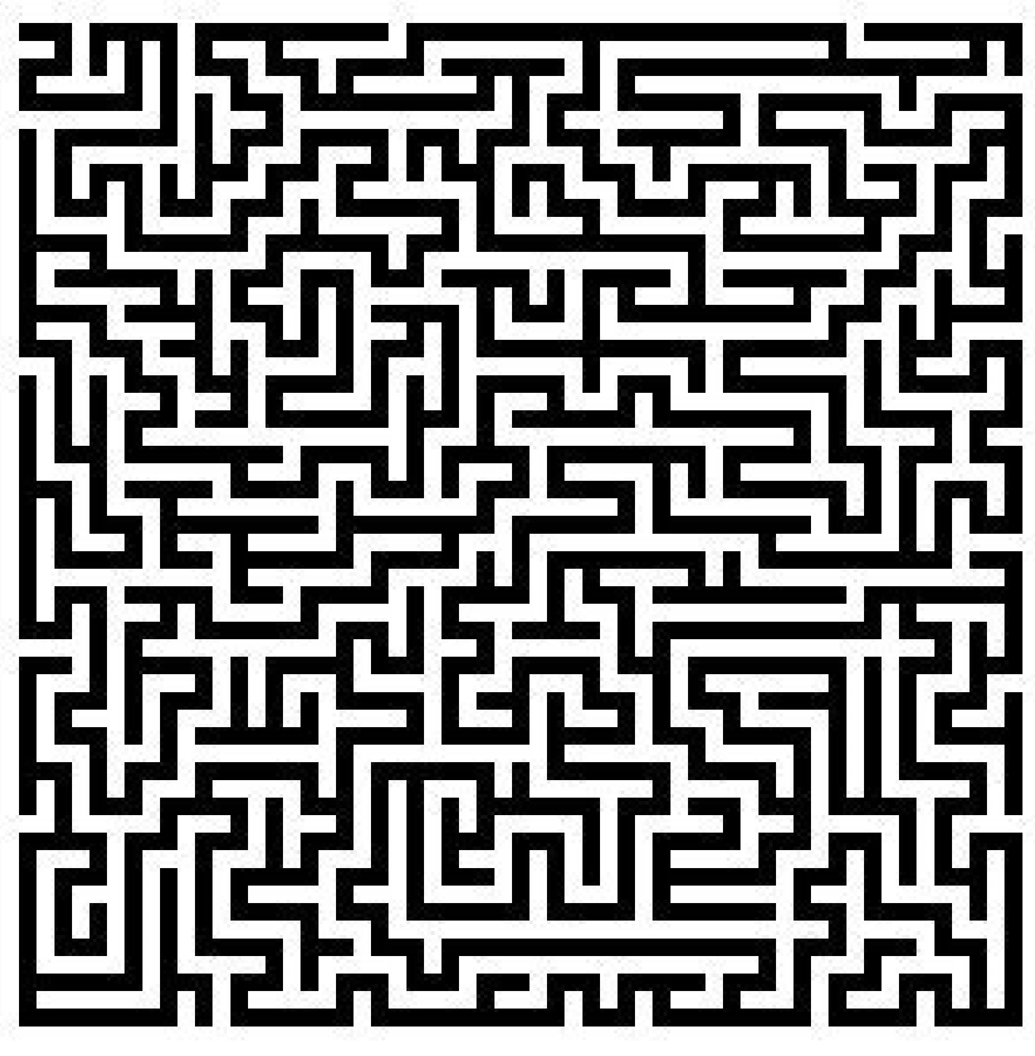
Chapter 3
Solving problems by searching



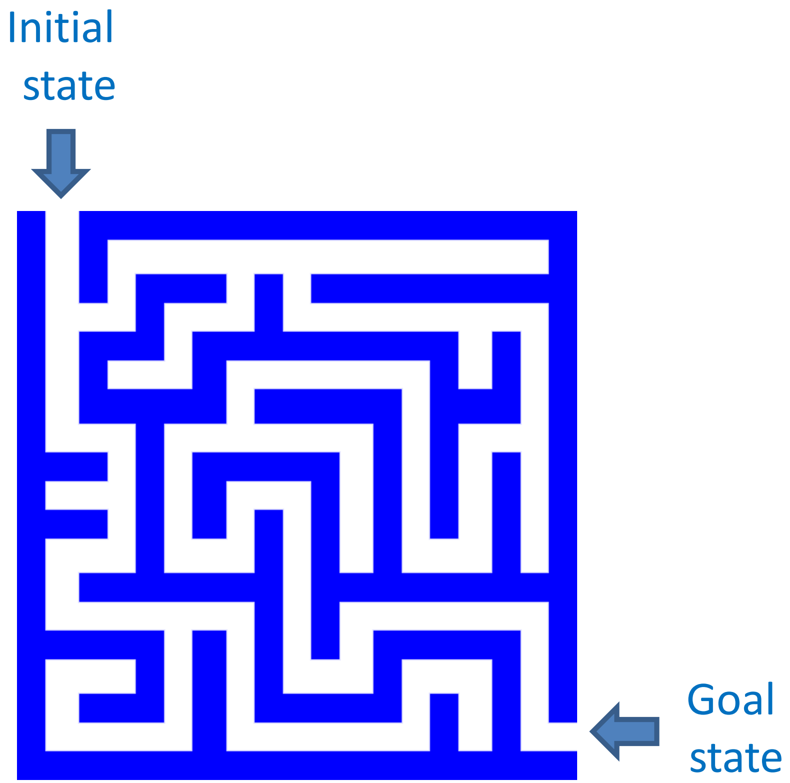
Search

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

- The solution is a fixed sequence of actions
- Search is the process of looking for the sequence of actions that reaches the goal
- Once the agent begins executing the search solution, it can ignore its percepts (**open-loop system**)

Search problem components

- **Initial state**
- **Actions**
- **Transition model**
  - What is the result of performing a given action in a given state?
- **Goal state**
- **Path cost**
  - Assume that it is a sum of nonnegative *step costs*

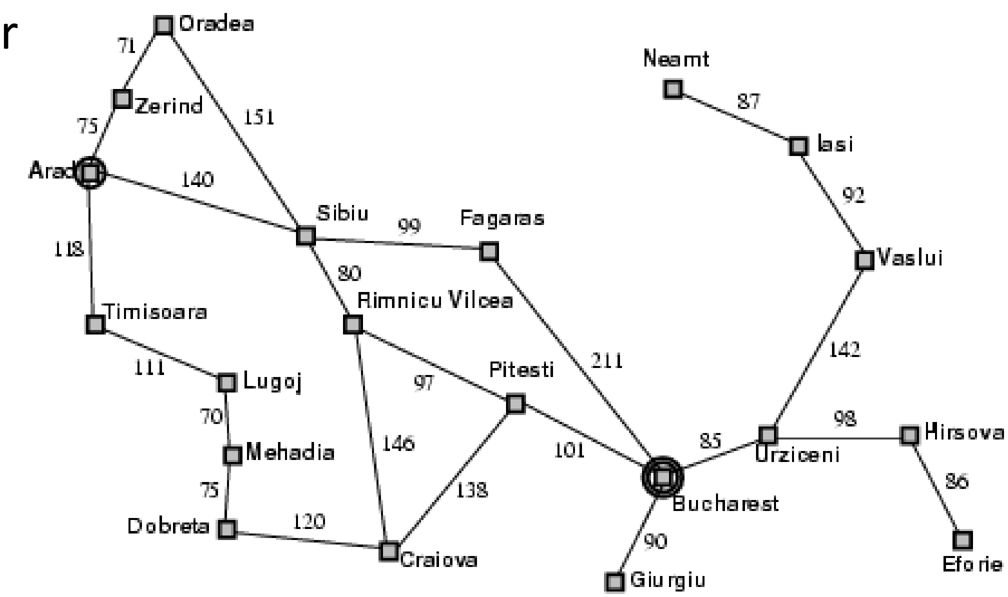


- The **optimal solution** is the sequence of actions that gives the lowest path cost for reaching the goal

Example: Romania

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

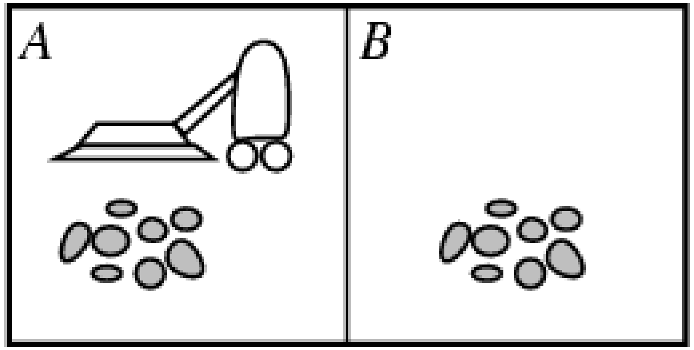
- **Initial state**
  - Arad
- **Actions**
  - Go from one city to another
- **Transition model**
  - If you go from city A to city B, you end up in city B
- **Goal state**
  - Bucharest
- **Path cost**
  - Sum of edge costs



State space

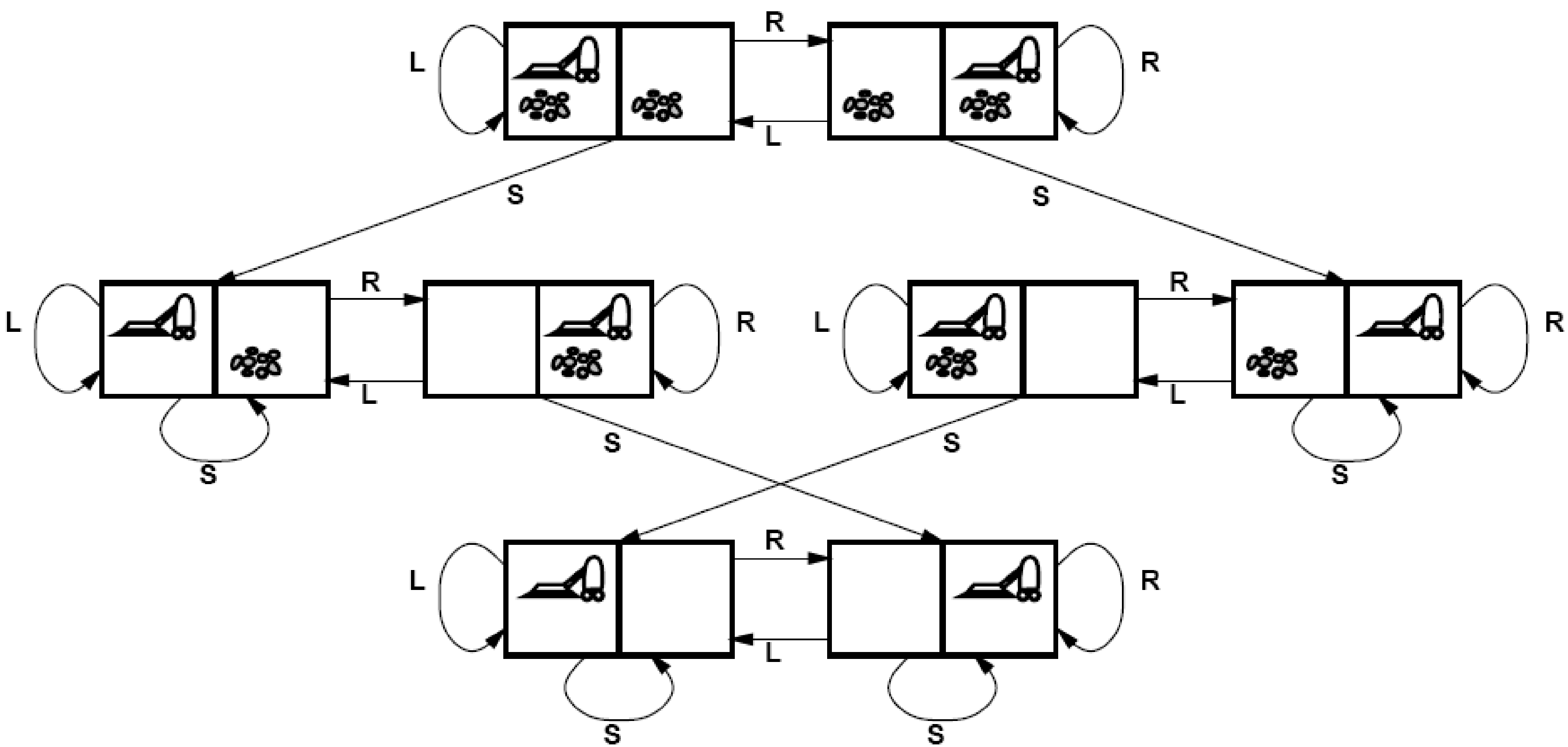
- The initial state, actions, and transition model define the **state space** of the problem
  - The set of all states reachable from initial state by any sequence of actions
  - Can be represented as a **directed graph** where the nodes are states and links between nodes are actions
- What is the state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

Example: Vacuum world



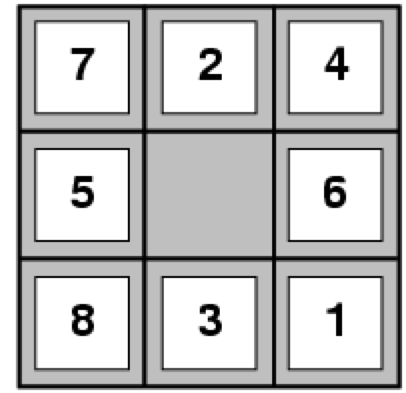
- **States**
  - Agent location and dirt location
  - How many possible states?
  - What if there are *n* possible locations?
- **Actions**
  - Left, right, suck
- **Transition model**

Vacuum world state space graph

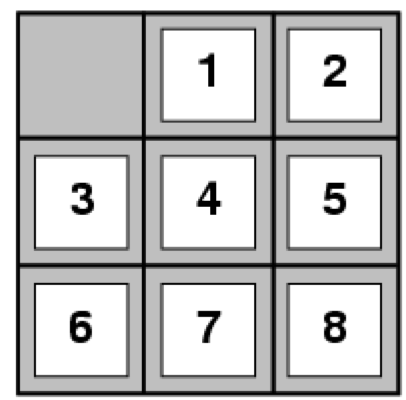


Example: The 8-puzzle

- **States**
  - Locations of tiles
    - 8-puzzle: 181,440 states
    - 15-puzzle: 1.3 trillion states
    - 24-puzzle: 10^{25} states
- **Actions**
  - Move blank left, right, up, down
- **Path cost**
  - 1 per move



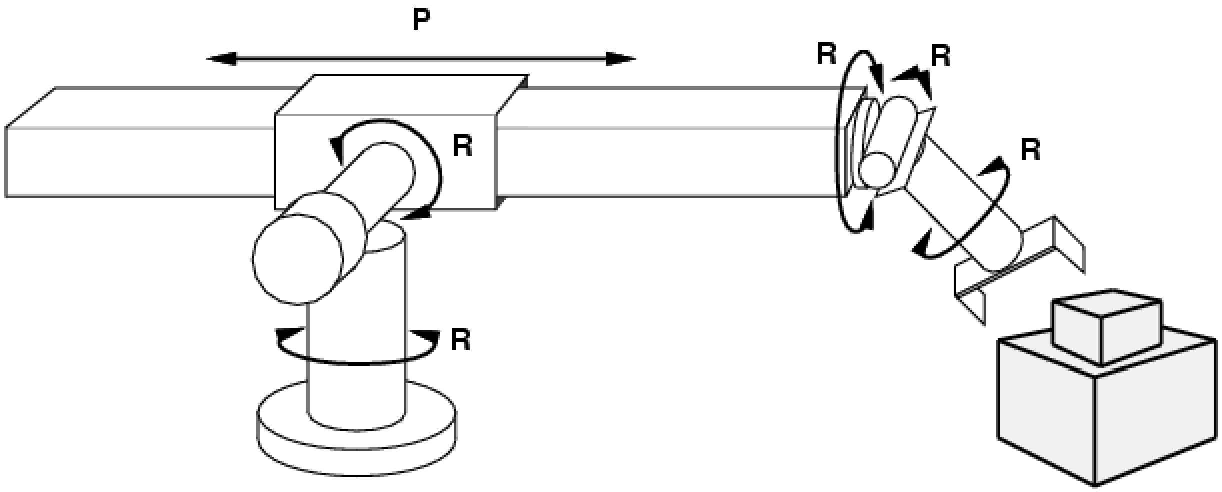
Start State



Goal State

- Optimal solution of n-Puzzle is NP-hard

Example: Robot motion planning



- **States**
  - Real-valued coordinates of robot joint angles
- **Actions**
  - Continuous motions of robot joints
- **Goal state**
  - Desired final configuration (e.g., object is grasped)
- **Path cost**
  - Time to execute, smoothness of path, etc.

Other Real-World Examples

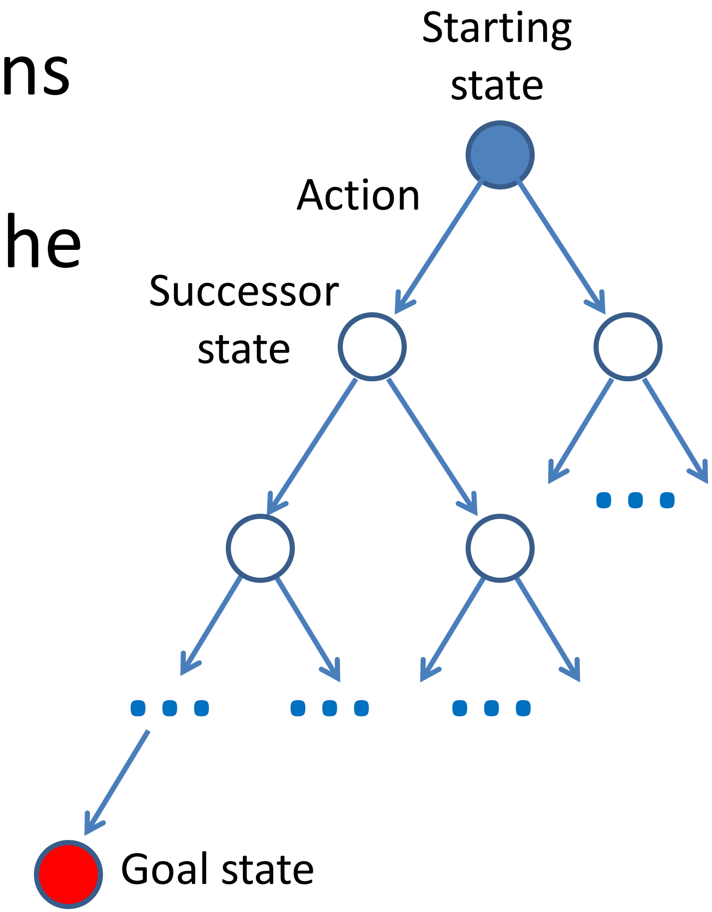
- Routing
- Touring
- VLSI layout
- Assembly sequencing
- Protein design

Tree Search

- Let's begin at the start node 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

Search tree

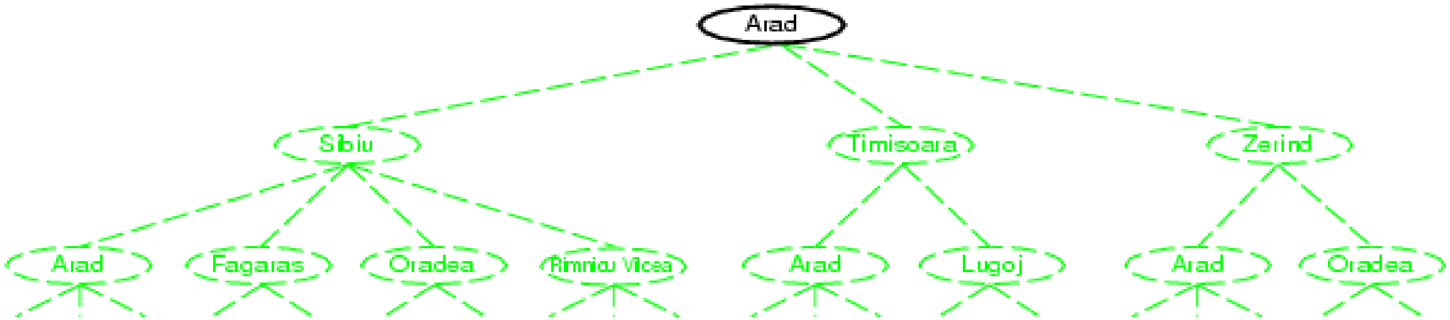
- "What if" tree of possible actions and outcomes
- The root node corresponds to the starting state
- The children of a node correspond to the **successor states** of that node's state
- A path through the tree corresponds to a sequence of actions
  - A solution is a path ending in the goal state

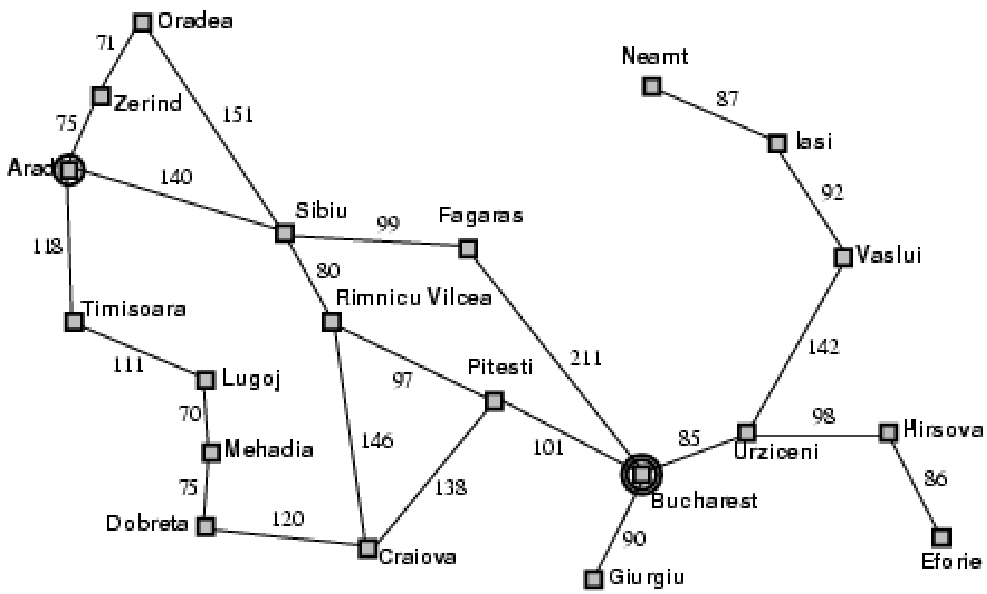


Tree Search Algorithm Outline

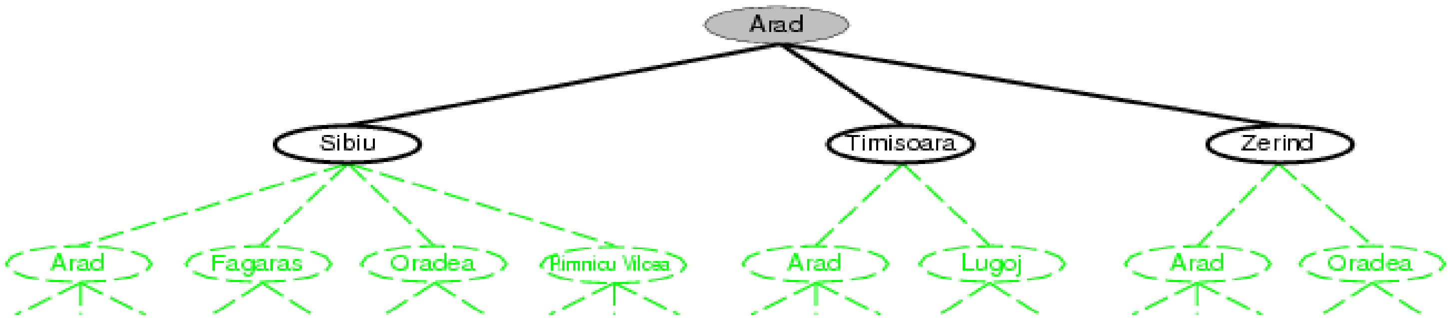
- Initialize the **fringe** using the **starting state**
- While the fringe is not empty
  - Choose a fringe node to expand according to **search strategy**
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the fringe

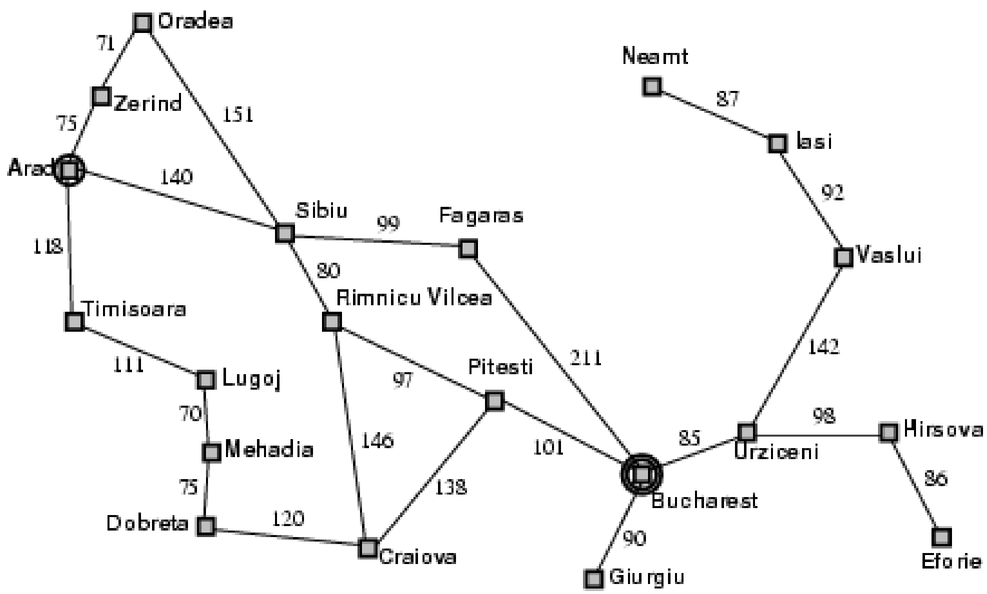
Tree search example



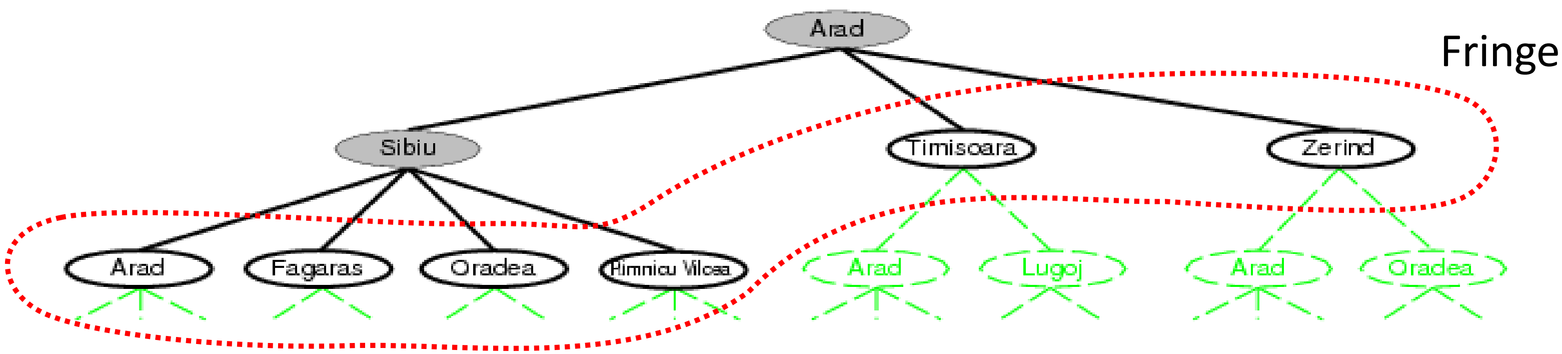


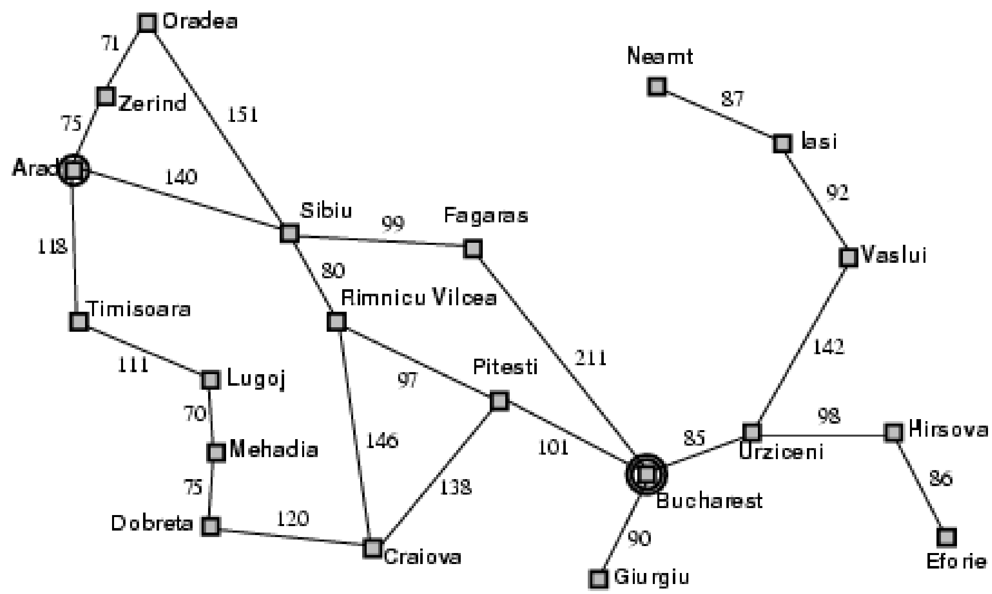
Tree search example





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 solution
  - *m*: maximum length of any path in the state space (may be infinite)

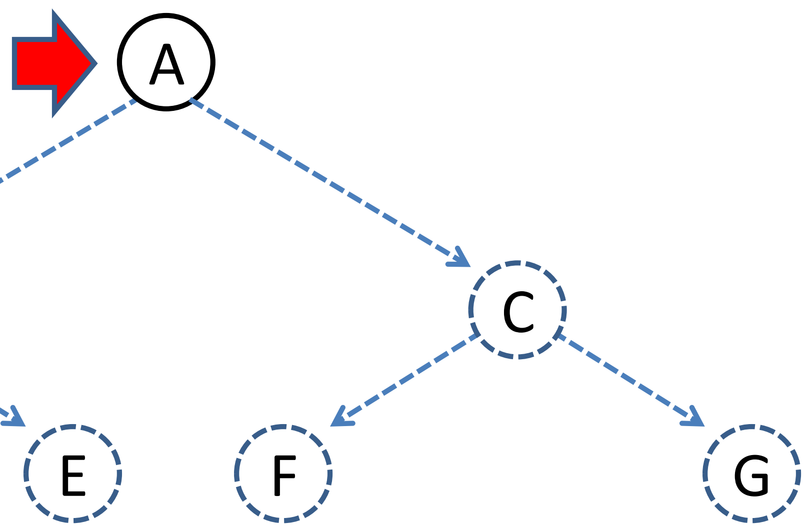
Uninformed search strategies

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

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

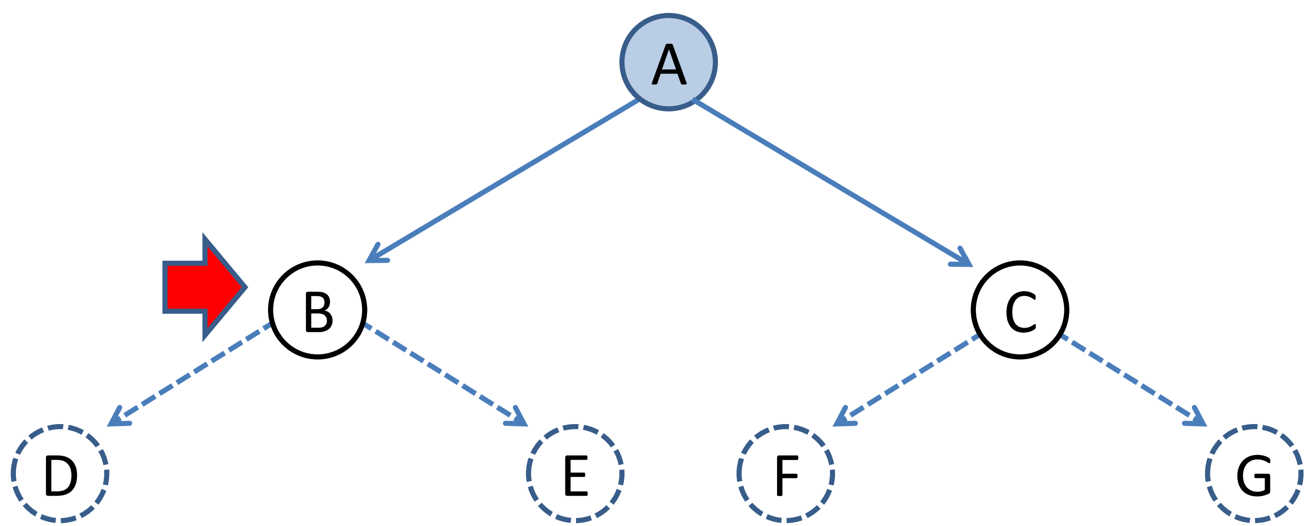
Breadth-first search

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



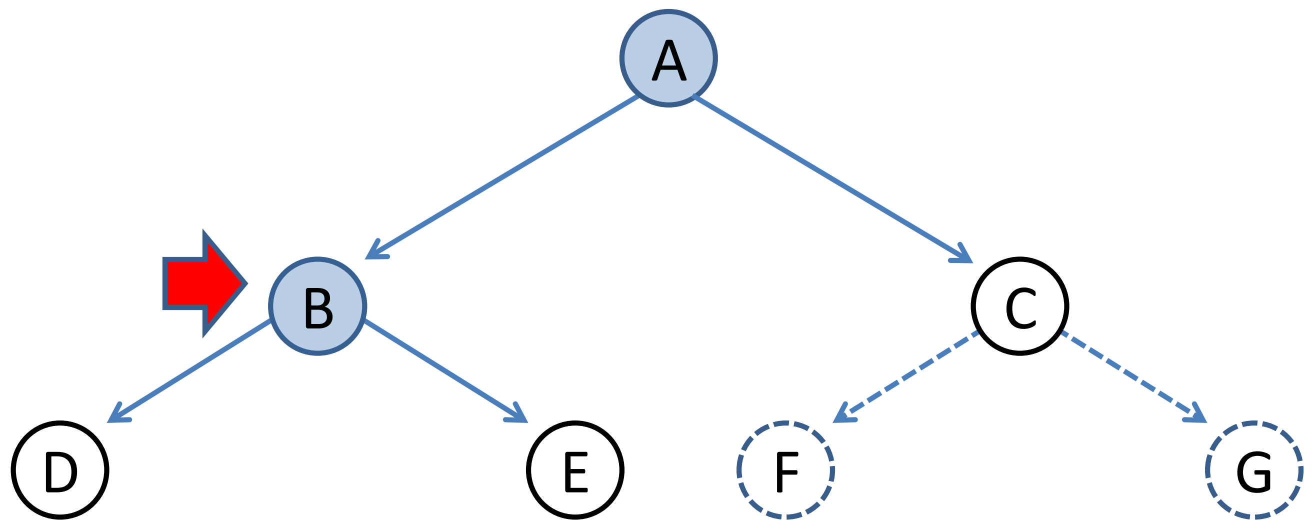
Breadth-first search

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



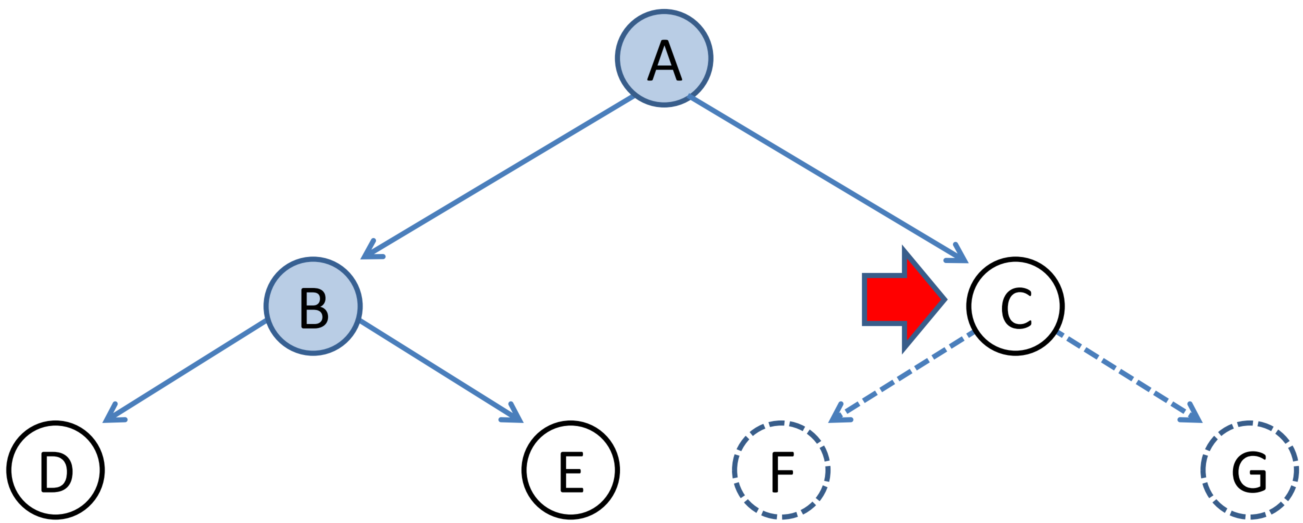
Breadth-first search

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



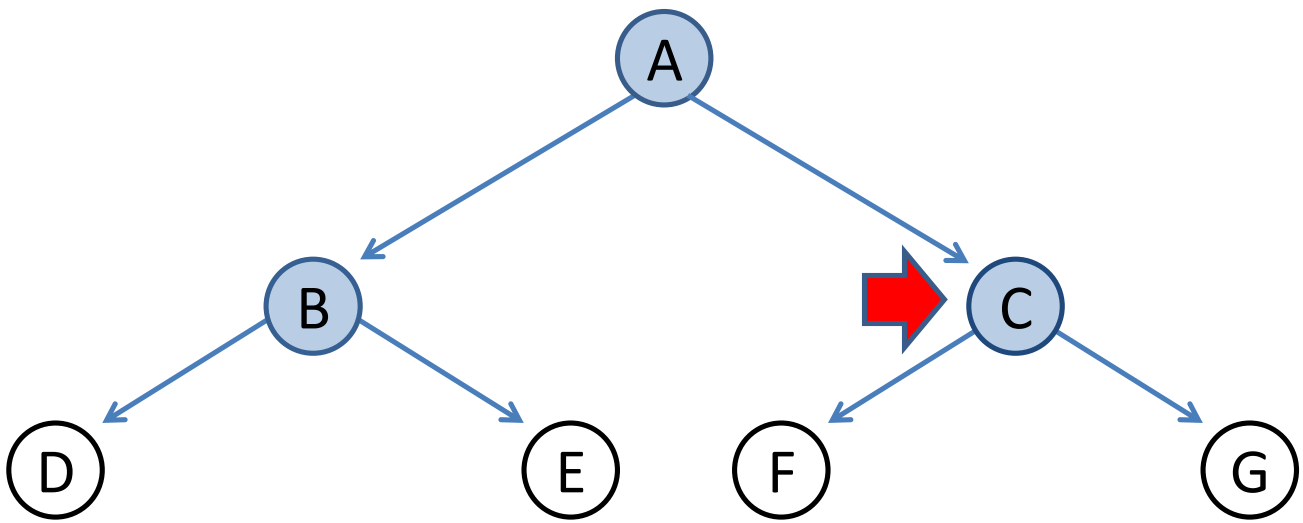
Breadth-first search

- 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 (if branching factor *b* is finite)
- **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)

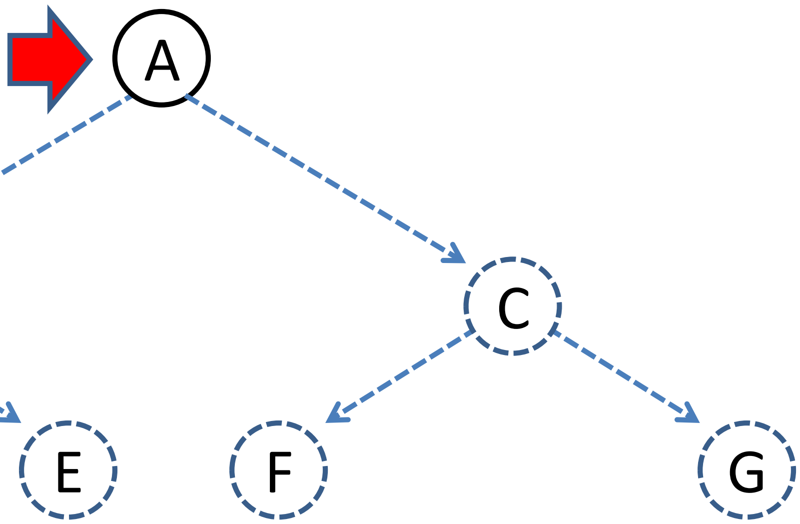
Uniform-cost search

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

- **Complete?**
  Yes, if step cost is greater than some positive constant ε
- **Optimal?**
  Yes – nodes expanded in increasing order of path cost
- **Time?**
  Number of nodes with path cost ≤ cost of optimal solution (C^*), $O(b^{C^*/\varepsilon})$
  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?**
  $O(b^{C^*/\varepsilon})$

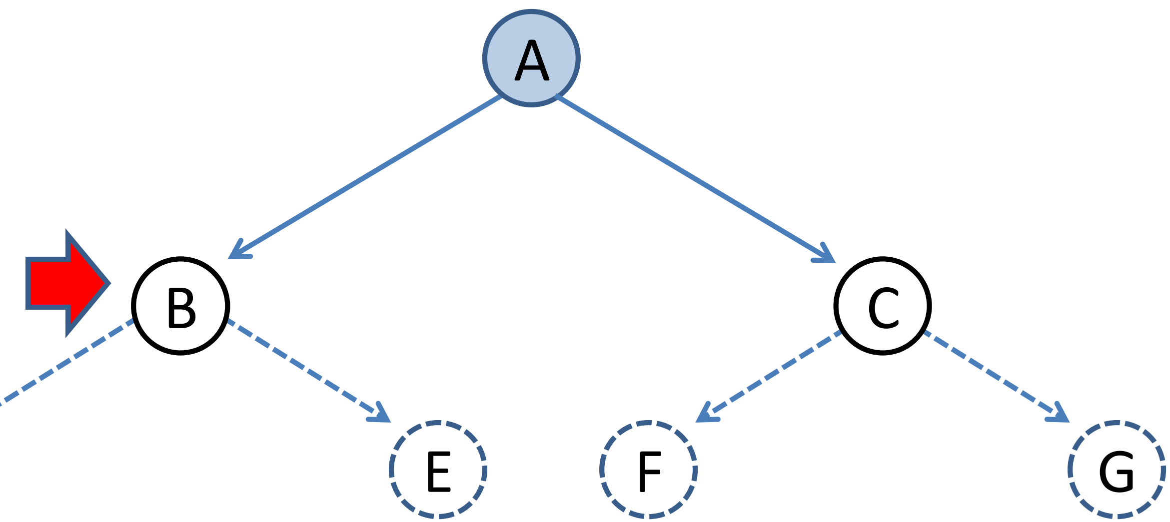
Depth-first search

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



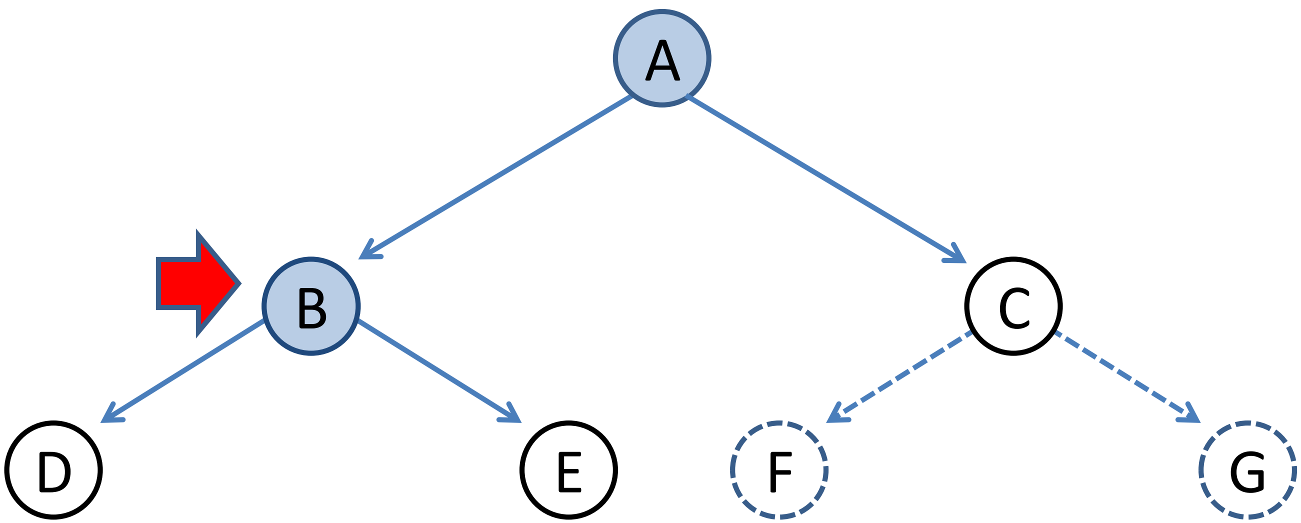
Depth-first search

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



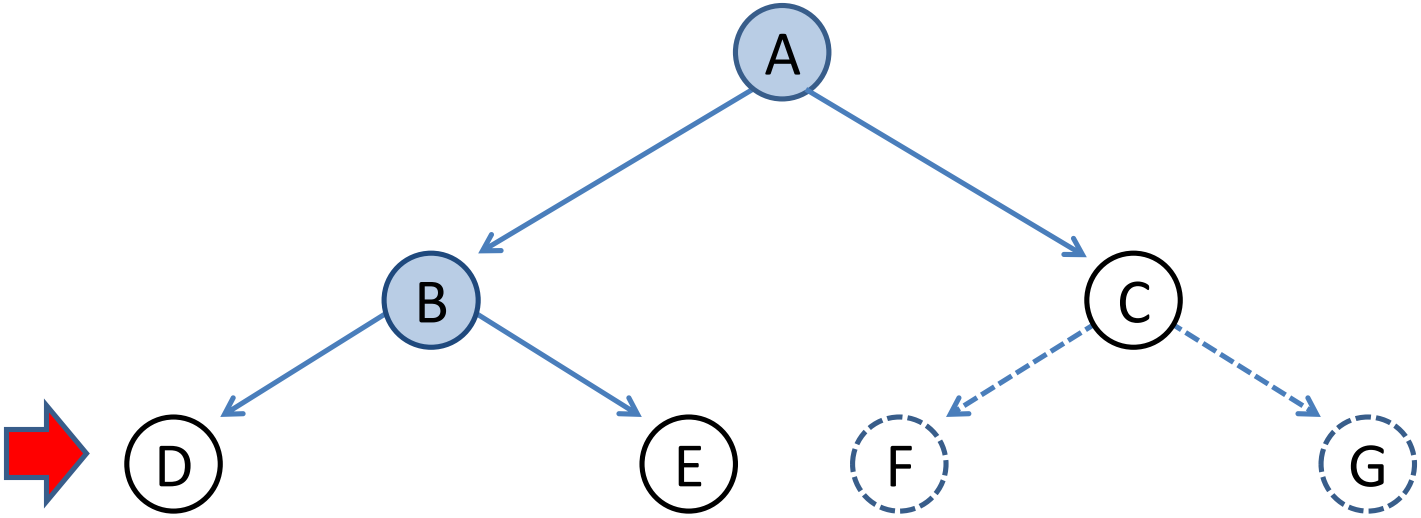
Depth-first search

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



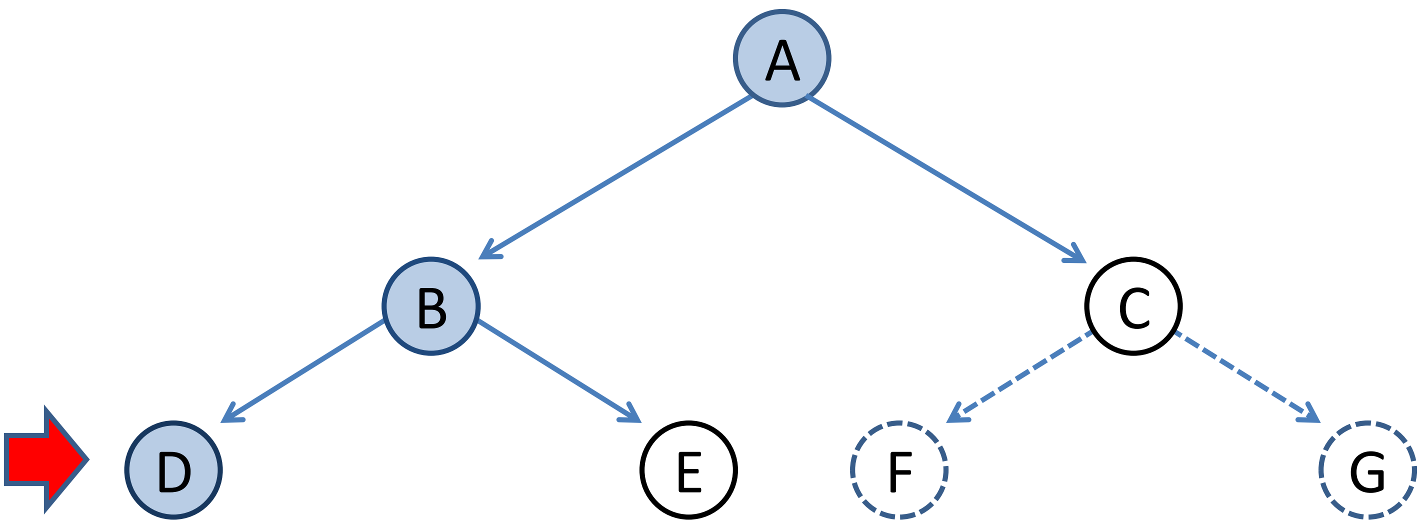
Depth-first search

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



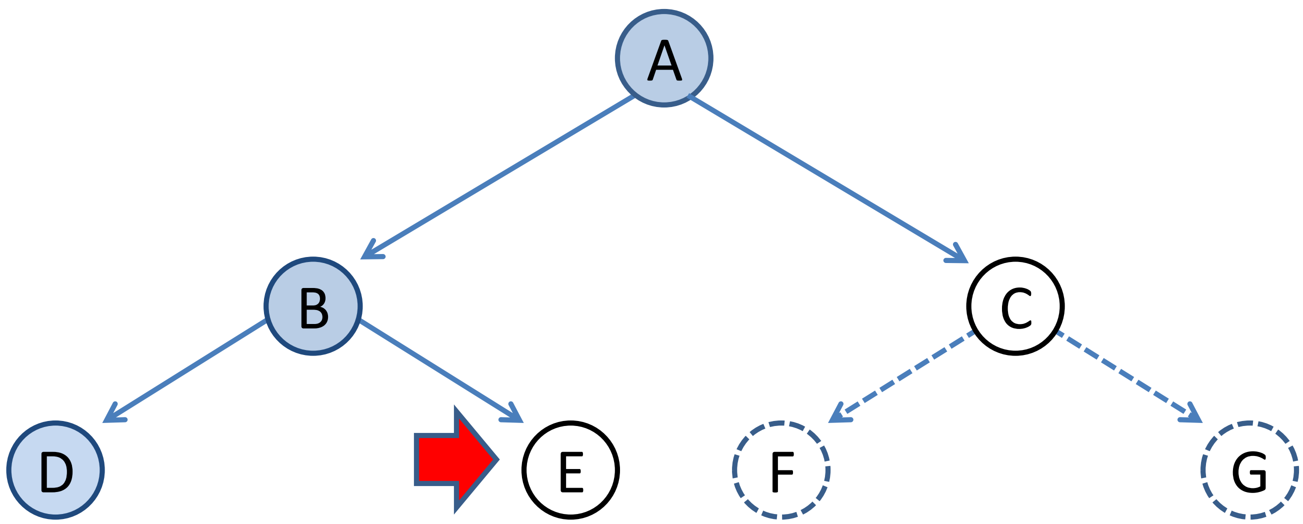
Depth-first search

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



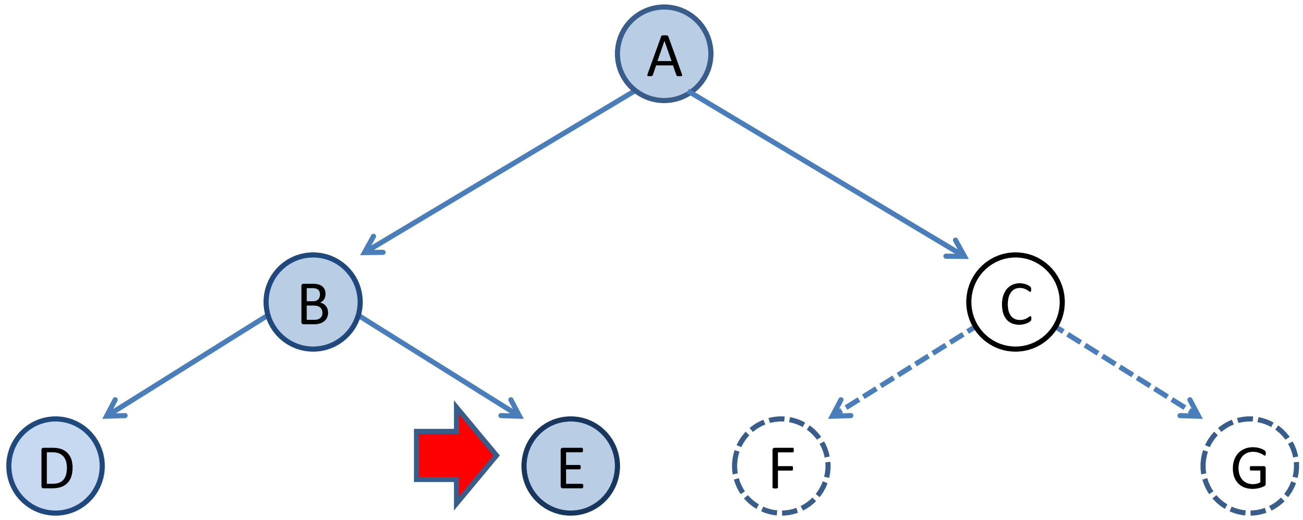
Depth-first search

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



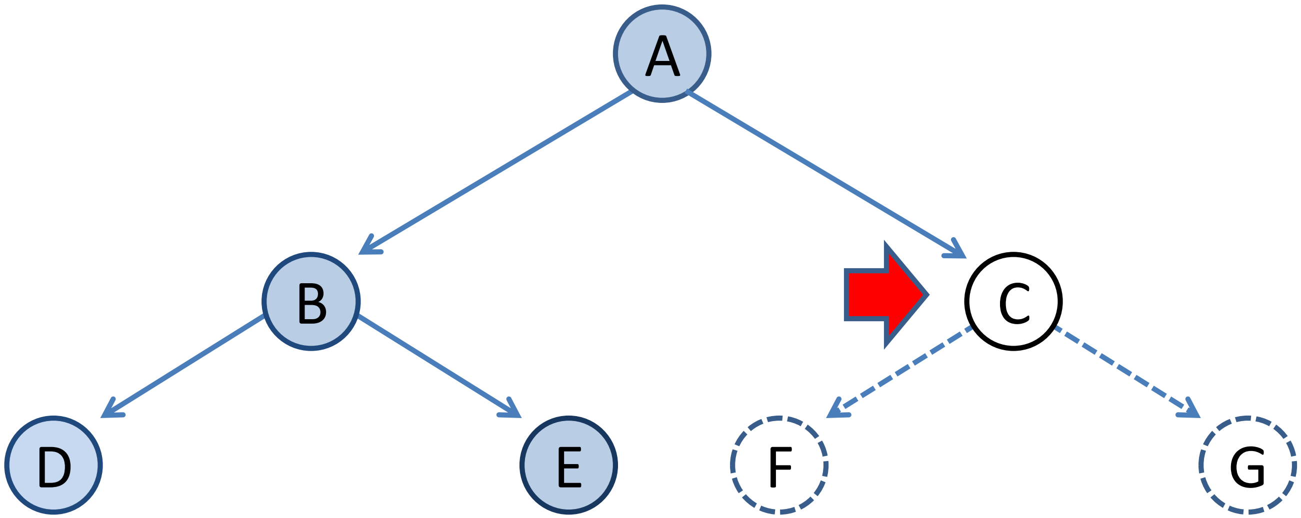
Depth-first search

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



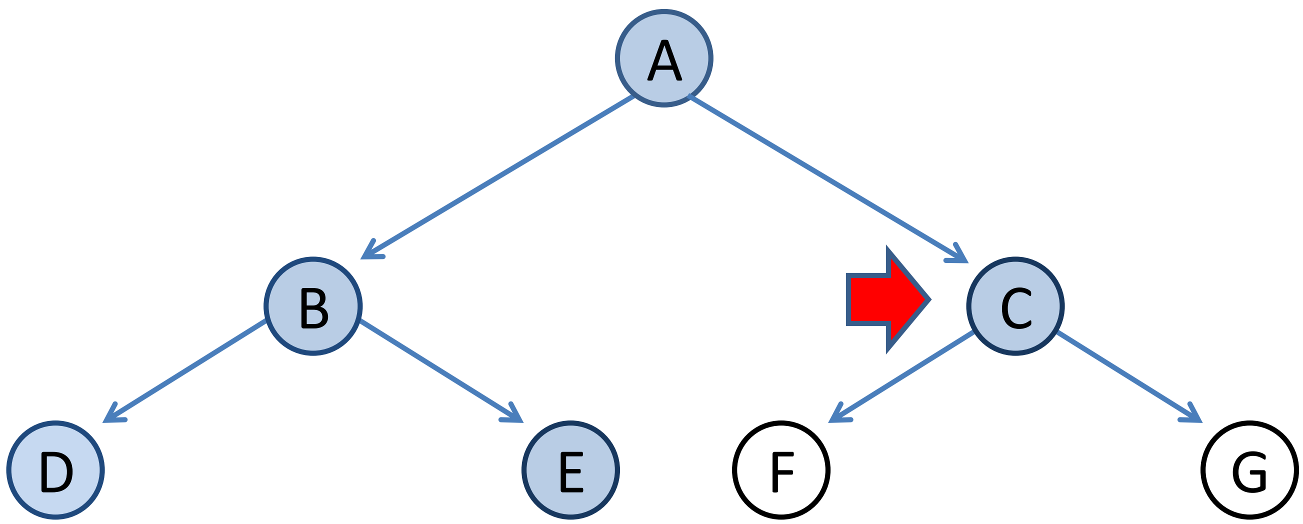
Depth-first search

- 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!

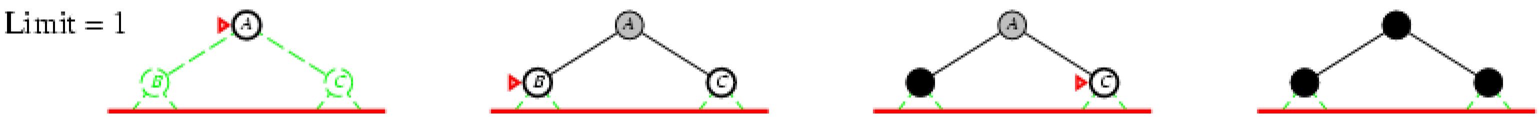
Iterative deepening search

- 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…

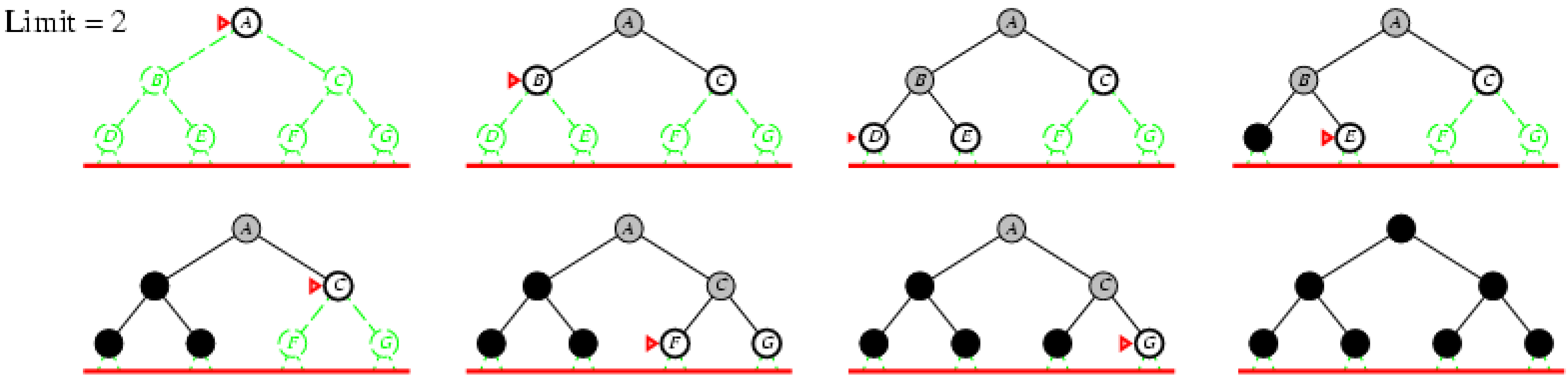
Iterative deepening search

Limit = 0

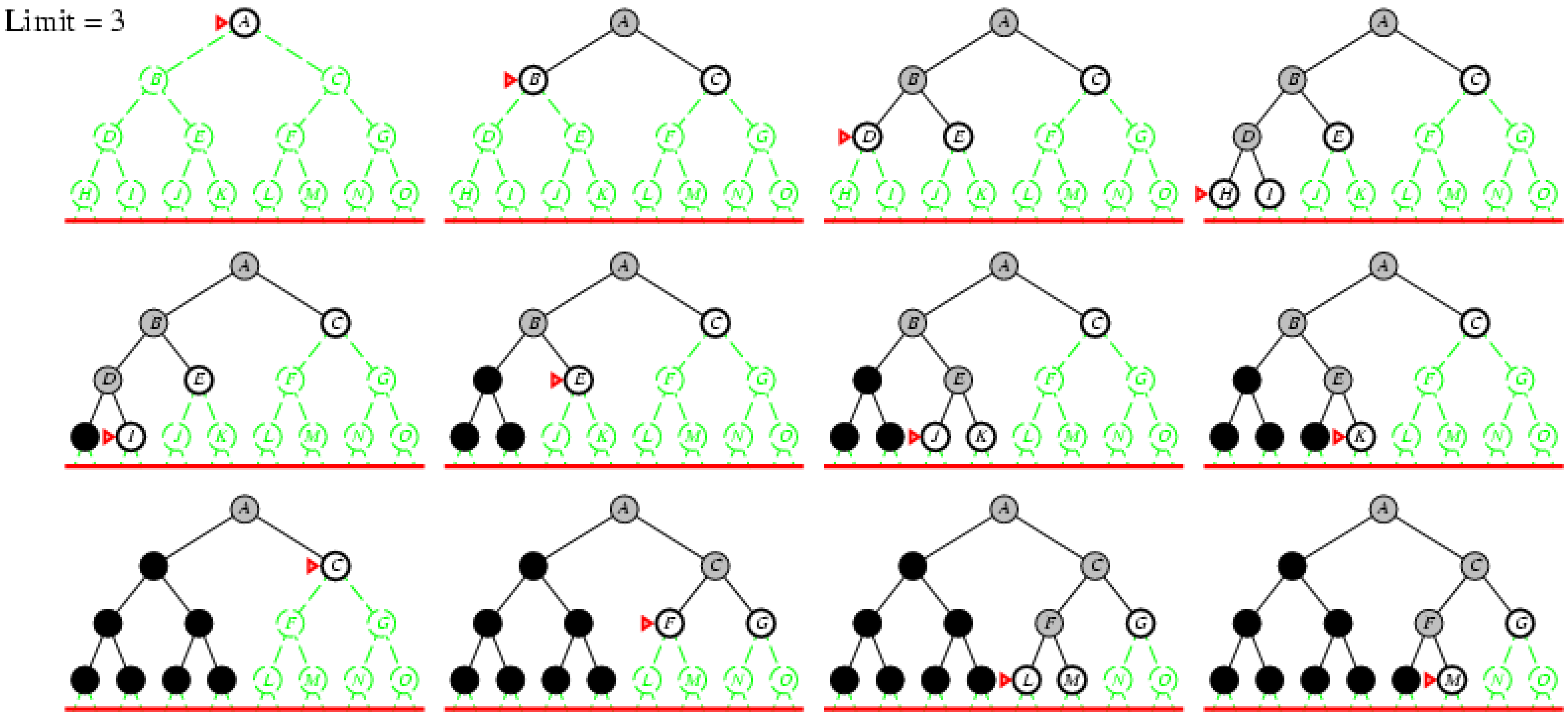
Iterative deepening search



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?**

  $O(bd)$

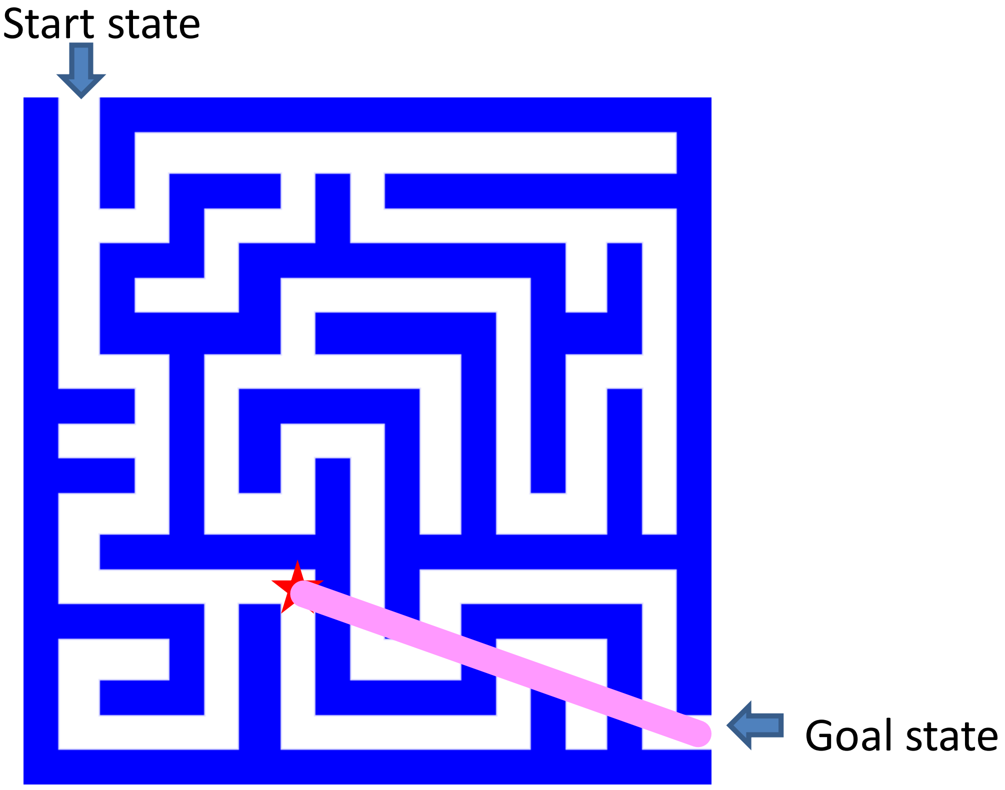
Informed search

- Idea: give the algorithm "hints" about the desirability of different states
  - Use an *evaluation function* to rank nodes and select the most promising one for expansion

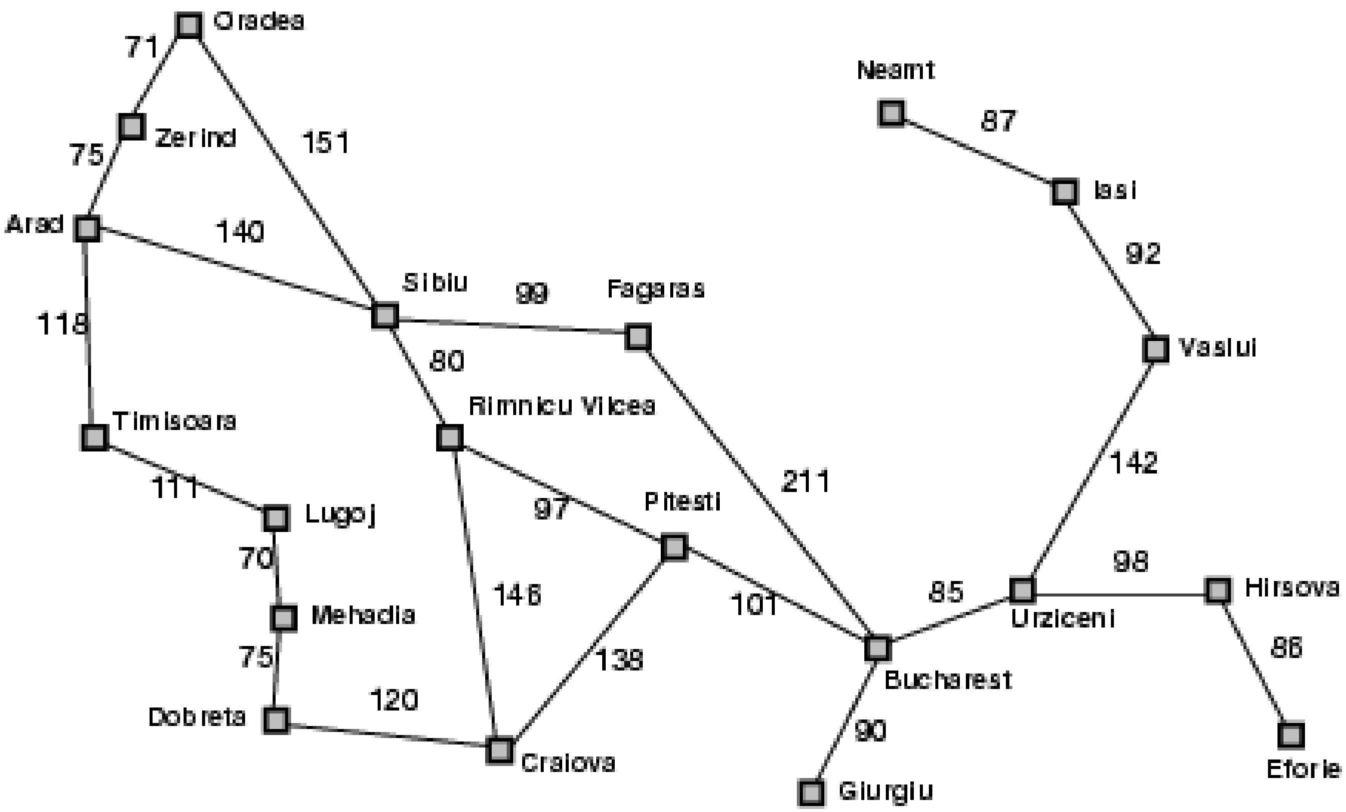
- Greedy best-first search
- A* search

Heuristic function

- Heuristic function $h(n)$ estimates the cost of reaching goal from node n
- Example:



Heuristic for the Romania problem

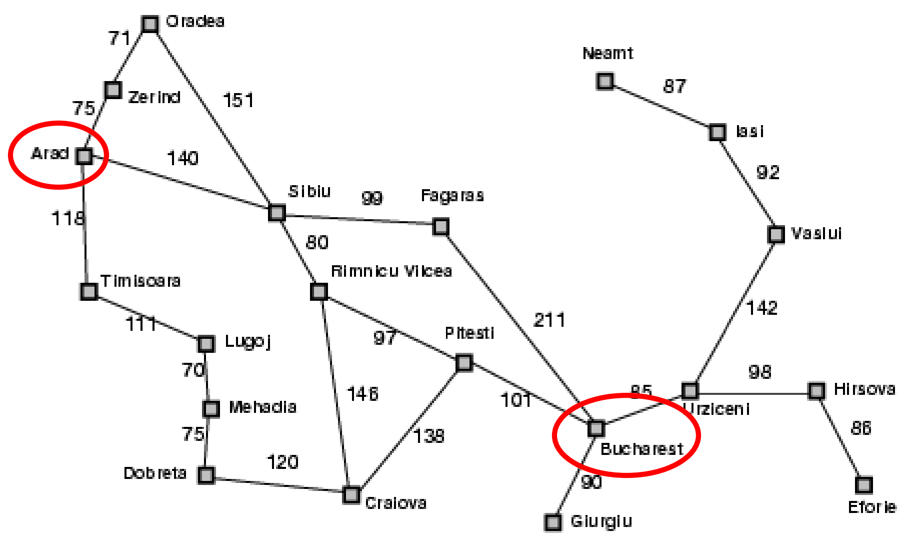


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

- Expand the node that has the lowest value of the heuristic function $h(n)$

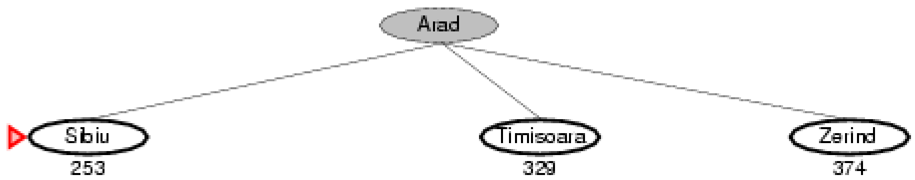
Greedy best-first search example

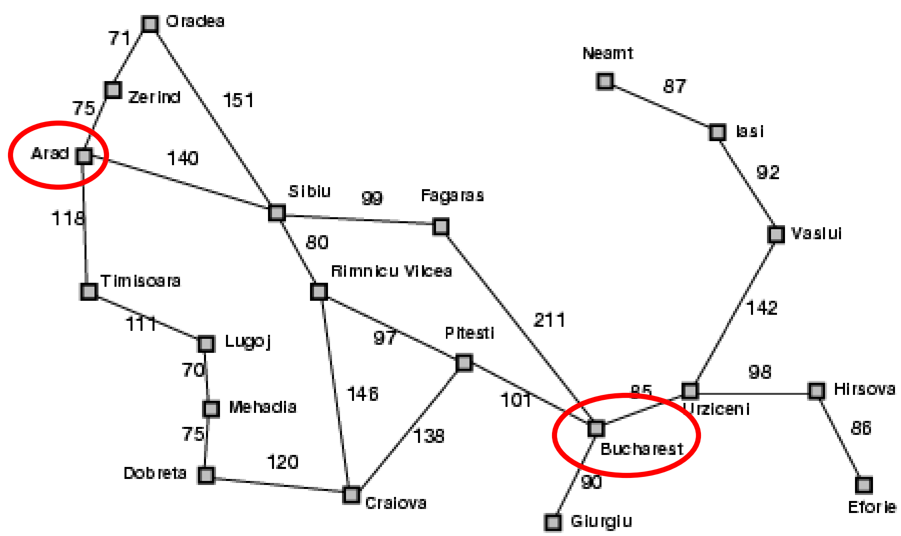


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

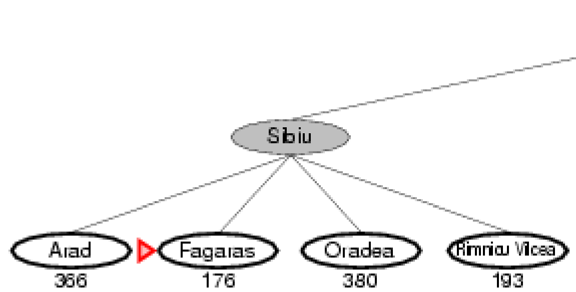


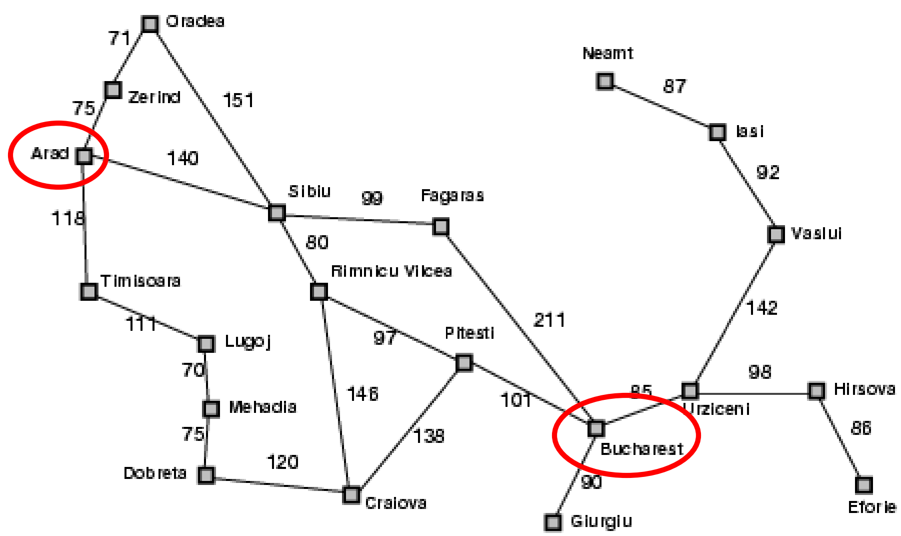


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

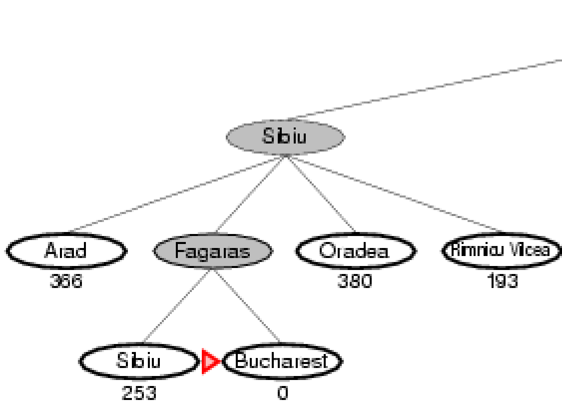


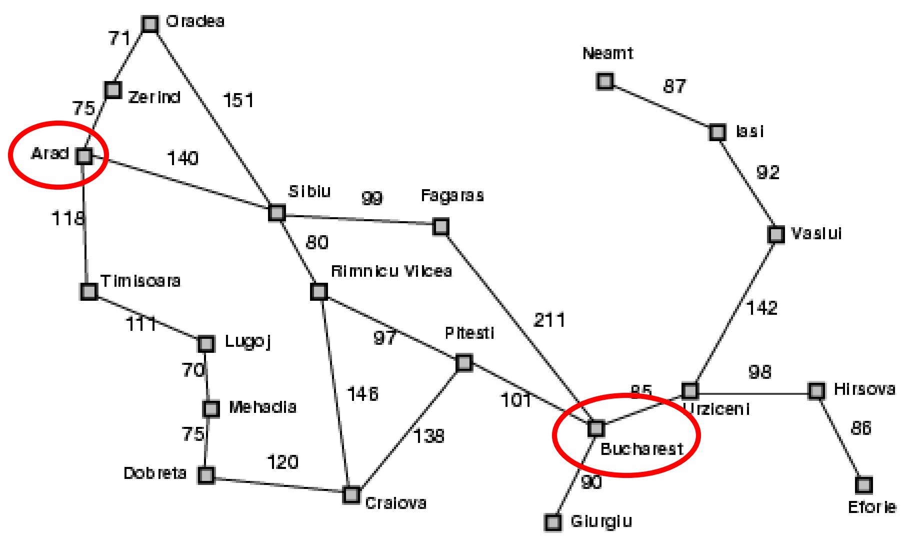


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example





Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops

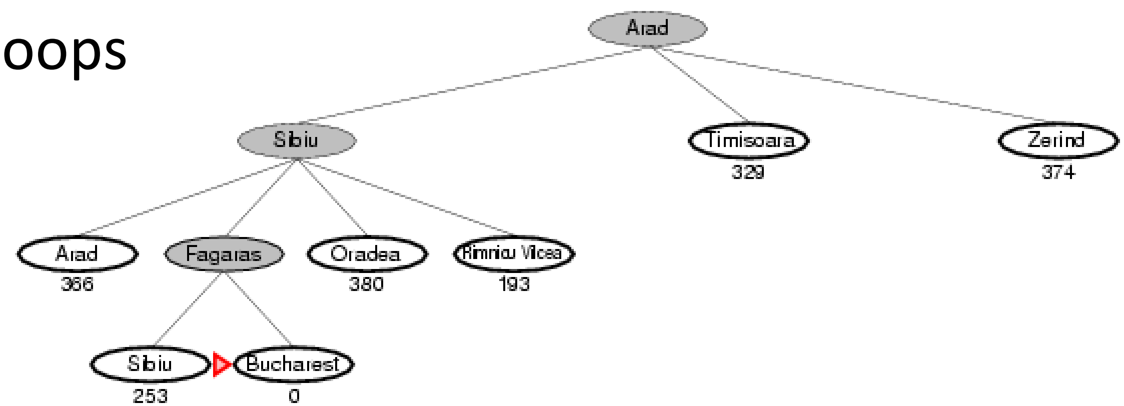
Properties of greedy best-first search

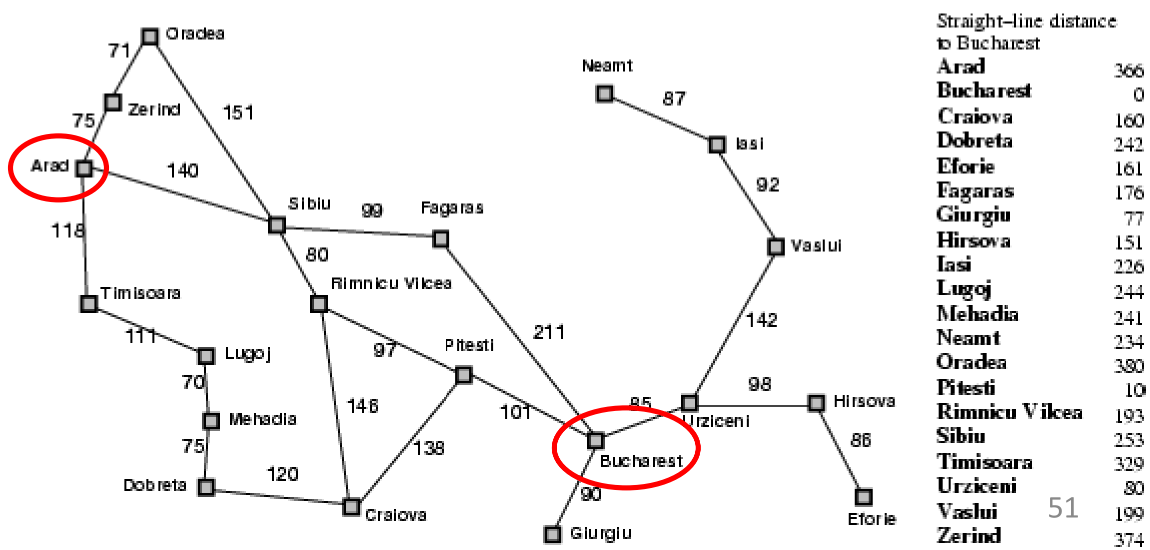
- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No





Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops
- **Optimal?**

  No
- **Time?**

  Worst case: $O(b^m)$

  Best case: $O(bd)$ – If $h(n)$ is 100% accurate
- **Space?**

  Worst case: $O(b^m)$

A* search

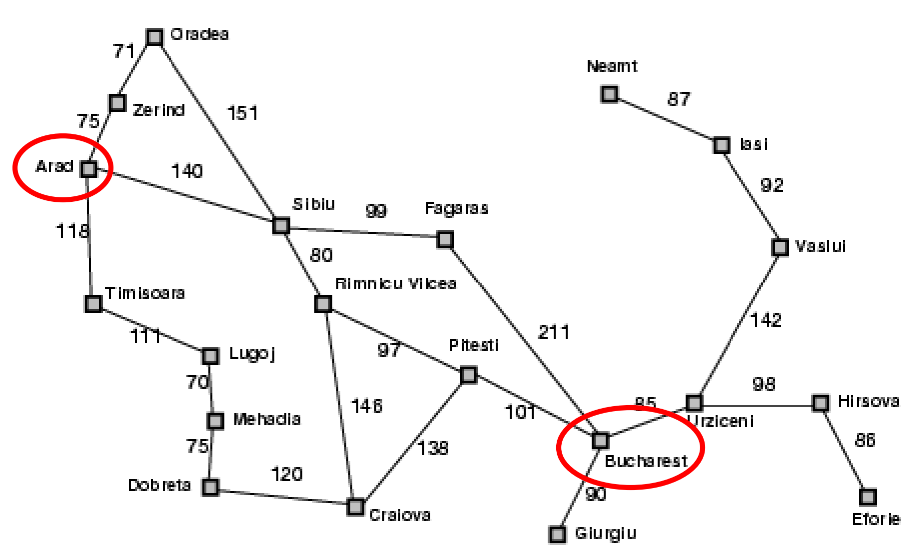
- Idea: avoid expanding paths that are already expensive
- The evaluation function $f(n)$ is the estimated total cost of the path through node n to the goal:

$$f(n) = g(n) + h(n)$$

$g(n)$: cost so far to reach n (path cost)

$h(n)$: estimated cost from n to goal (heuristic)

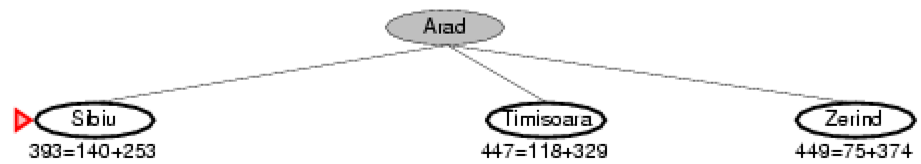
A* search example

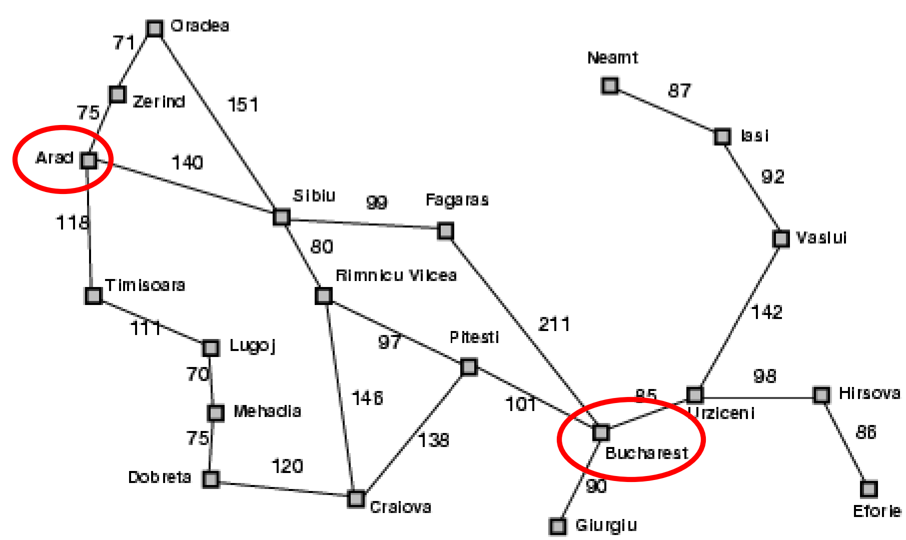


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

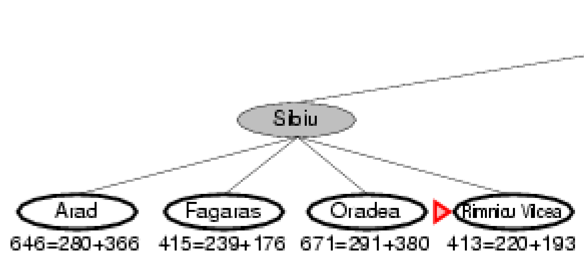


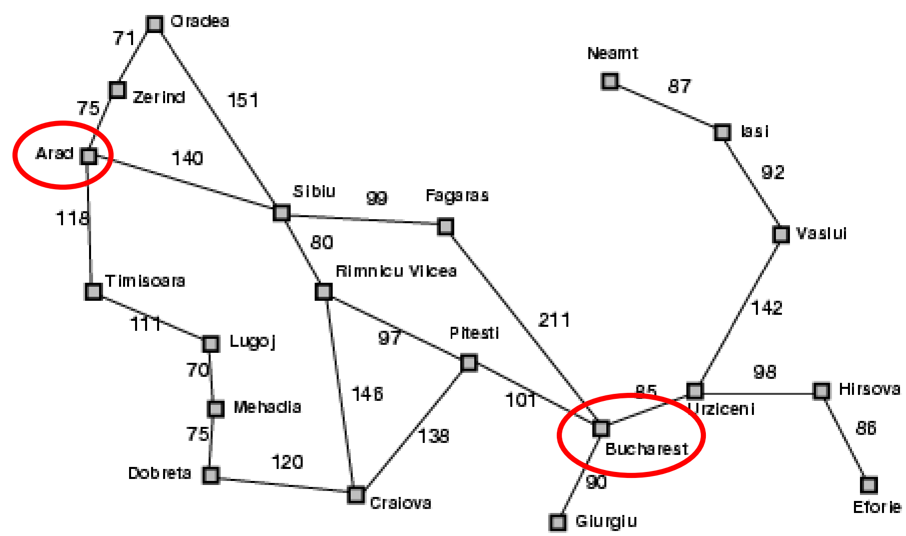


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

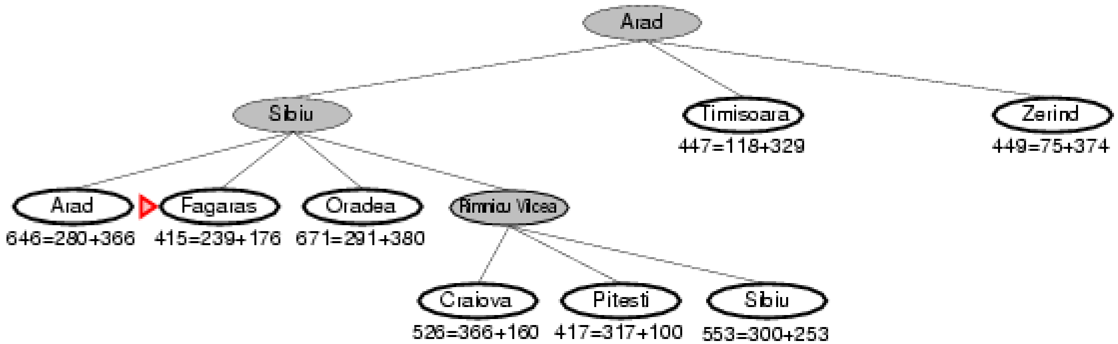


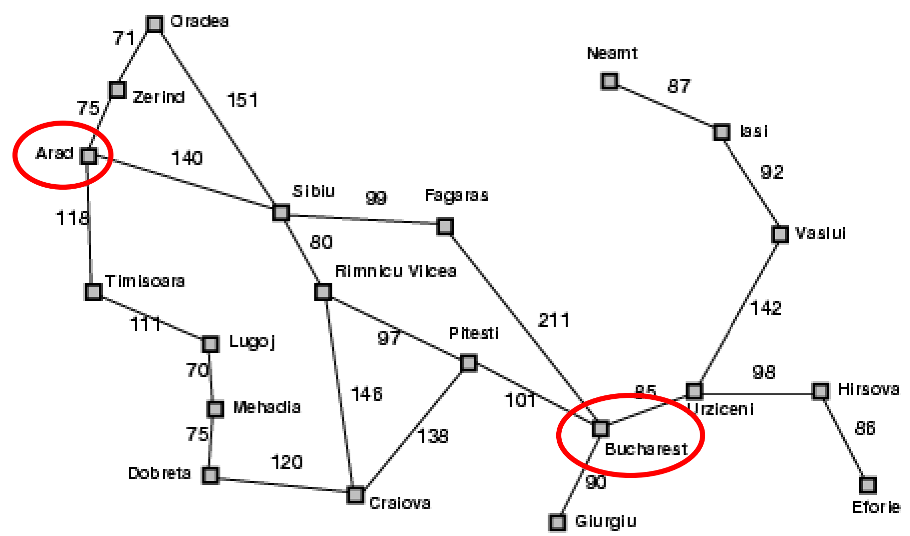


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

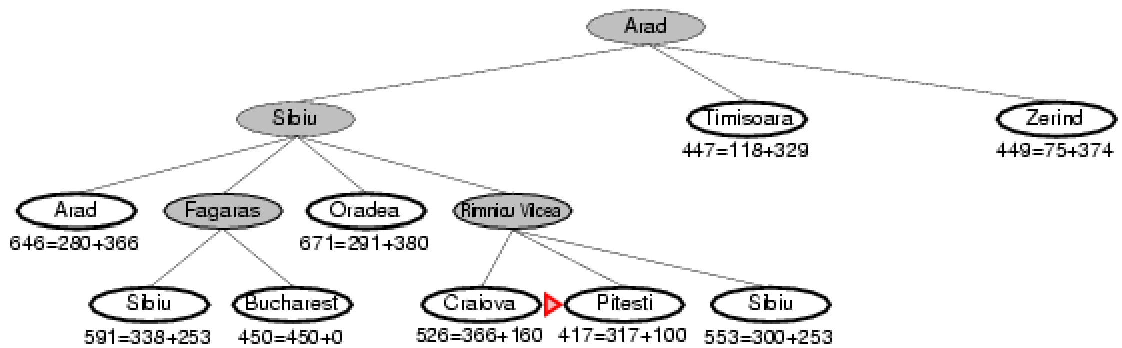
A* search example

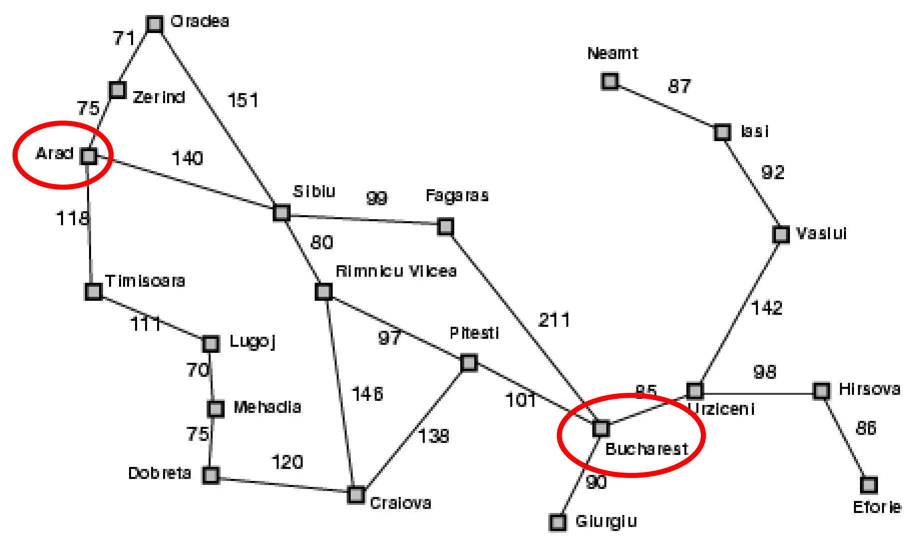




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

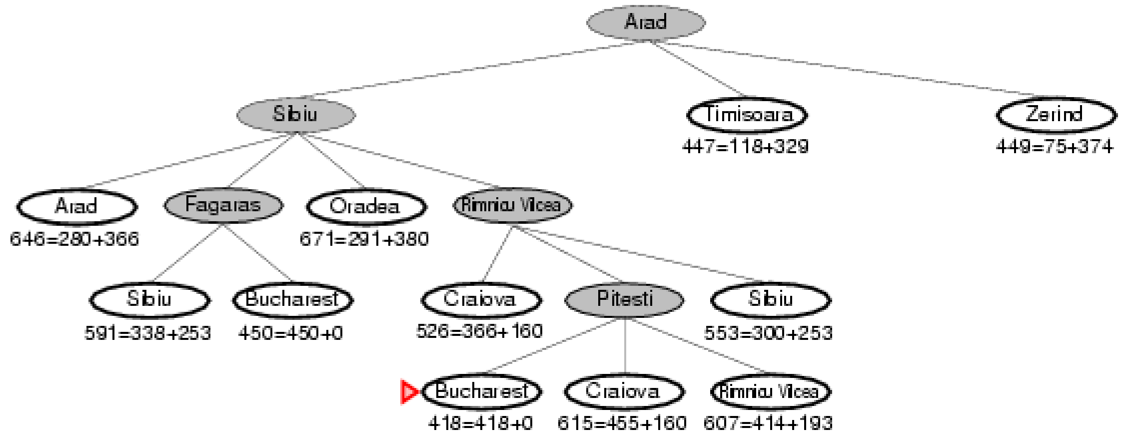


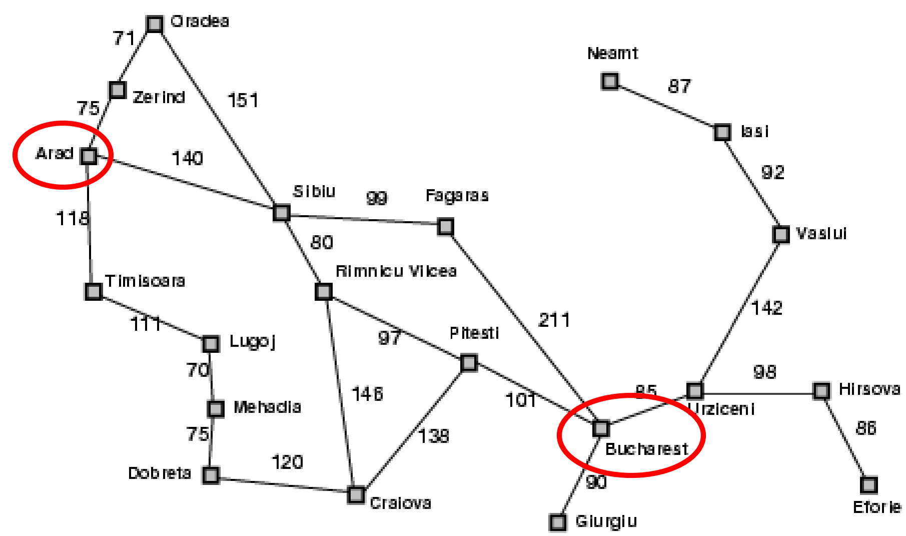


Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example





Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of A*

- **Complete?**

  Yes – unless there are infinitely many nodes with $f(n) \leq C^*$

- **Optimal?**

  Yes

- **Time?**

  Number of nodes for which $f(n) \leq C^*$ (exponential)

- **Space?**

  Exponential

Admissible heuristics

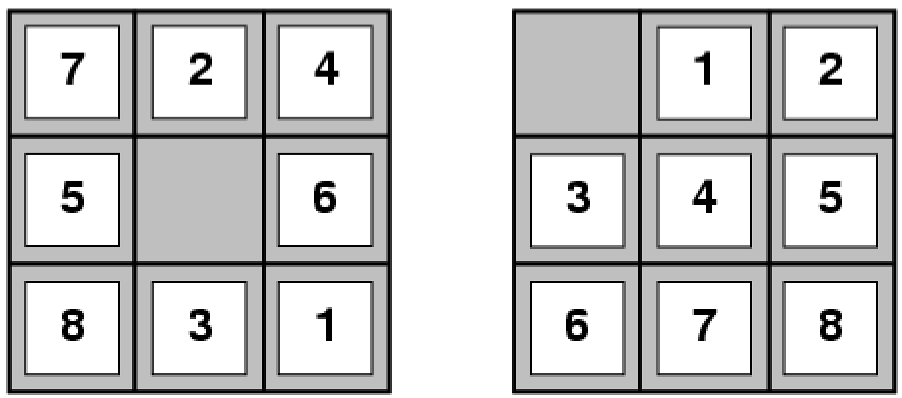
- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: straight line distance never overestimates the actual road distance
- Theorem: If $h(n)$ is admissible, A^* is optimal

Designing heuristic functions

- Heuristics for the 8-puzzle

  $h_1(n)$ = number of misplaced tiles

  $h_2(n)$ = total Manhattan distance (number of squares from desired location of each tile)



Start State Goal State

$$h_1(\text{start}) = 8$$

$$h_2(\text{start}) = 3+1+2+2+2+3+3+2 = 18$$

- Are h_1 and h_2 admissible?

Comparison of search strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
UCS	Yes	Yes	Number of nodes with $g(n) \le C^*$	
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
Greedy	No	No	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes	Number of nodes with $g(n)+h(n) \le C^*$	