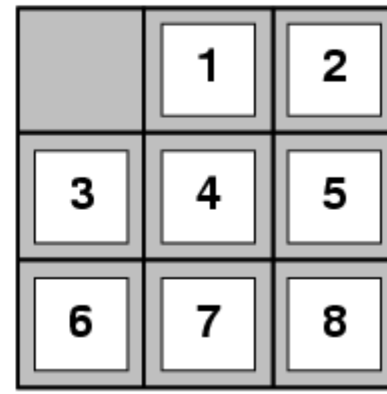
Vacuum world state space graph



Example: The 8-puzzle

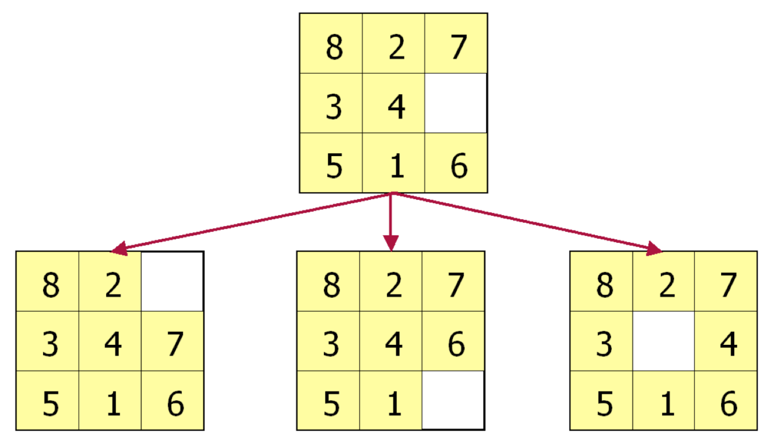


Start State



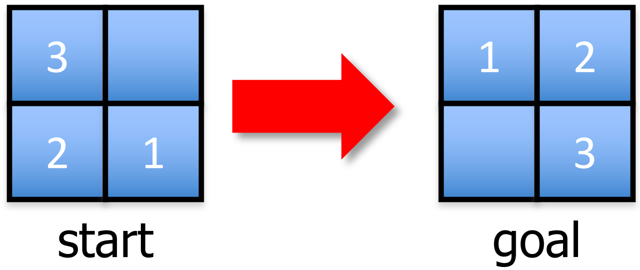
Goal State

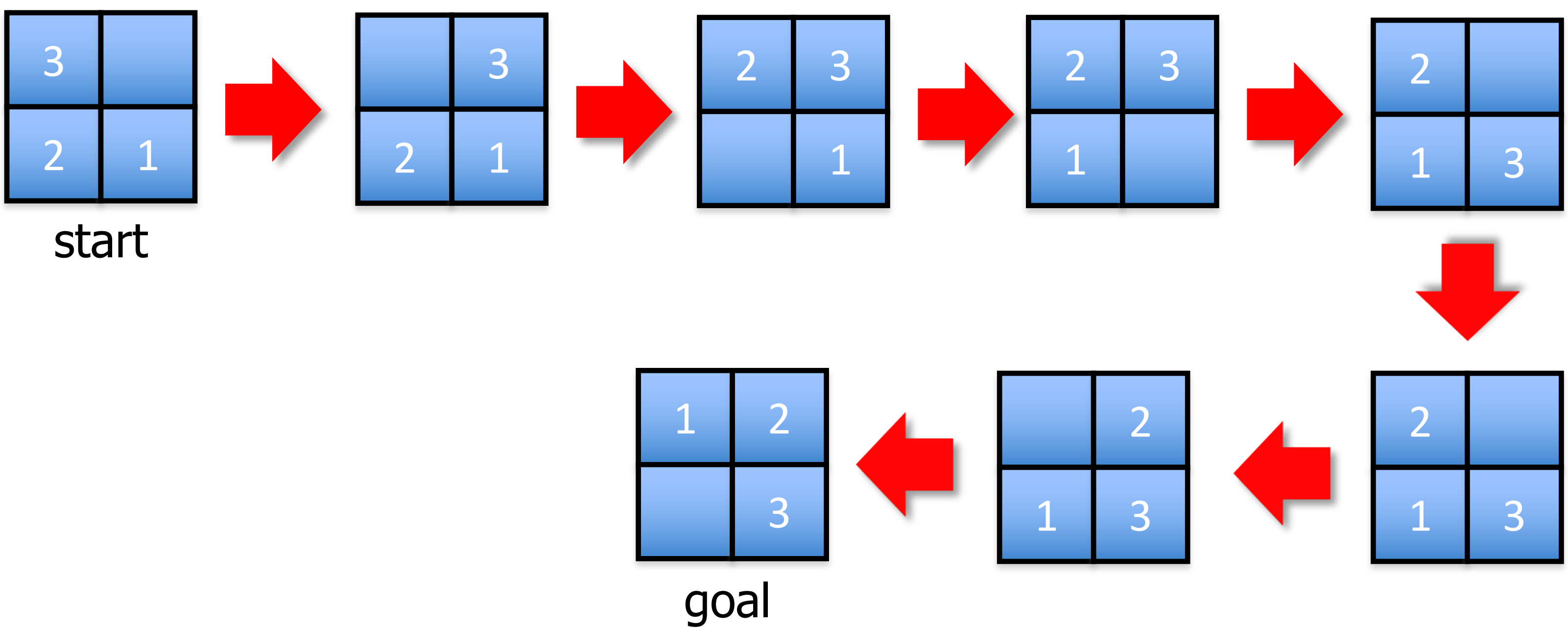
- **Initial State**
  - Any locations of tiles, number of states:
    - 8-puzzle: 181,440 states
    - 15-puzzle: 1.3 trillion states
    - 24-puzzle: 10^{25} states
- **Successor Function**
  - Actions: Move blank left, right, up, down, and consequent states
- **Path cost**
  - 1 per move



- Optimal solution of n-Puzzle is NP-hard

Simpler: 3-Puzzle





Building goal-based agents

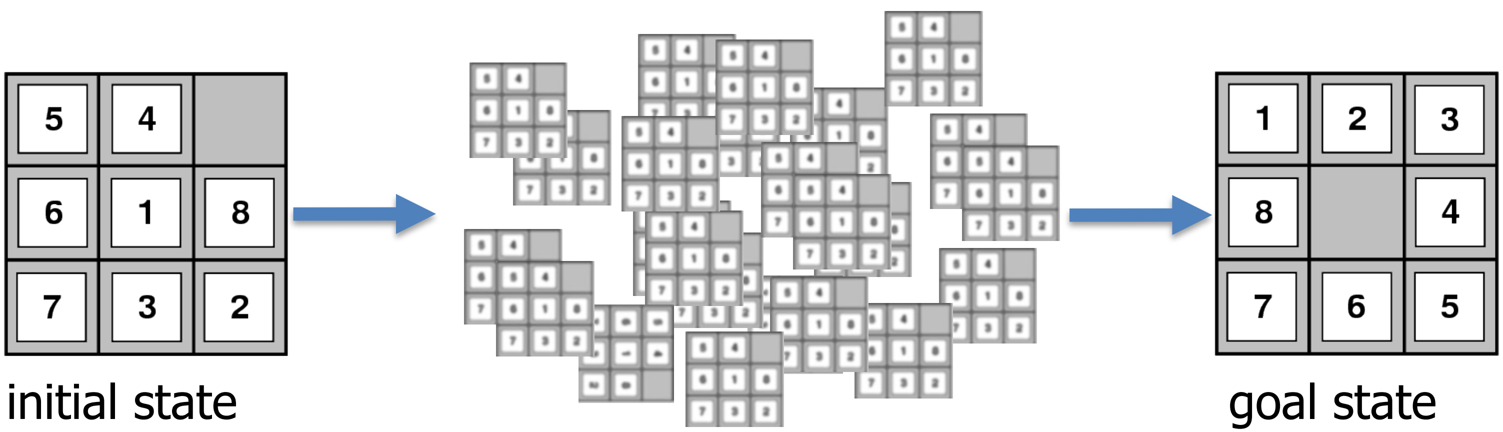
We must answer the following questions

– How do we represent the **state** of the "world"?

– What is the **goal** and how can we recognize it?

– What are the possible **actions**?

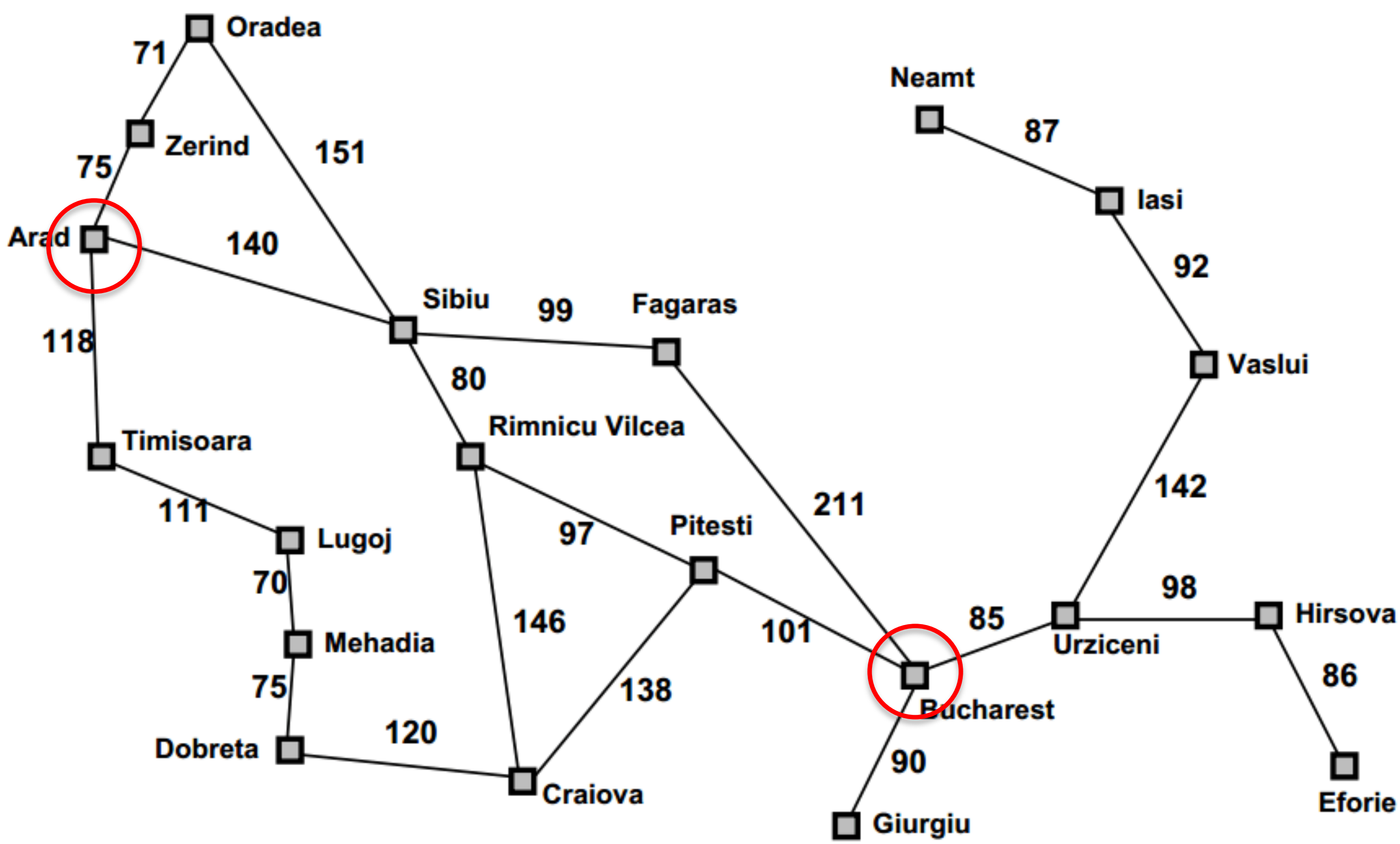
– What *relevant* information do we encode to describe states, actions and their effects and thereby solve the problem?



initial state

goal state

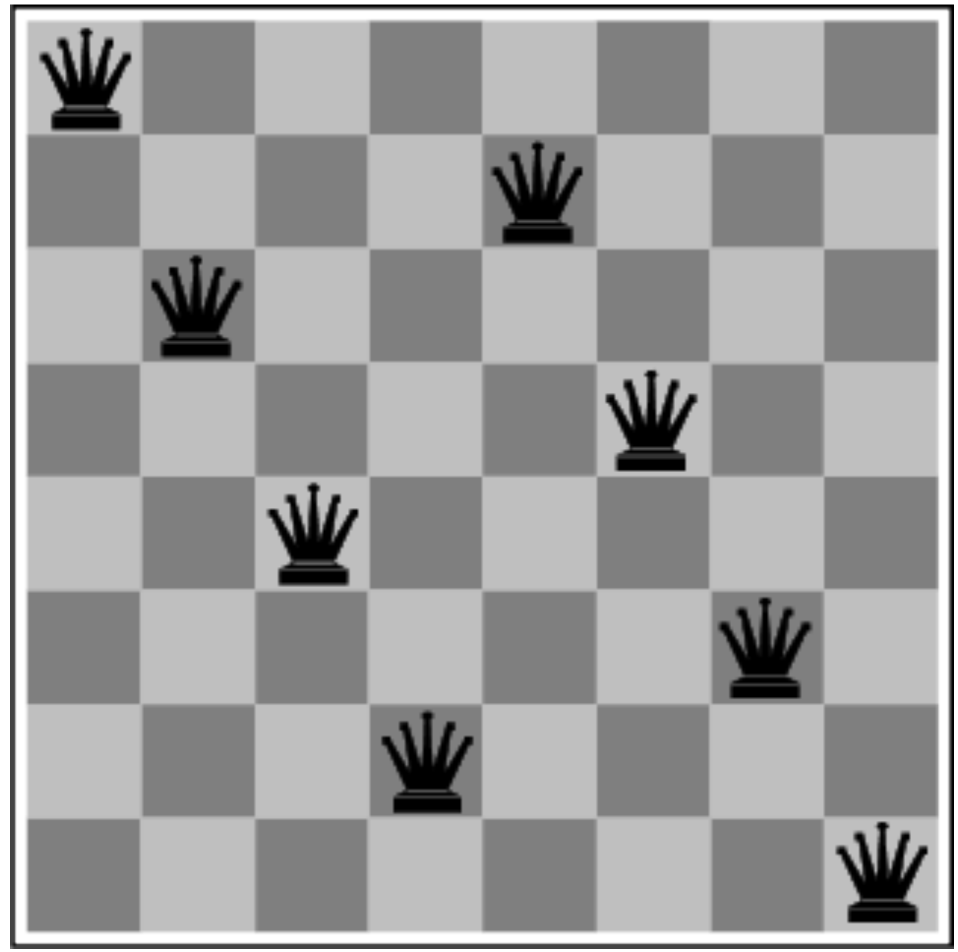
Example: Route Planning
Find a route from Arad to Bucharest



Example: The 8-Queens Puzzle

Place eight queens on a chessboard such that no queen attacks any other

We can generalize the problem to a NxN chessboard



What are the states, successor function, goal test?

Example: Remove 5 Sticks

Remove 5 Sticks Problem

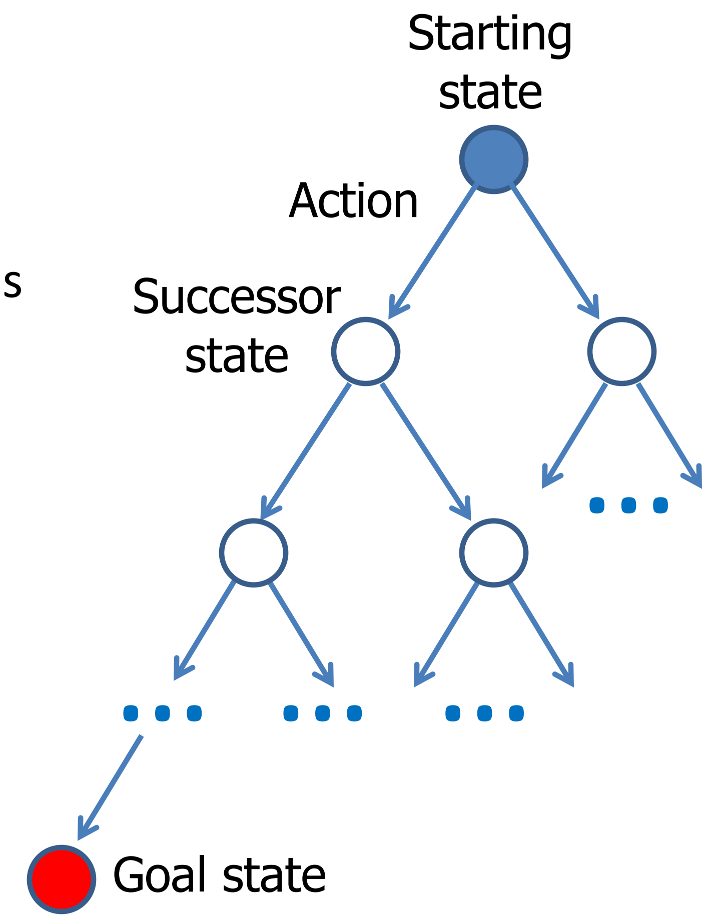
Remove exactly 5 of the 17 sticks so that the result forms exactly 3 squares

How could this be stated as a search problem? (States? Initial State? Successor Function? Goal Test? Path Cost?)

Tree Search

- Begin at the root node (starting state) and **expand** it by making a list of all possible successor states
- Maintain a **fringe** or a list of unexpanded states
- At each step, pick a state from the fringe to expand
- Keep going until you reach the goal state
- Try to expand as few states as possible
- A solution is a path ending in the goal state

- Nodes vs. states
  – A state is a representation of a physical configuration, while a node is a data structure that is part of the search tree

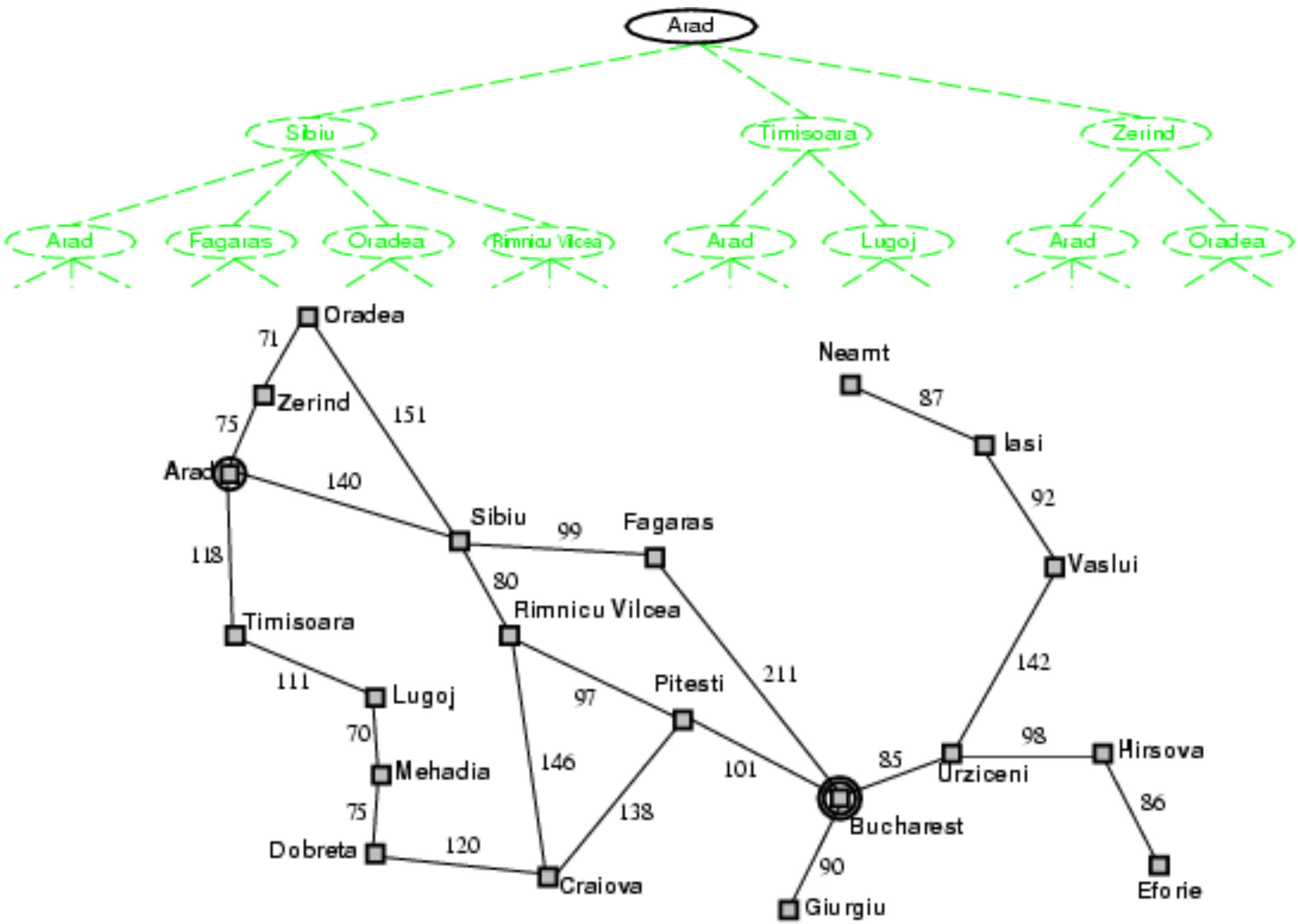


Tree Search Algorithm Outline

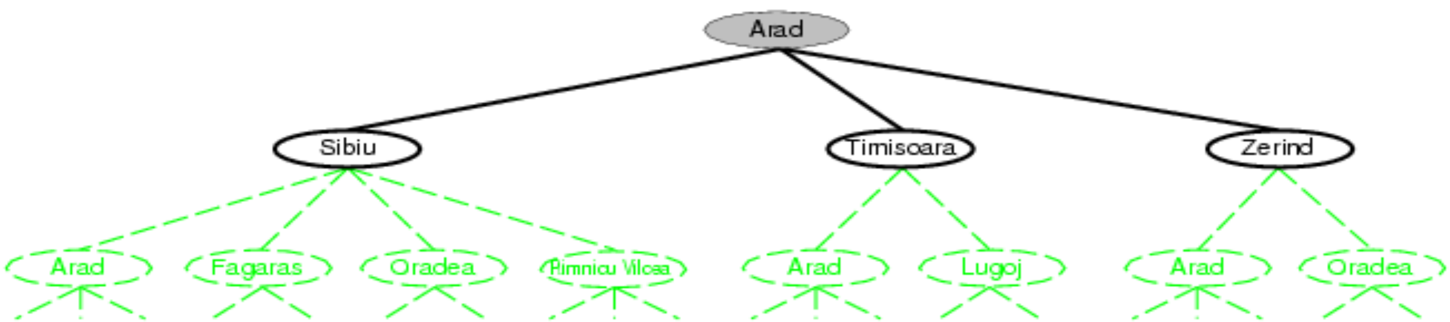
- Initialize the **fringe** using the **starting state**
- While the fringe is not empty
  - Choose a fringe node to expand according to **search strategy**
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the fringe

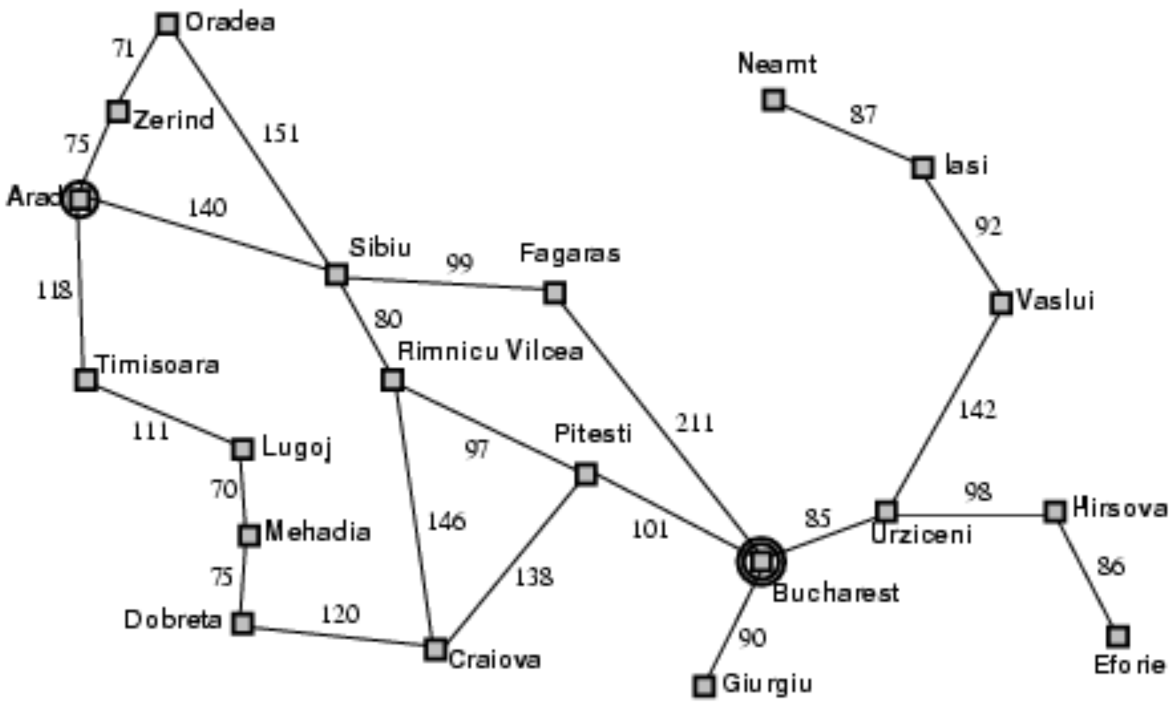
- To handle repeated states:
  - Keep an **explored set**; add each node to the explored set every time you expand it
  - Every time you add a node to the fringe, check whether it already exists in the fringe with a higher path cost, and if yes, replace that node with the new one

Tree search example

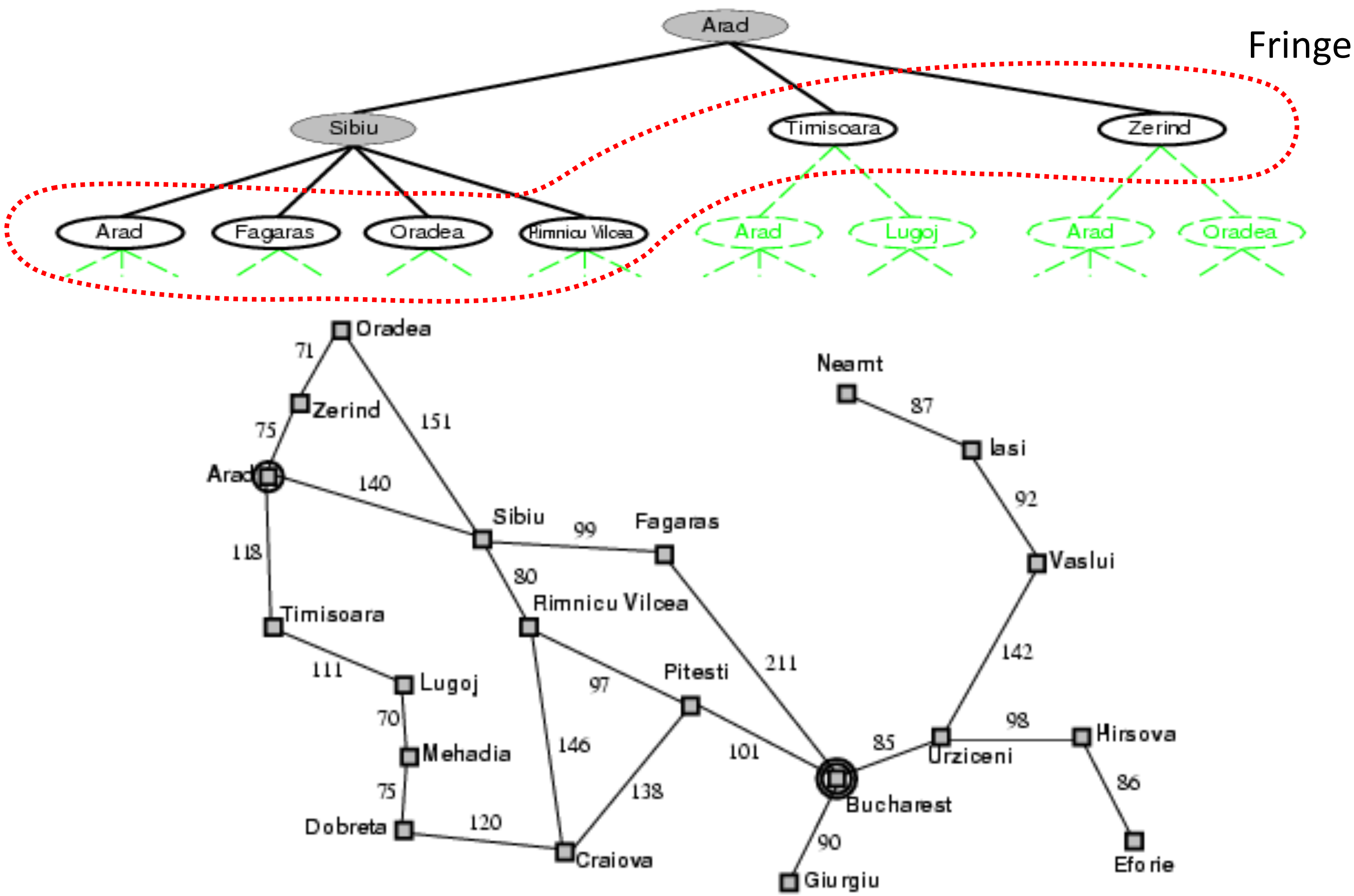


Tree search example





Tree search example



Search strategies

- A **search strategy** is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - **Completeness:** does it always find a solution if one exists?
  - **Optimality:** does it always find a least-cost solution?
  - **Time complexity:** number of nodes generated
  - **Space complexity:** maximum number of nodes in memory
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost (cheapest) solution
  - *m*: maximum length of any path in the state space (may be infinite)

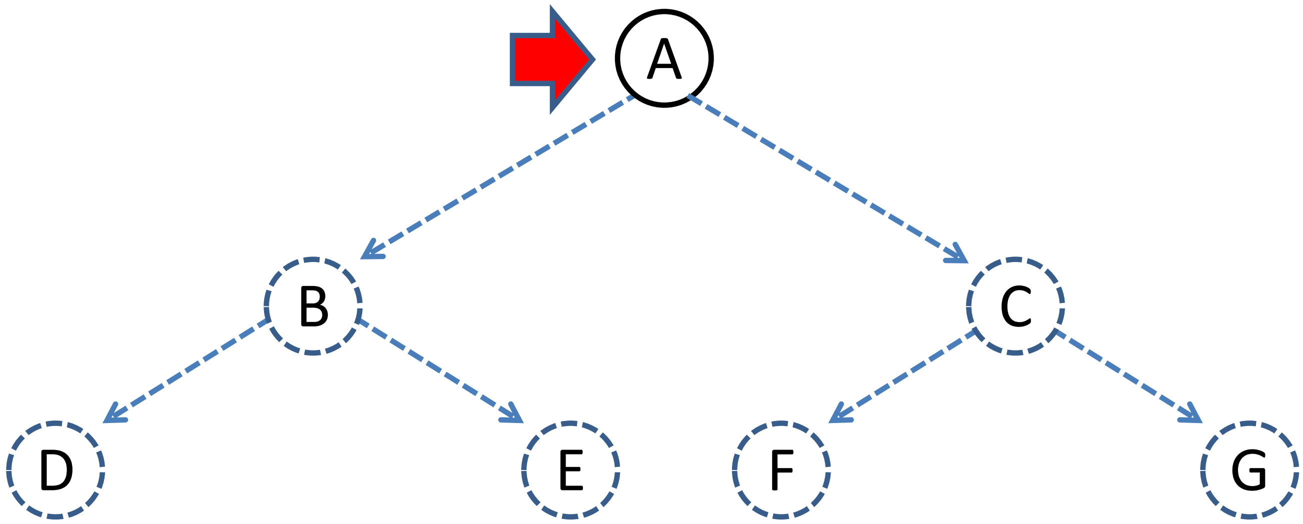
Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition

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

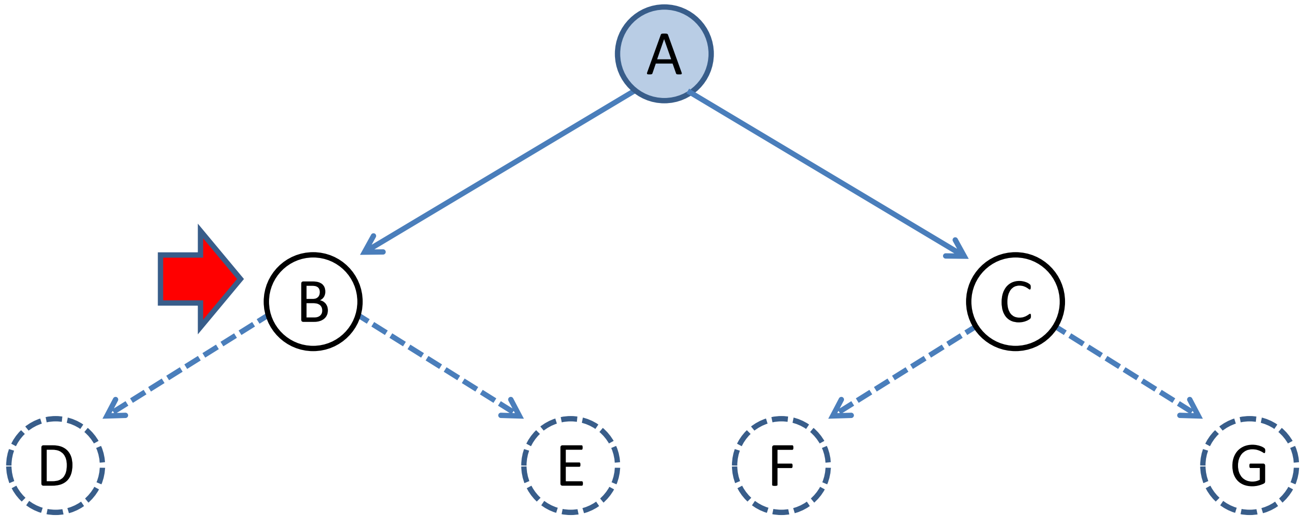
Breadth-first search

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



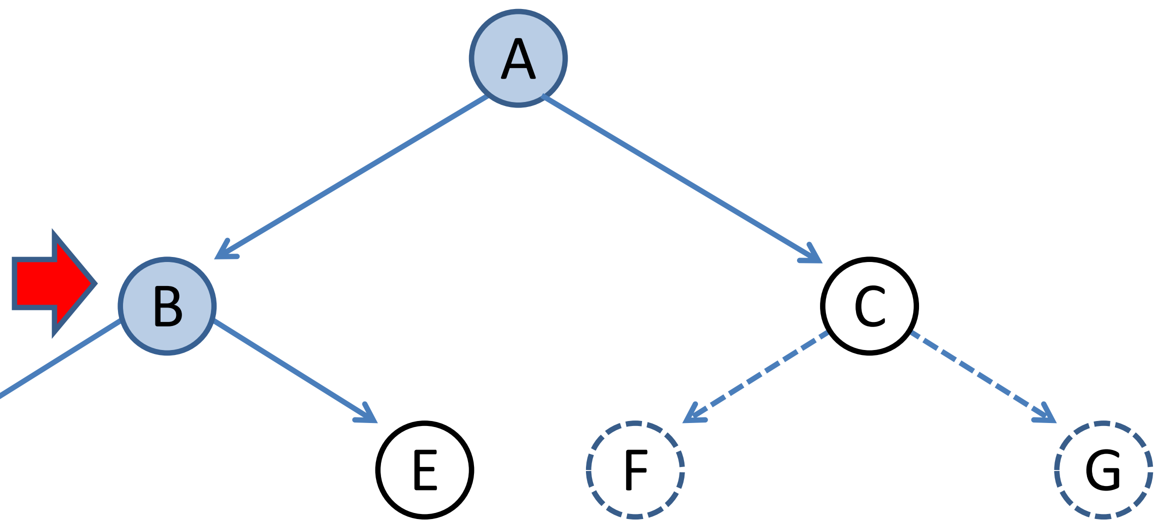
Breadth-first search

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



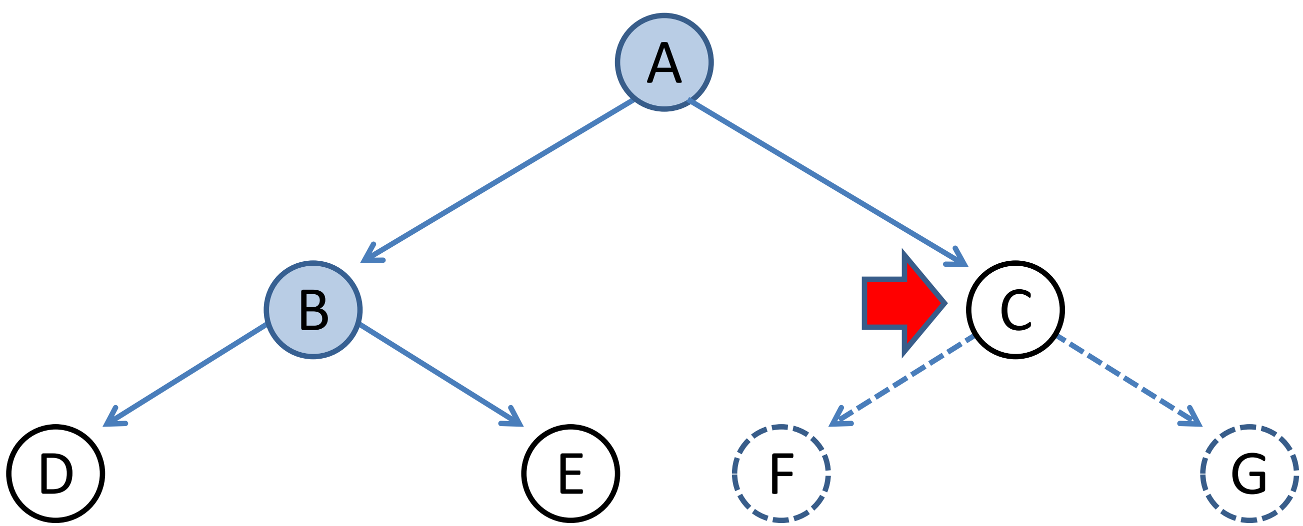
Breadth-first search

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



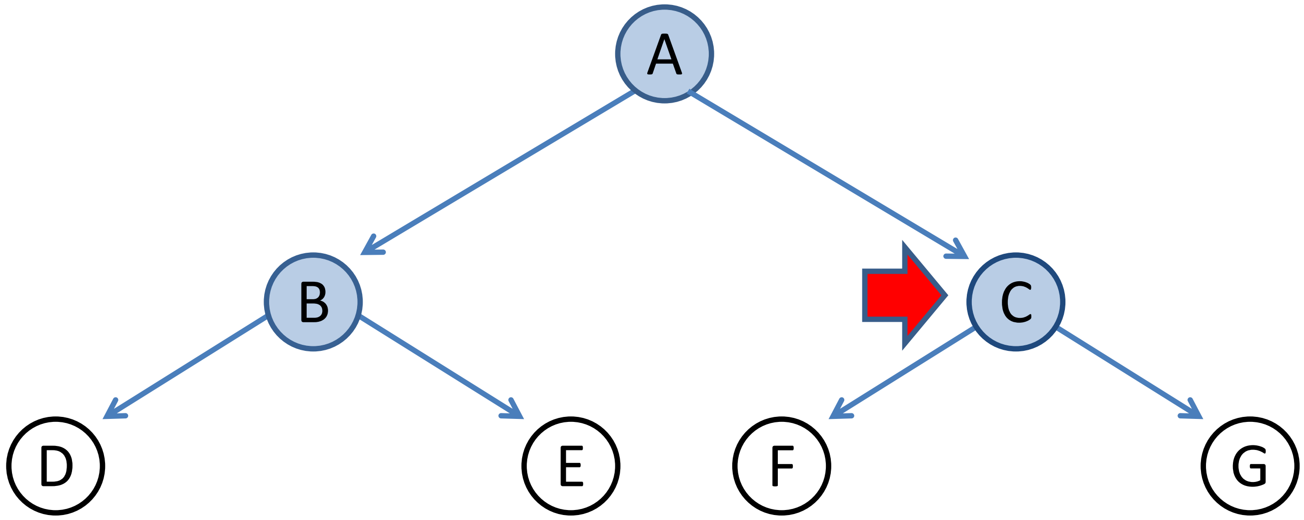
Breadth-first search

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



Breadth-first search

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



Properties of breadth-first search

- **Complete?**

  Yes

- **Optimal?**

  Yes – if cost = 1 per step

- **Time?**

  Number of nodes in a b-ary tree of depth d: $O(b^d)$

  (d is the depth of the optimal solution)

- **Space?**

  $O(b^d)$

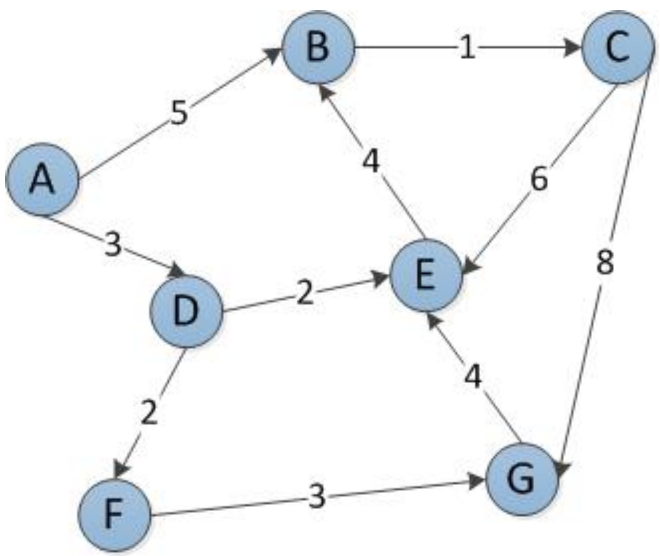
- Space is the bigger problem (more than time)

Uniform-cost search

- Expand least-cost unexpanded node
- Implementation: *fringe* ordered by path cost (priority queue)
- Equivalent to breadth-first if step costs all equal

- **Complete?**
  Yes
- **Optimal?**
  Yes – nodes expanded in increasing order of path cost
- **Time?**
  Number of nodes with path cost ≤ cost of optimal solution (C^*), $O(b^{C^*/\varepsilon})$, where every step cost is at least $\varepsilon > 0$
  This can be greater than $O(b^d)$: the search can explore long paths consisting of small steps before exploring shorter paths consisting of larger steps
- **Space?**
  $O(b^{C^*/\varepsilon})$

Uniform-Cost Search Example



Start Node: A

Goal Node: G

Step 1

Fringe:

Node A

Cost 0

Explored: -

Step 2

Expand A

Fringe:

Node D B

Cost 3 5

Explored: A

Step 3

Expand D

Fringe:

Node B E F

Cost 5 5 5

Explored: A D

Step 4

Expand B

Fringe:

Node E F C

Cost 5 5 6

Explored: A D B

Step 5

Expand E

Fringe:

Node F C

Cost 5 6

Explored: A D B E

Step 6

Expand F

Fringe:

Node C G

Cost 6 8

Explored: A D B E F

Step 7

Expand C

Fringe:

Node G

Cost 8

Explored: A D B E F C

Step 8

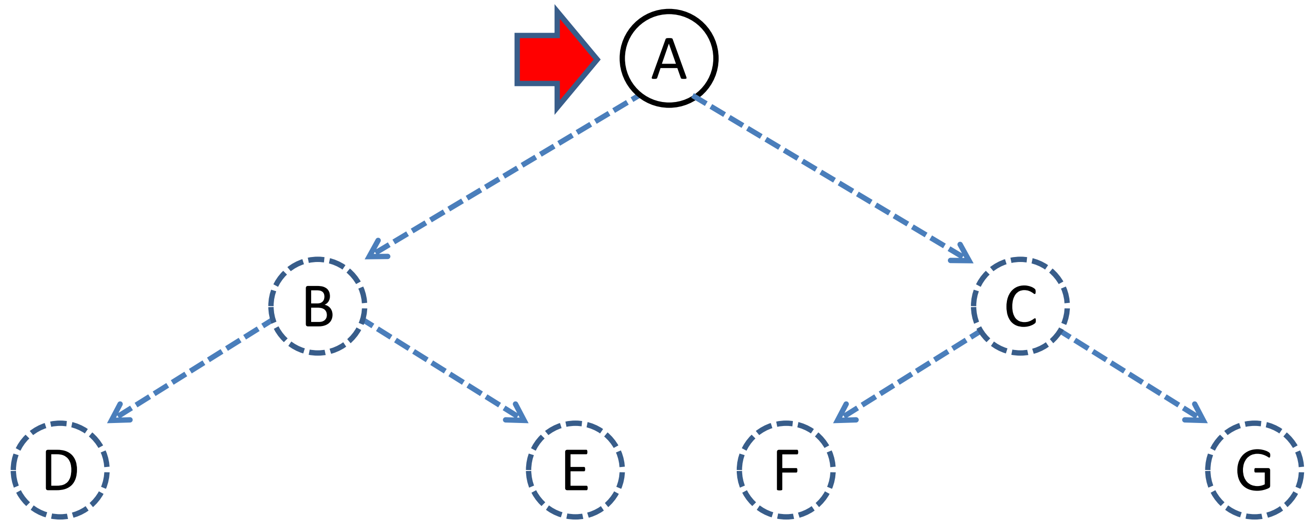
Expand G

Found the path:

A to D to F to G

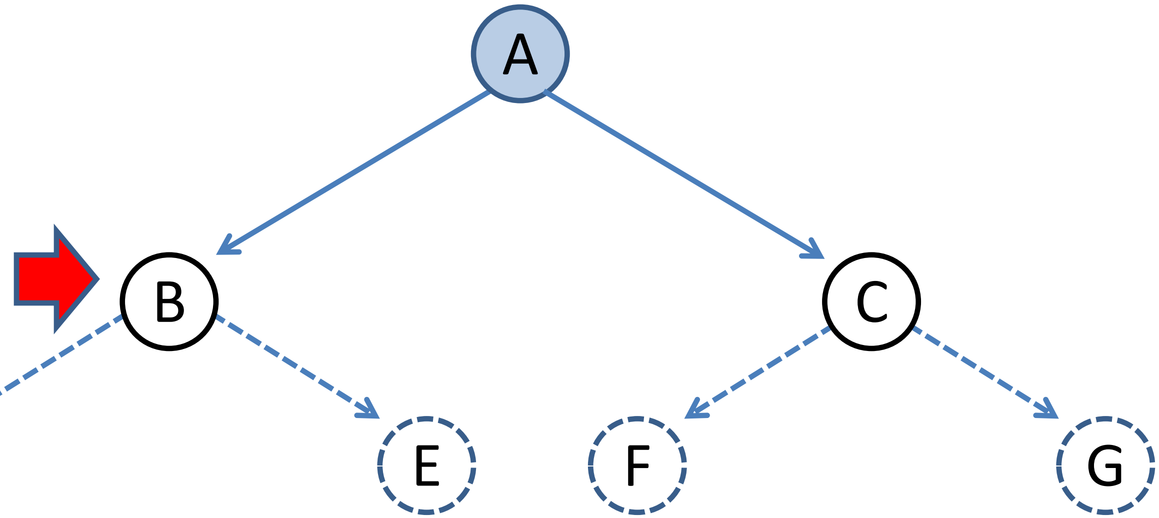
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



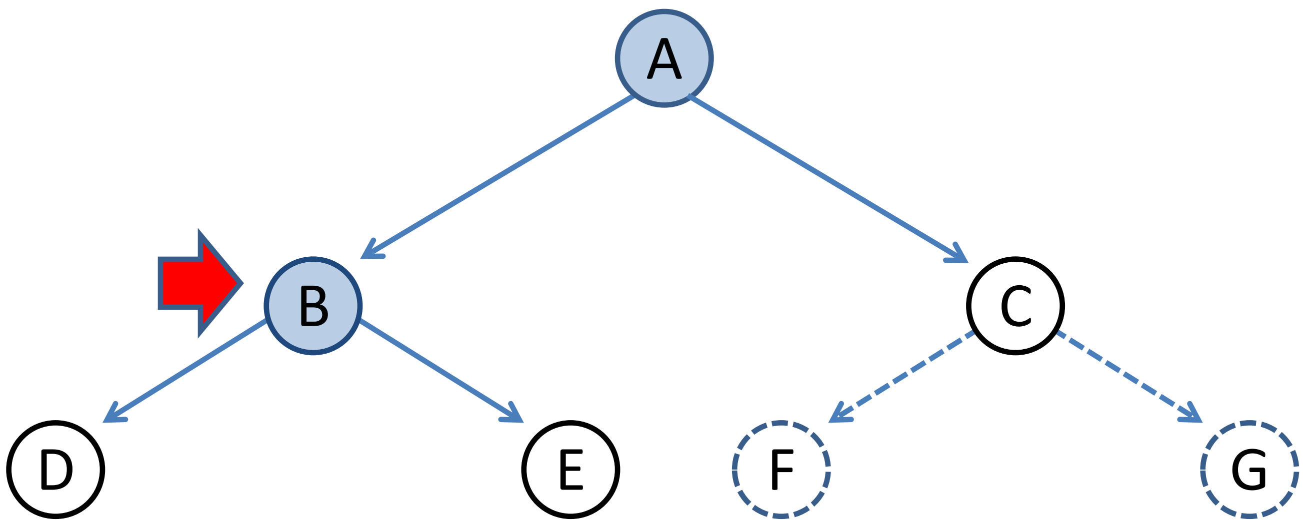
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



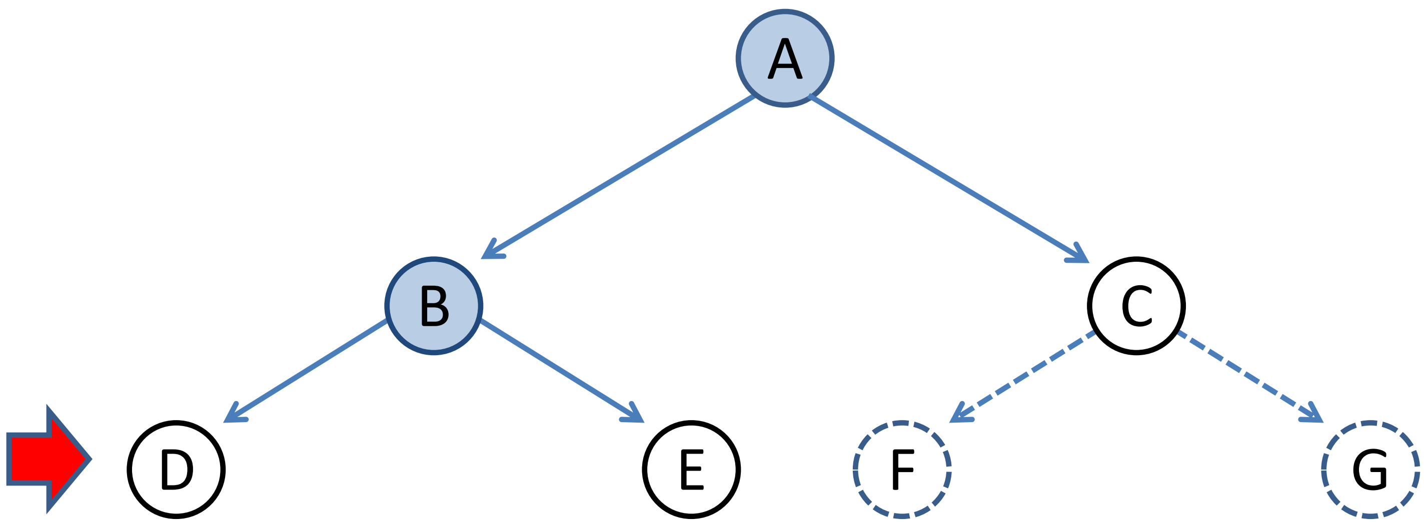
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



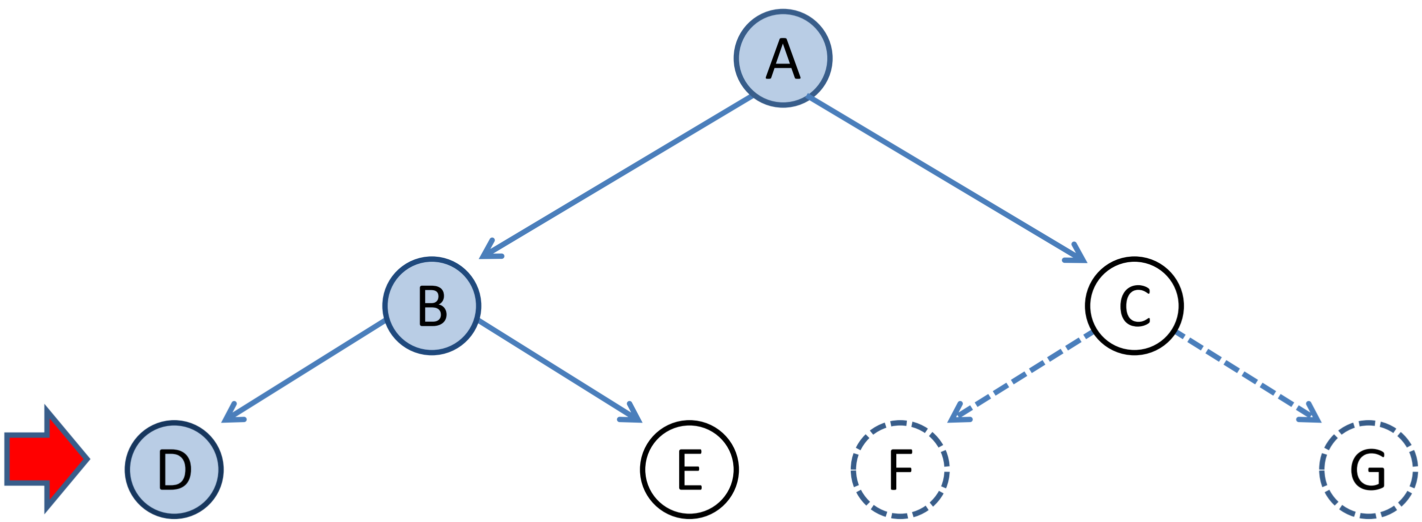
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



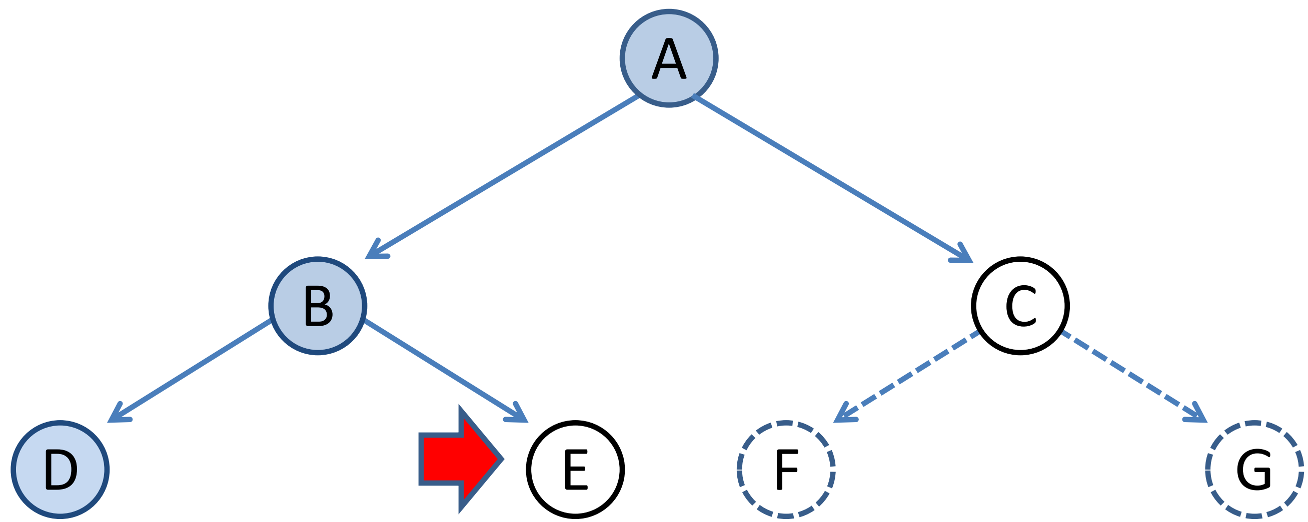
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



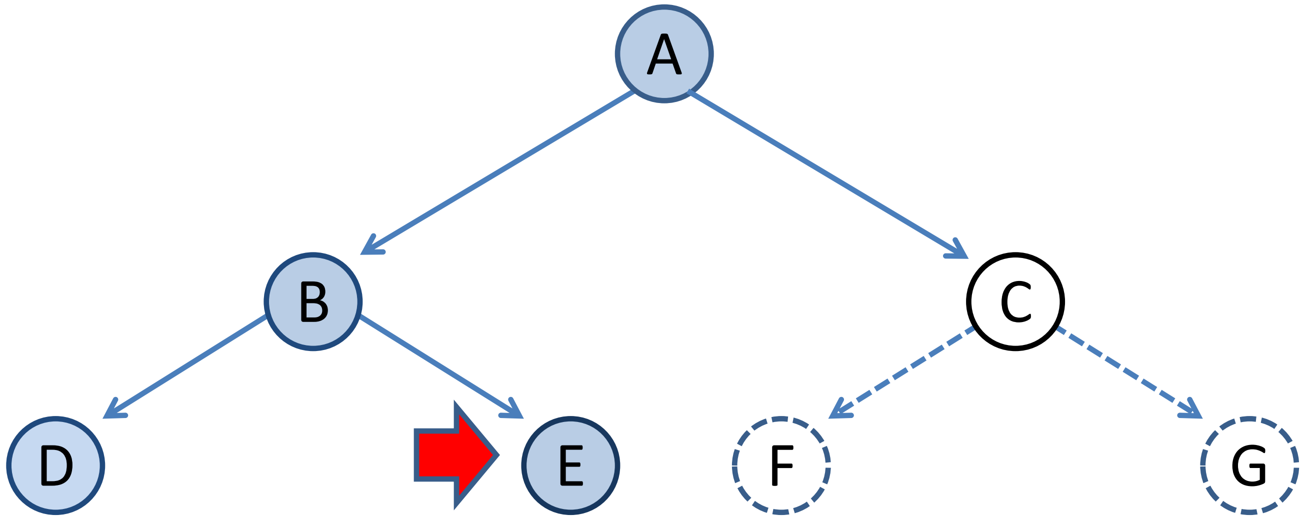
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



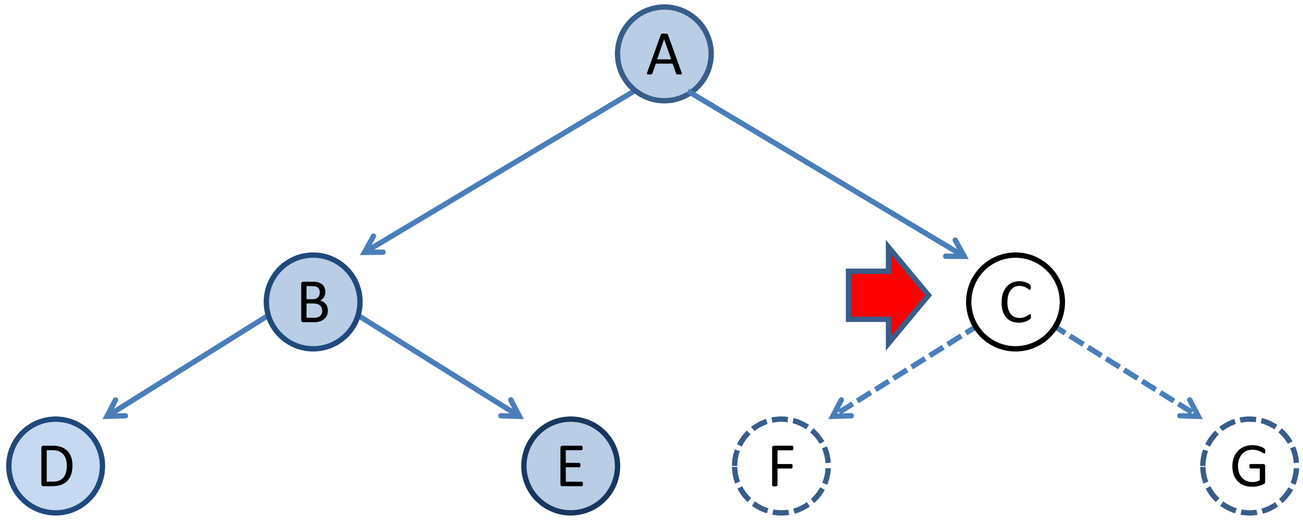
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



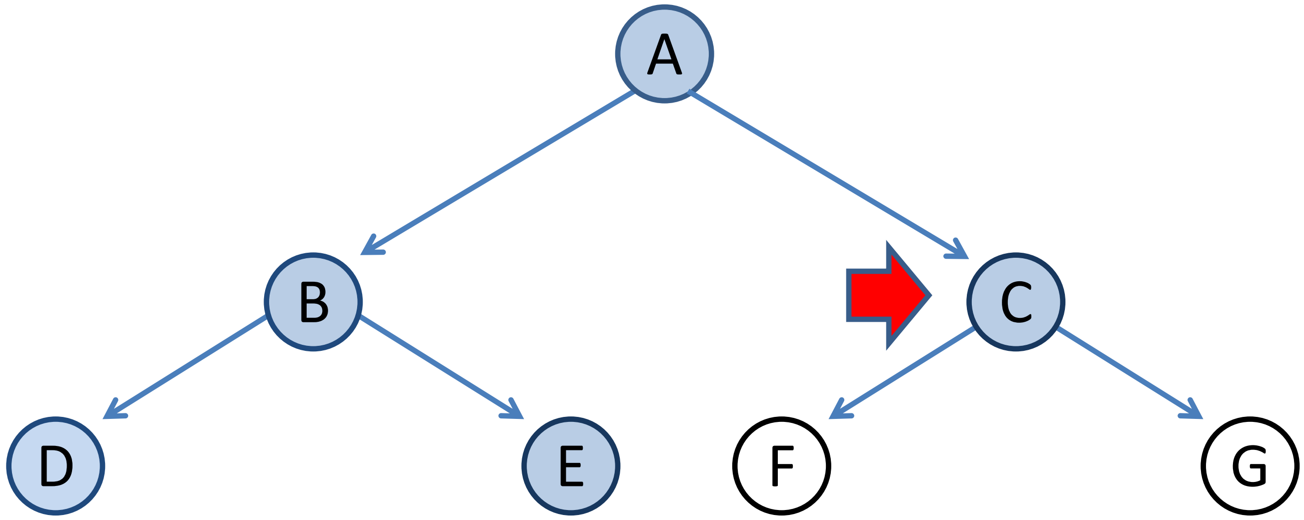
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Properties of depth-first search

- **Complete?**

  Fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path

  → complete in finite spaces

- **Optimal?**

  No – returns the first solution it finds

- **Time?**

  Could be the time to reach a solution at maximum depth *m*: $O(b^m)$

  Terrible if *m* is much larger than *d*

  But if there are lots of solutions, may be much faster than BFS

- **Space?**

  $O(bm)$, i.e. linear space!

Iterative deepening search

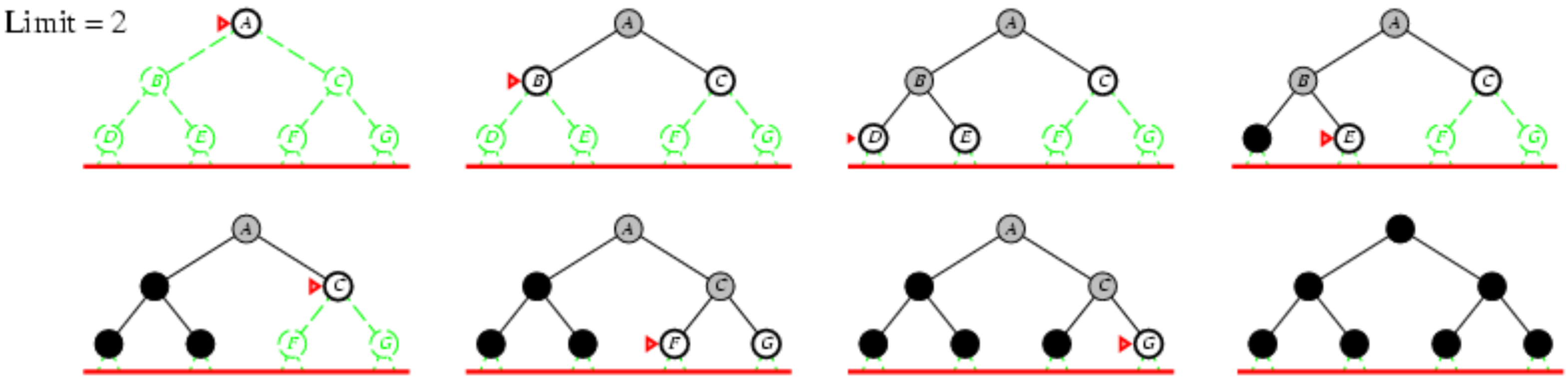
- Use DFS as a subroutine
  1. Check the root
  2. Do a DFS searching for a path of length 1
  3. If there is no path of length 1, do a DFS searching for a path of length 2
  4. If there is no path of length 2, do a DFS searching for a path of length 3…

Iterative deepening search

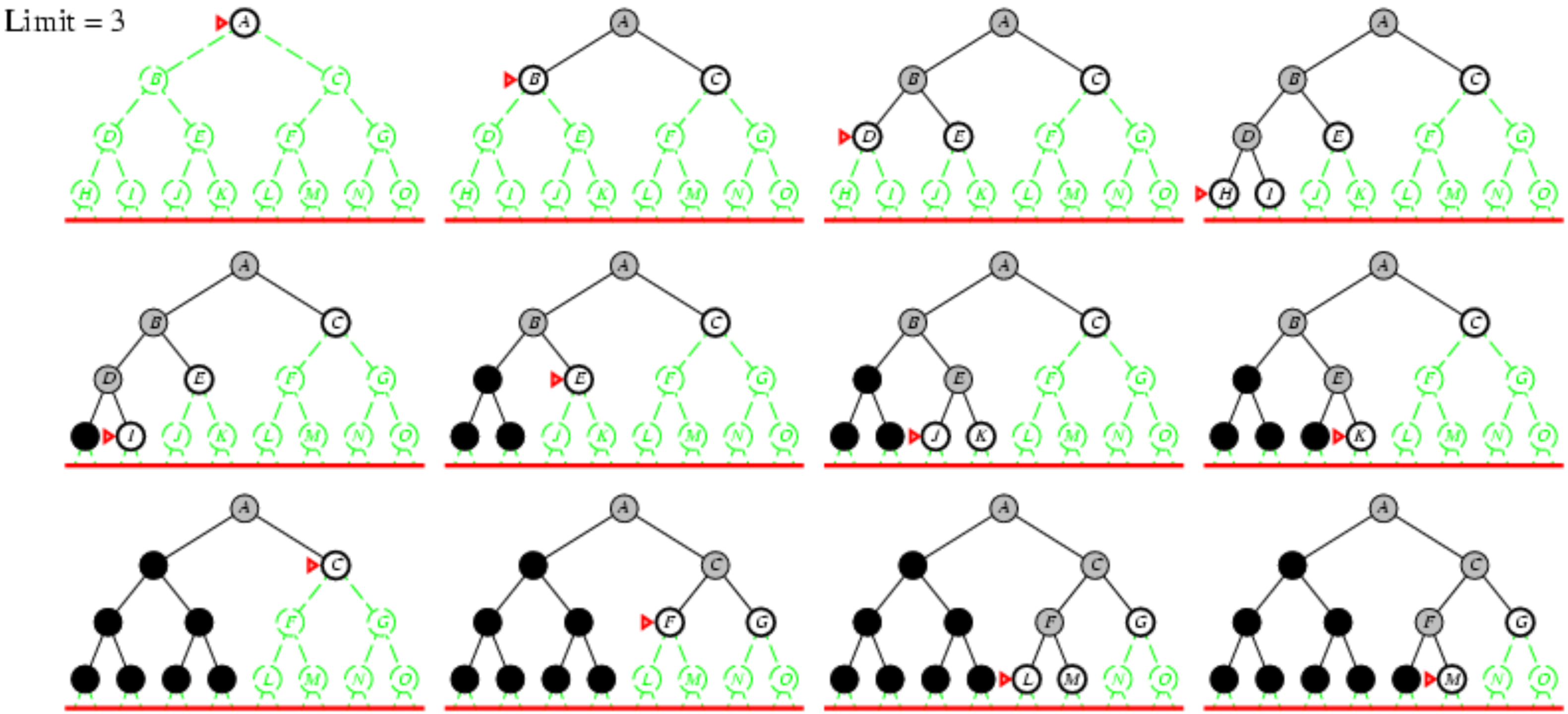
Limit = 0

Iterative deepening search

Iterative deepening search



Iterative deepening search



Properties of iterative deepening search

- **Complete?**

  Yes

- **Optimal?**

  Yes, if step cost = 1

- **Time?**

  $(d+1)b^0 + d\ b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

- **Space?**

  $O(bd)$

Next Class

- Informed Search