Outline: Problem solving and search (Uninformed Search – AIMA Ch. 3]

- **Introduction to Problem Solving**
- **Complexity**
- **Uninformed search**
  - Problem formulation
  - Search strategies: depth-first, breadth-first
- **Informed search**
  - Search strategies: best-first, A*
  - Heuristic functions

Slides taken from AI class based on Russell&Norvig:

Example: Romania

- In Romania, on vacation. Currently in Arad.
- Flight leaves tomorrow from Bucharest.
- **Formulate goal:**
  - be in Bucharest
- **Formulate problem:**
  - states: various cities
  - operators: drive between cities
- **Find solution:**
  - sequence of cities, such that total driving distance is minimized.

Problem formulation (cont'd)

A problem is defined by four items:

initial state e.g., "at Arad"

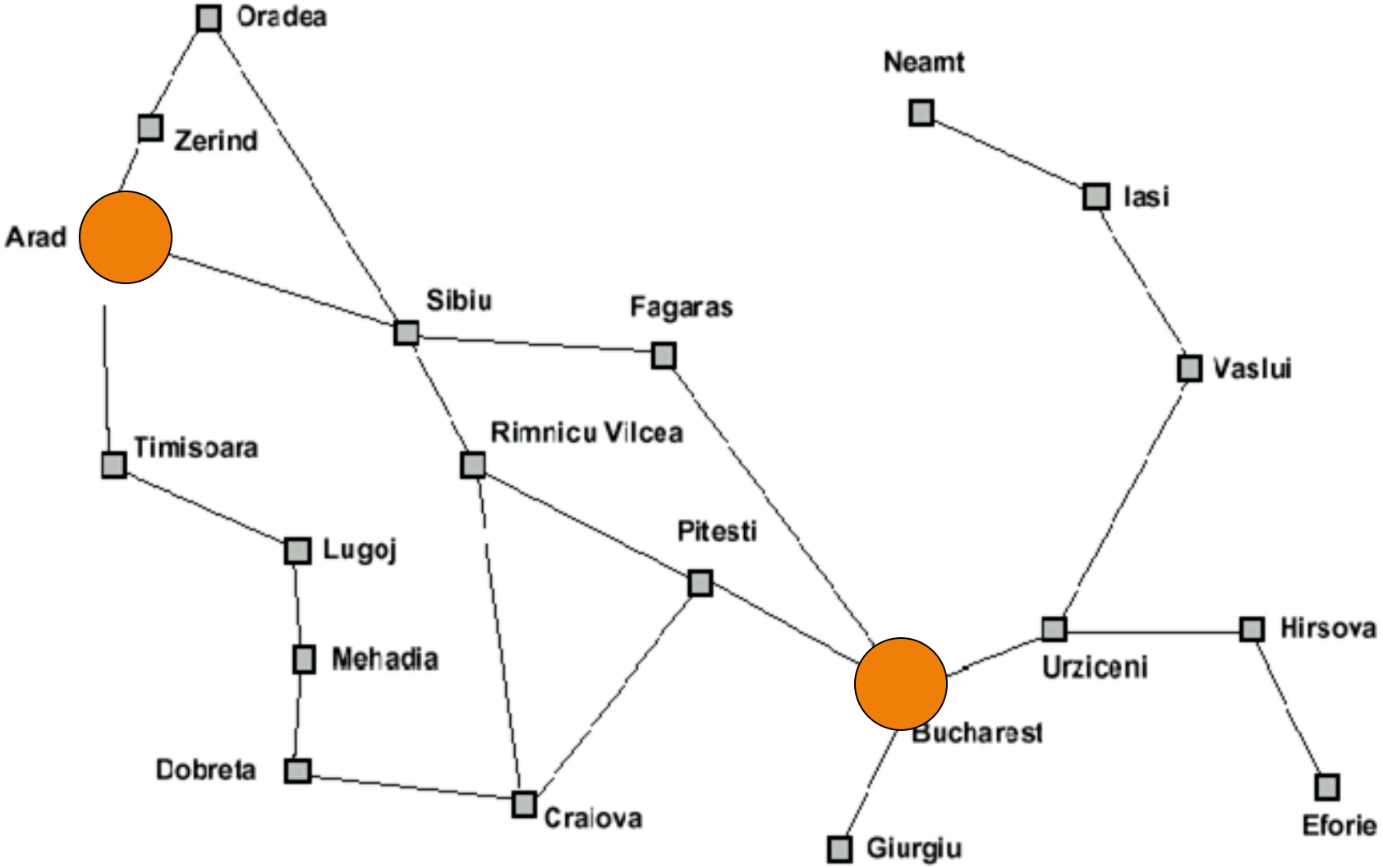
successor function $S(x)$ = set of action–state pairs
e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \ldots\}$

goal test, can be
explicit, e.g., $x =$ "at Bucharest"
implicit, e.g., $NoDirt(x)$

path cost (additive)
e.g., sum of distances, number of actions executed, etc.
$c(x, a, y)$ is the step cost, assumed to be ≥ 0

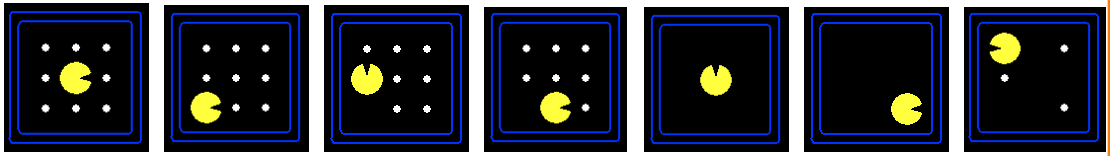
A solution is a sequence of actions
leading from the initial state to a goal state

Example: Traveling from Arad To Bucharest

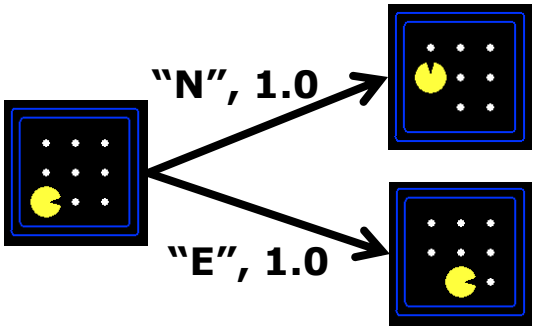


Problem formulation

- A search problem consists of:
  - A state space



  - A successor function



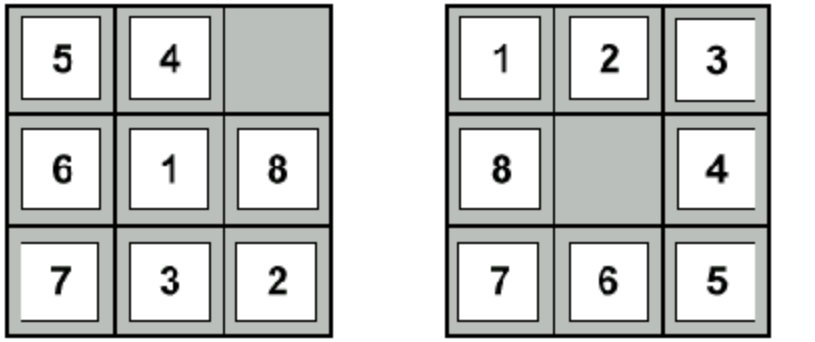
  - A start state and a goal test

- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Selecting a state space

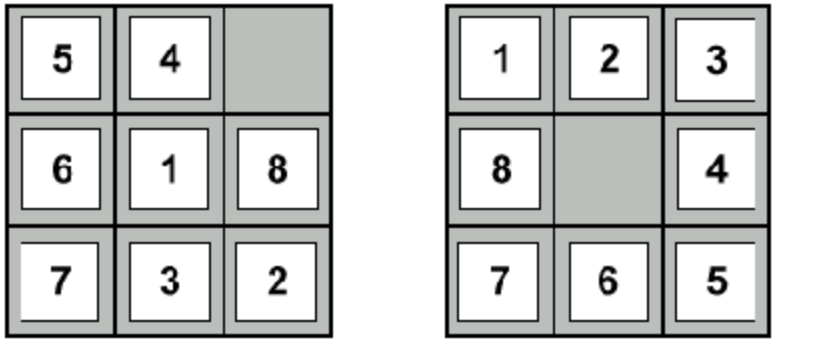
- Real world is absurdly complex; some abstraction is necessary to allow us to reason on it…
- Selecting the correct abstraction and resulting state space is a difficult problem!
- Abstract states ⇔ real-world states
- Abstract operators ⇔ sequences or real-world actions
  (e.g., going from city i to city j costs Lij ⇔ actually drive from city i to j)
- Abstract solution ⇔ set of real actions to take in the real world such as to solve problem

Example: 8-puzzle



- State:
- Operators:
- Goal test:
- Path cost:

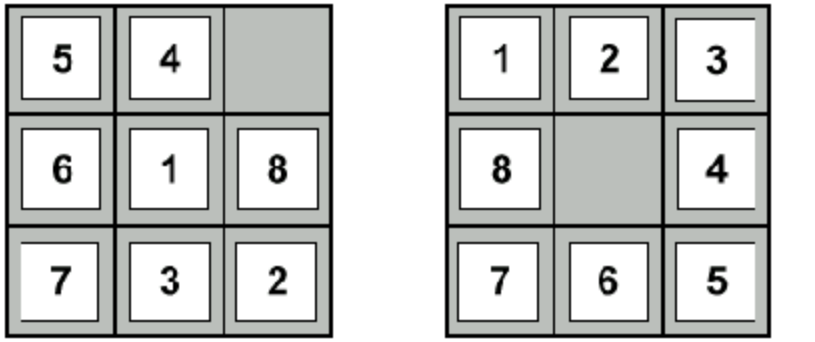
Example: 8-puzzle



start state goal state

- State: integer location of tiles (ignore intermediate locations)
- Operators: moving blank left, right, up, down (ignore jamming)
- Goal test: does state match goal state?
- Path cost: 1 per move

Example: 8-puzzle



start state goal state

Why search algorithms?

- 8-puzzle has 362,880 states
- 15-puzzle has 10^12 states
- 24-puzzle has 10^25 states

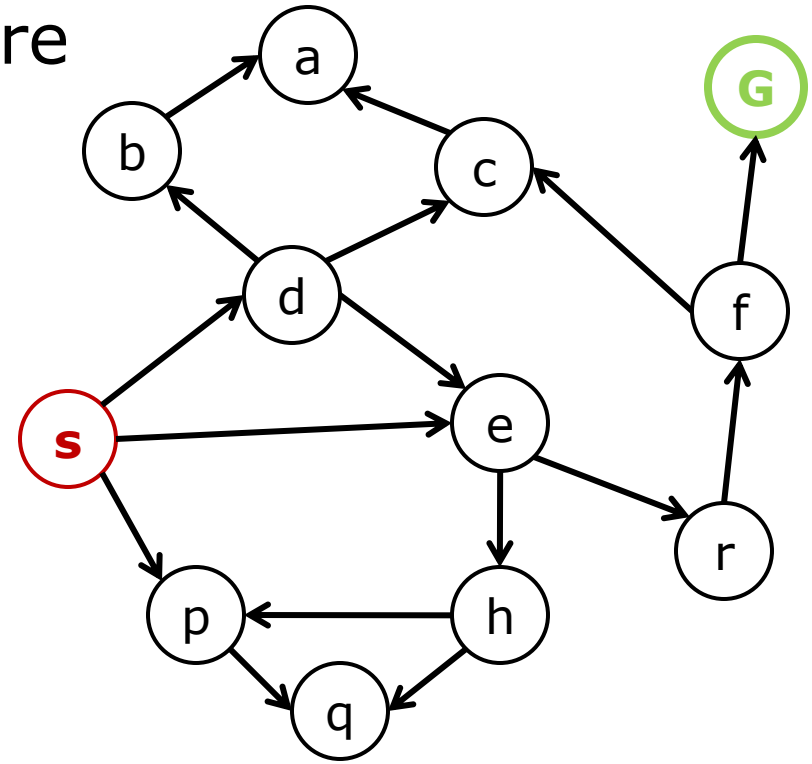
So, we need a principled way to look for a solution in these huge search spaces…

State Space Graphs

- There's a big graph where
  - Each state is a node
  - Each successor is an outgoing arc

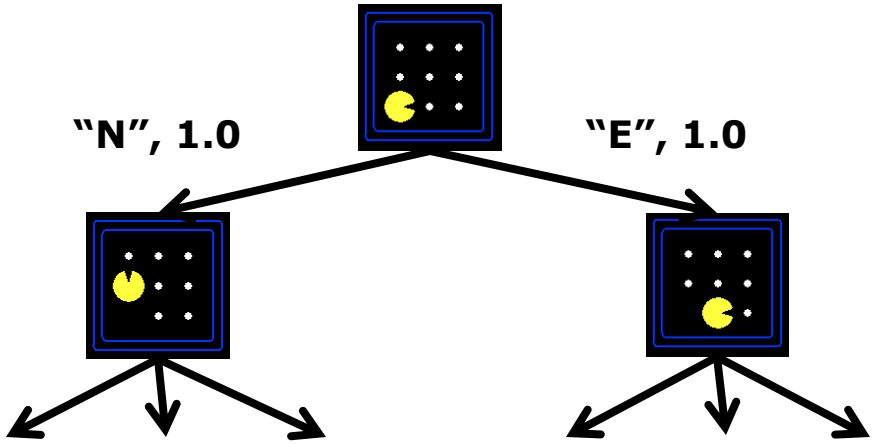
- Important: For most problems we could never actually build this graph

- How many states in Pacman?



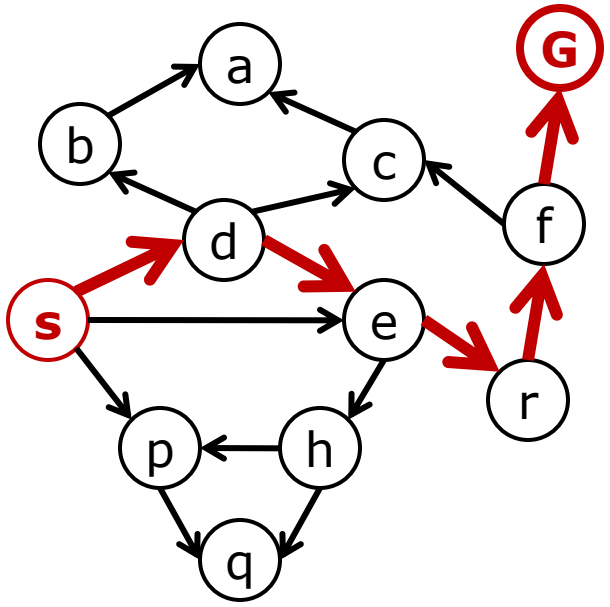
Laughably tiny search graph for a tiny search problem

Search Trees



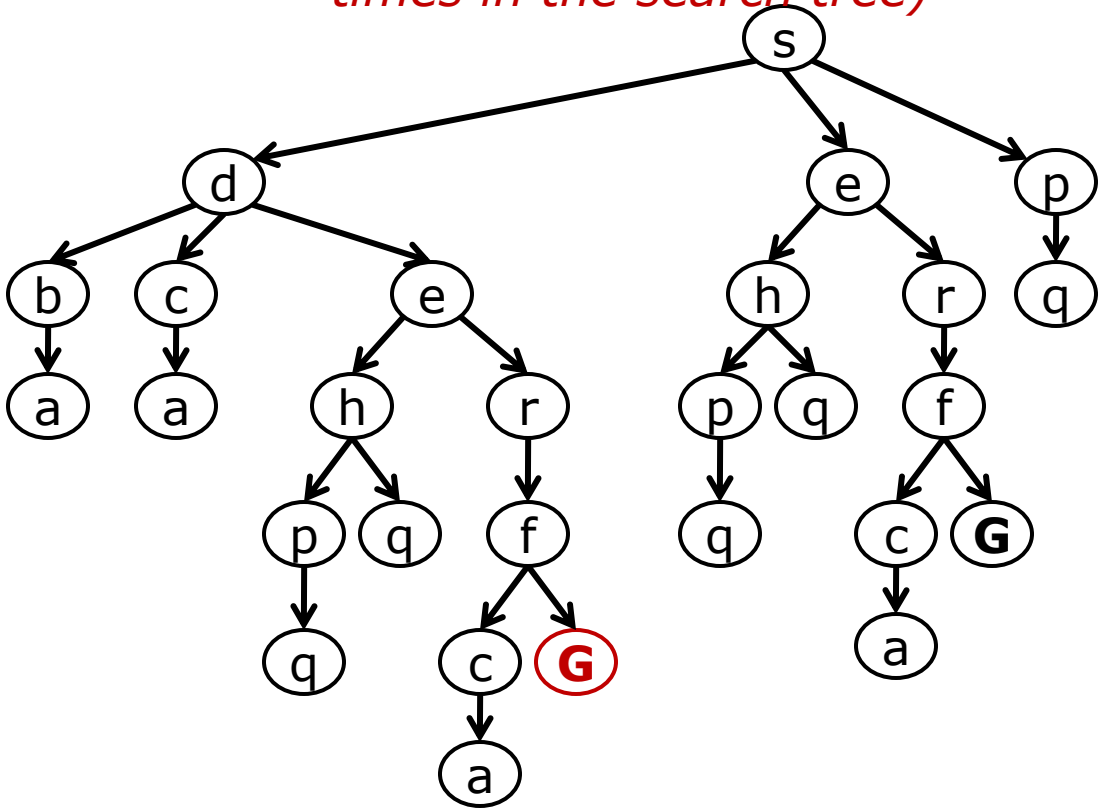
- A search tree:
  - This is a "what if" tree of plans and outcomes
  - Start state at the root node
  - Children correspond to successors
  - Nodes labeled with states, correspond to PLANS to those states
  - For most problems, can never build the whole tree
    - So, have to find ways to use only the important parts!

State Graphs vs. Search Trees



We almost always construct both on demand – and we construct as little as possible

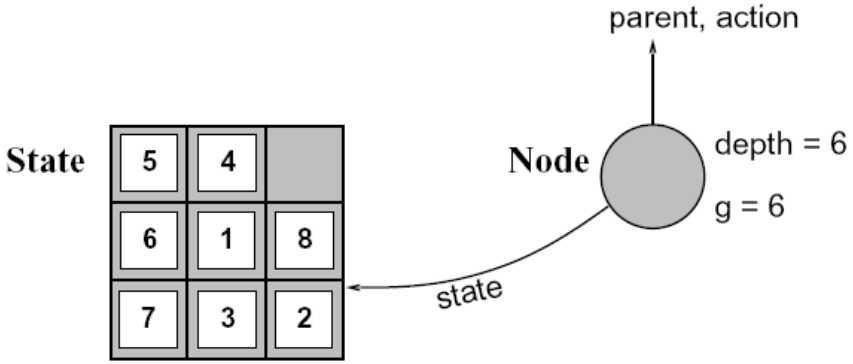
Each NODE in the search tree is an entire PATH in the state graph (note how many nodes in the graph appear multiple times in the search tree)



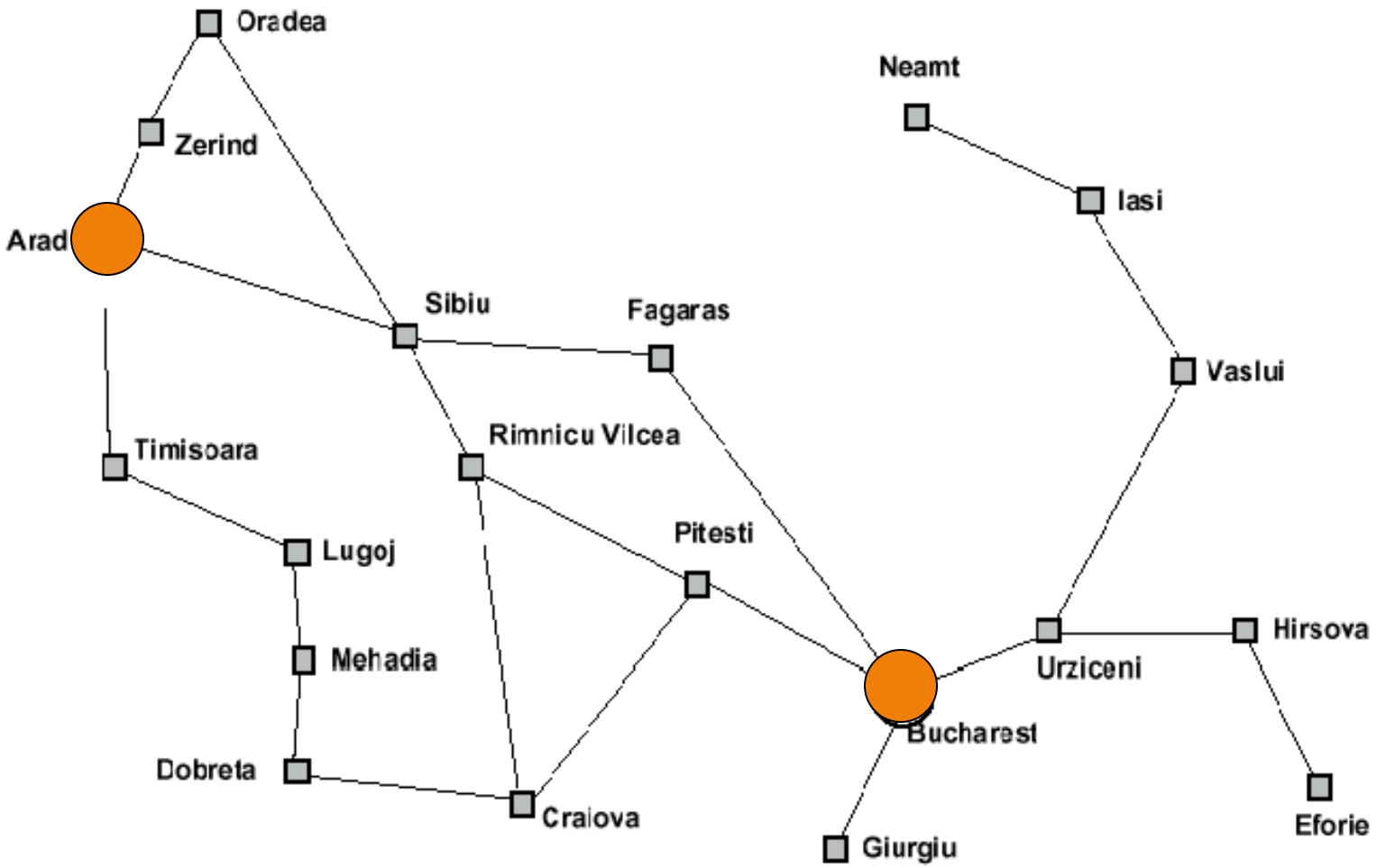
States vs. Nodes

- Problem graphs have problem states
  - Represent an abstracted state of the world
  - Have successors, predecessors, can be goal / non-goal

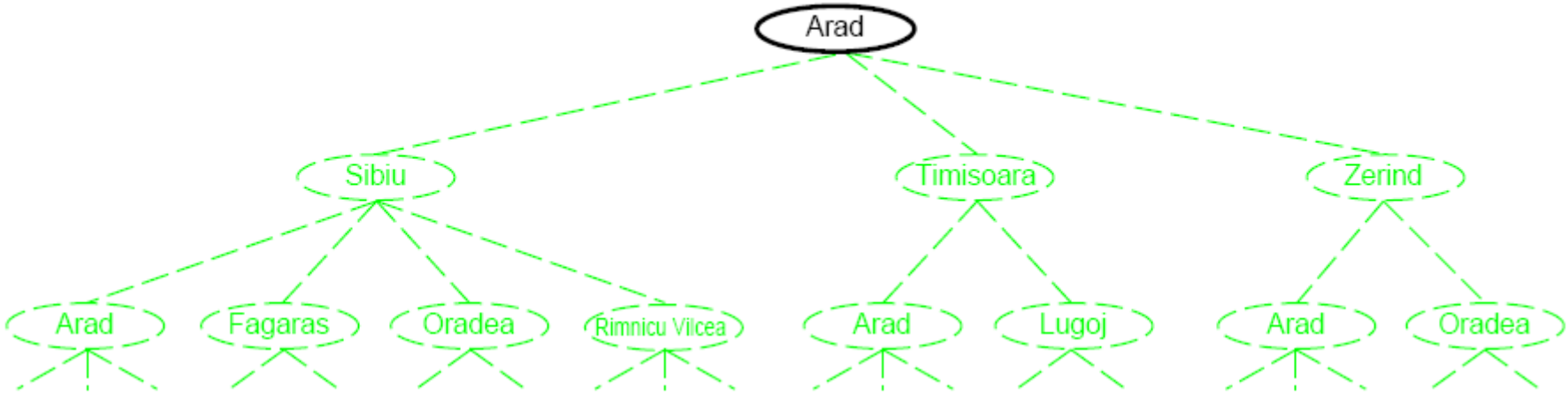
- Search trees have search nodes
  - Represent a plan (path) which results in the node's state
  - Have 1 parent, a length and cost, point to a problem state
  - Expand uses successor function to create new tree nodes
  - The same problem state in multiple search tree nodes



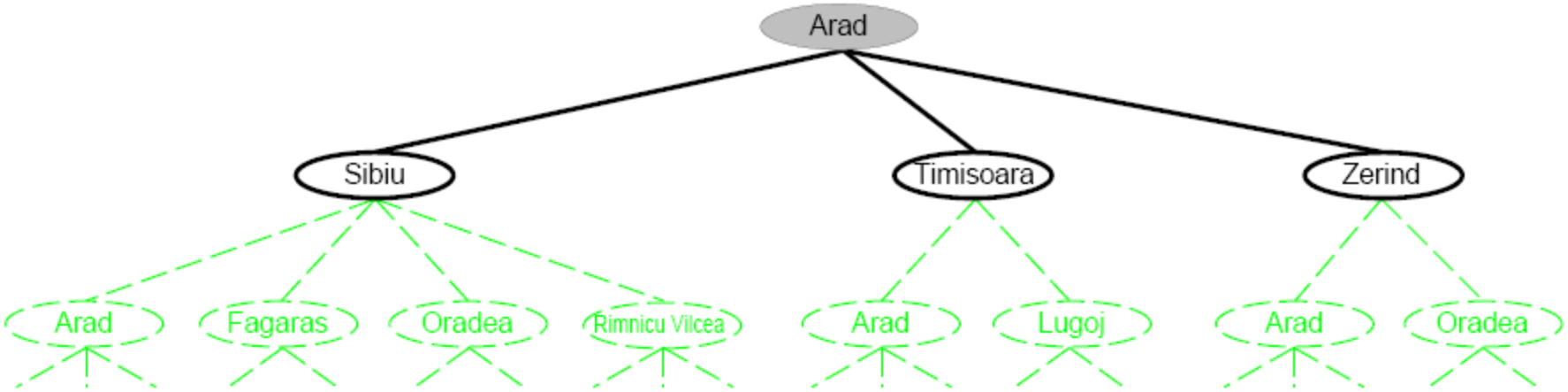
Example: Traveling from Arad To Bucharest



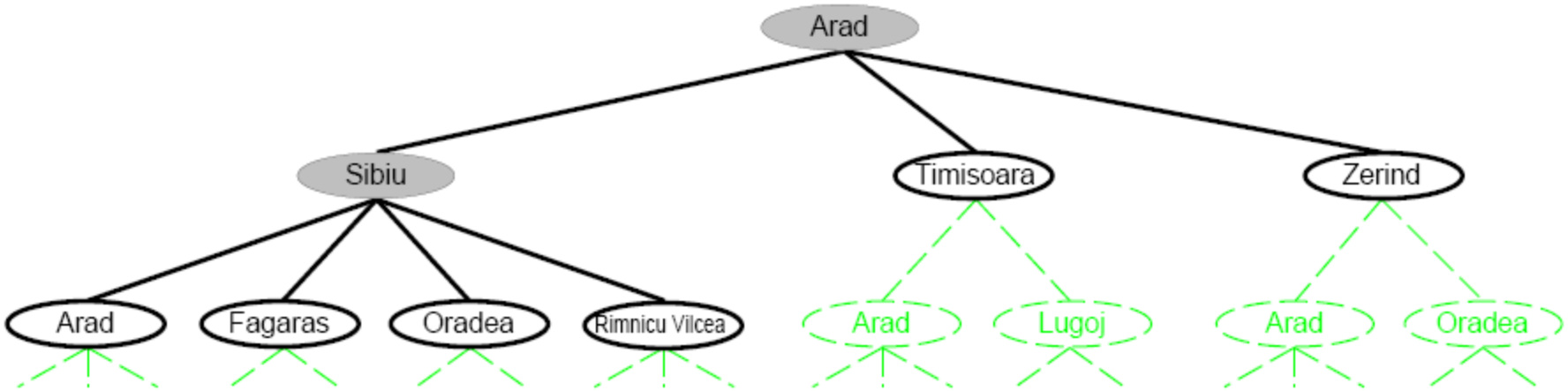
Search example



Search example



Search example



Uninformed search strategies

Use only information available in the problem formulation

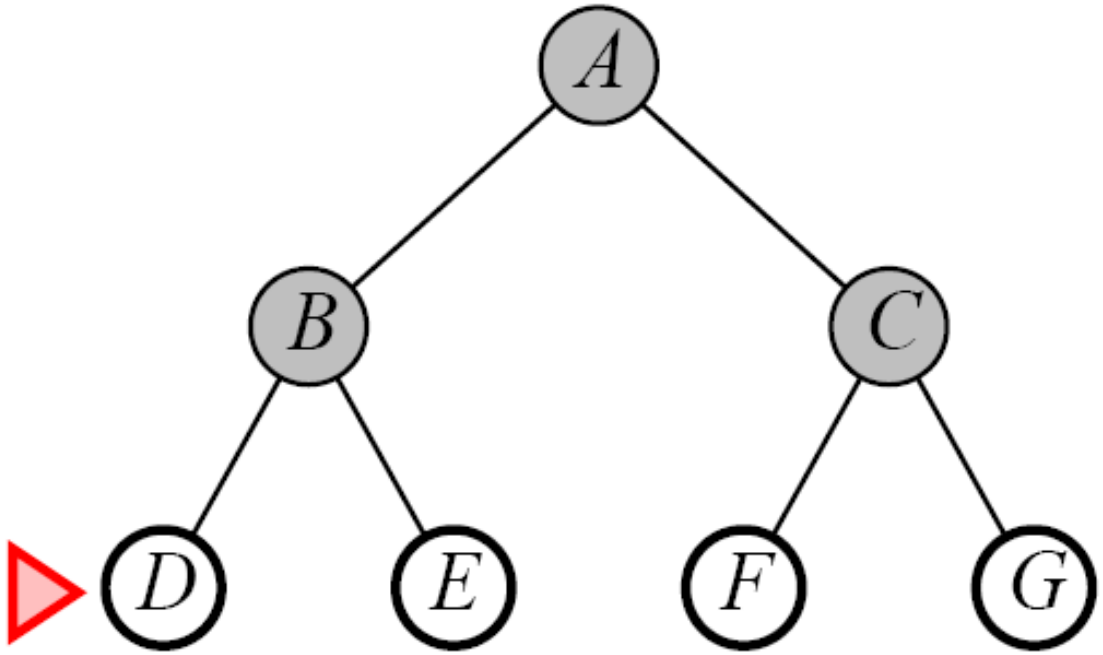
- Breadth-first
- Uniform-cost
- Depth-first
- Depth-limited
- Iterative deepening

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

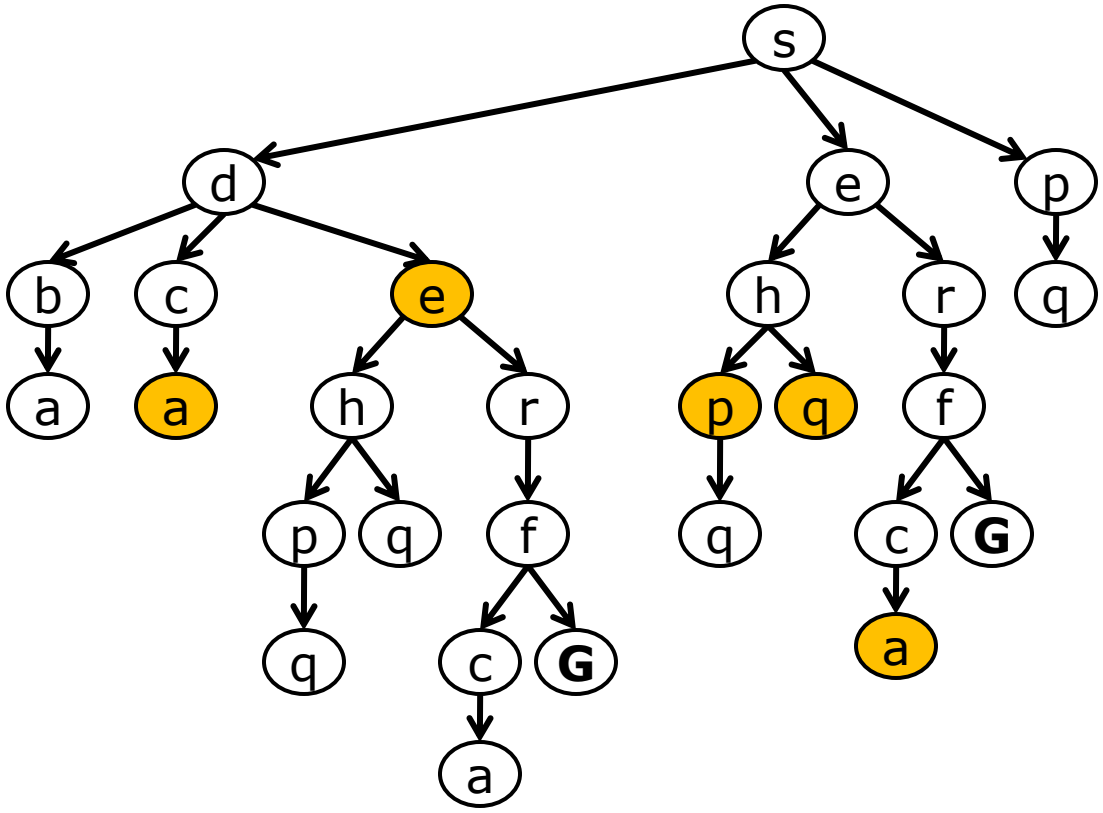
- Completeness: Yes (if b is finite)
- Time complexity: $1+b+b^2+b^3+\ldots+b^d+b\,(b^d-1)=O(b^{d+1})$
- Space complexity: $O(b^{d+1})$ (keeps every node in memory)
- Optimality: Yes, if cost=1 per step; not in general

- Search algorithms are commonly evaluated according to the following four criteria:
  - **Completeness:** does it always find a solution if one exists?
  - **Time complexity:** how long does it take as function of num. of nodes?
  - **Space complexity:** how much memory does it require?
  - **Optimality:** does it guarantee the least-cost solution?

- Time and space complexity are measured in terms of:
  - *b* – max branching factor of the search tree
  - *d* – depth of the least-cost solution
  - *m* – max depth of the search tree (may be infinity)

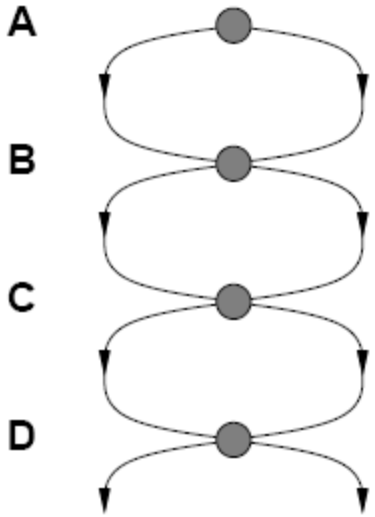
Graph Search

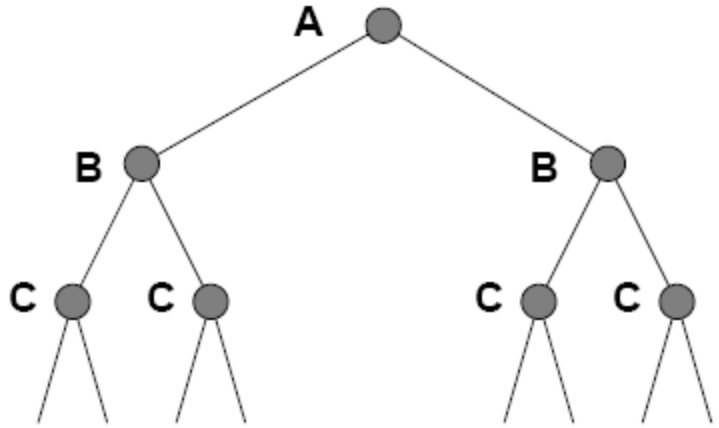
- In BFS we shouldn't bother expanding the filled-out nodes (why?)



Repeated States

- Failure to detect repeated states can turn a linear problem into an exponential one!



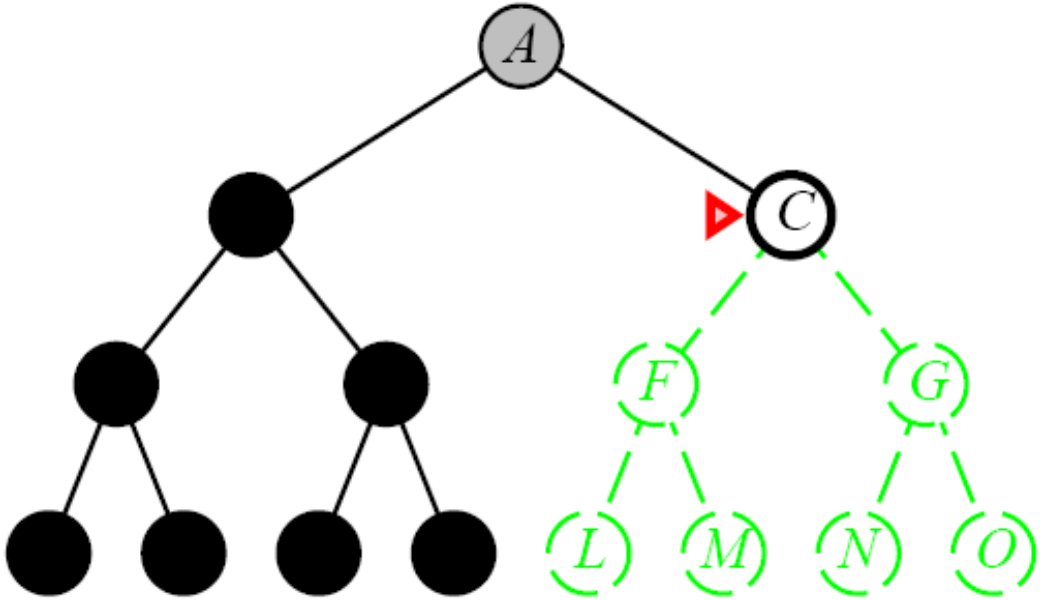


Depth-first search

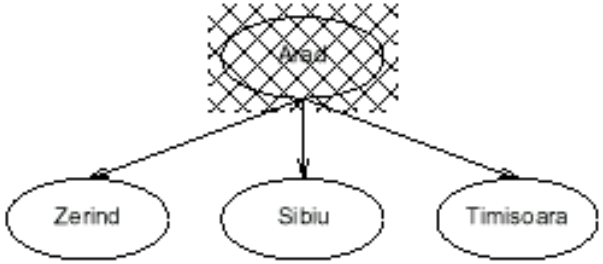
Expand deepest unexpanded node

Implementation:

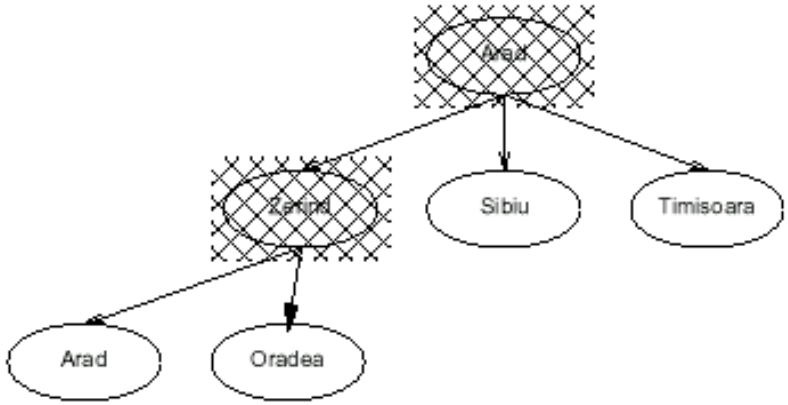
fringe = LIFO queue, i.e., put successors at front



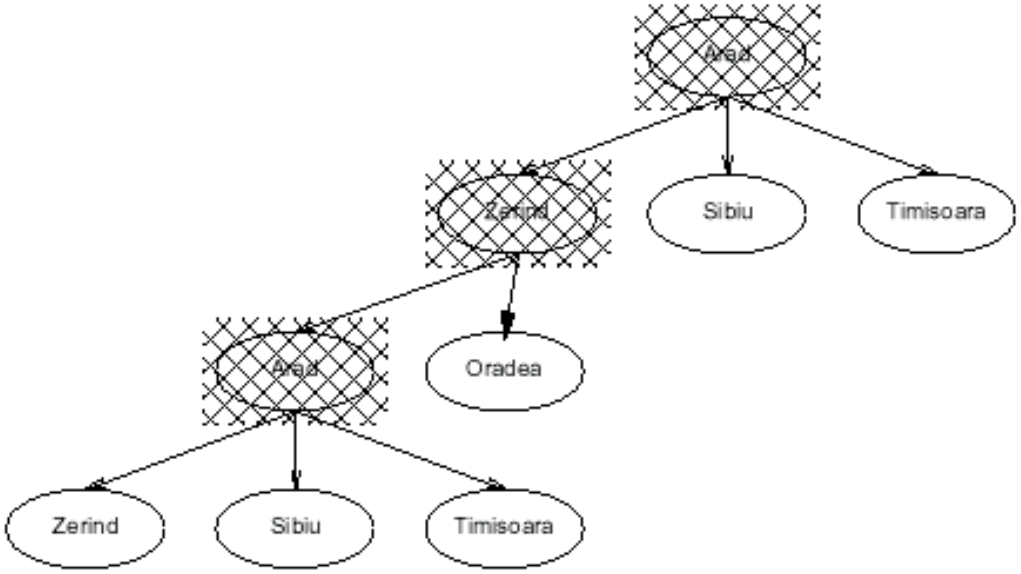
Depth-first search



Depth-first search



Depth-first search



I.e., depth-first search can perform infinite cyclic excursions
Need a finite, non-cyclic search space (or repeated-state checking)

Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d

but if solutions are dense, may be much faster than breadth-first

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

Optimal?? No

Comparing uninformed search strategies

Criterion	Breadth-first	Uniform cost	Depth-first	Depth-limited	Iterative deepening	Bidirectional (if applicable)
Time	b^{d+1}	$b^{\lceil C^*/\varepsilon \rceil}$	b^m	b^l	b^d	$b^{(d/2)}$
Space	b^{d+1}	$b^{\lceil C^*/\varepsilon \rceil}$	bm	bl	bd	$b^{(d/2)}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes if $l \geq d$	Yes	Yes

- b – max branching factor of the search tree
- d – depth of the least-cost solution
- m – max depth of the state-space (may be infinity)
- l – depth cutoff

Now: heuristic search [AIMA Ch. 4]

Informed search:

Use heuristics to guide the search

- Best first
- A*
- Heuristics
- Hill-climbing
- Simulated annealing

A* search

- Idea: avoid expanding paths that are already expensive

  evaluation function: $f(n) = g(n) + h(n)$

  where:

  $g(n)$ – cost so far to reach *n*

  $h(n)$ – estimated cost to goal from *n*

  $f(n)$ – estimated total cost of path through *n* to goal

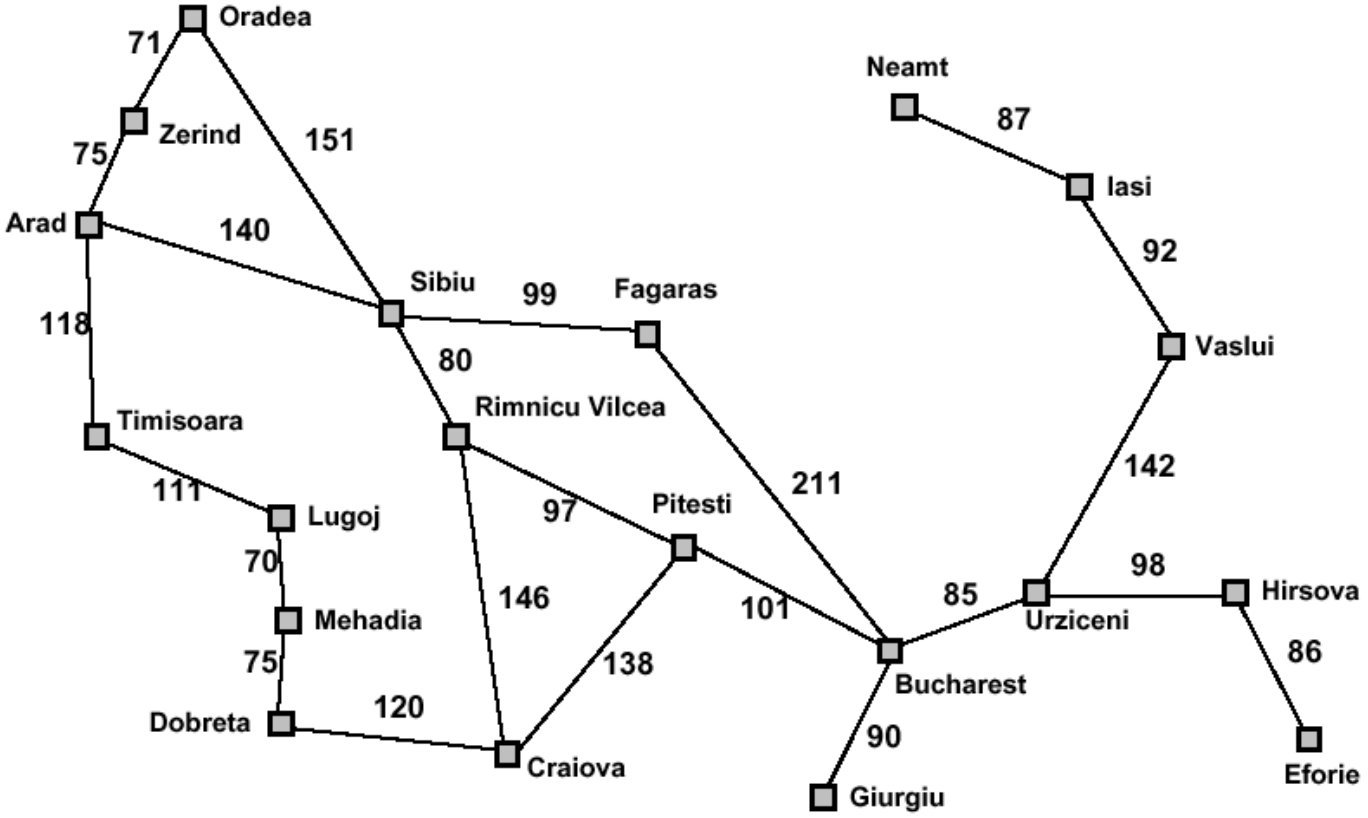
- A* search uses an *admissible* heuristic, that is,

  $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from *n*.

  For example: $h_{SLD}(n)$ never overestimates actual road distance.

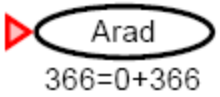
- Theorem: A* search is optimal

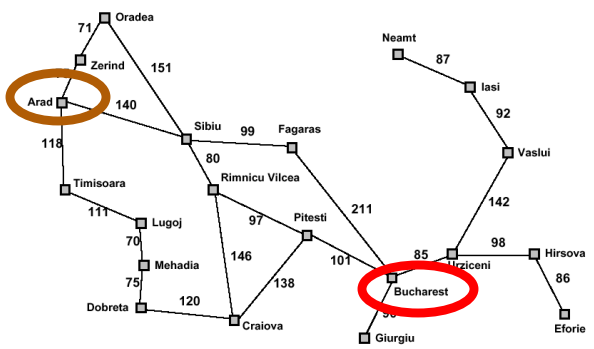
Romania with step costs in km



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

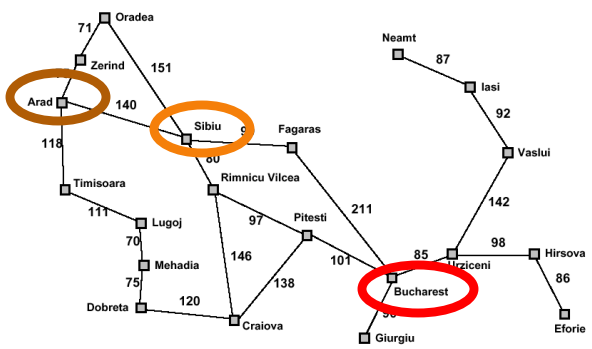
A* Search Example





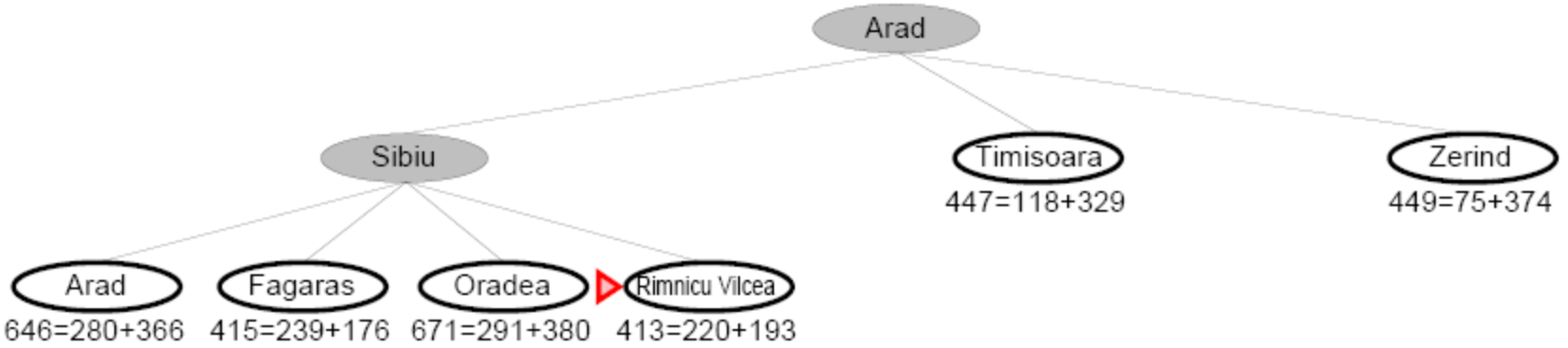
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

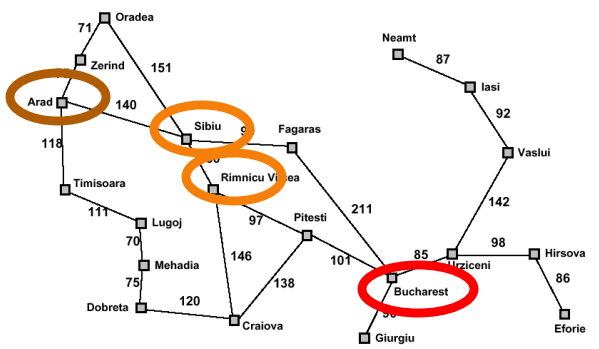
A* Search Example



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

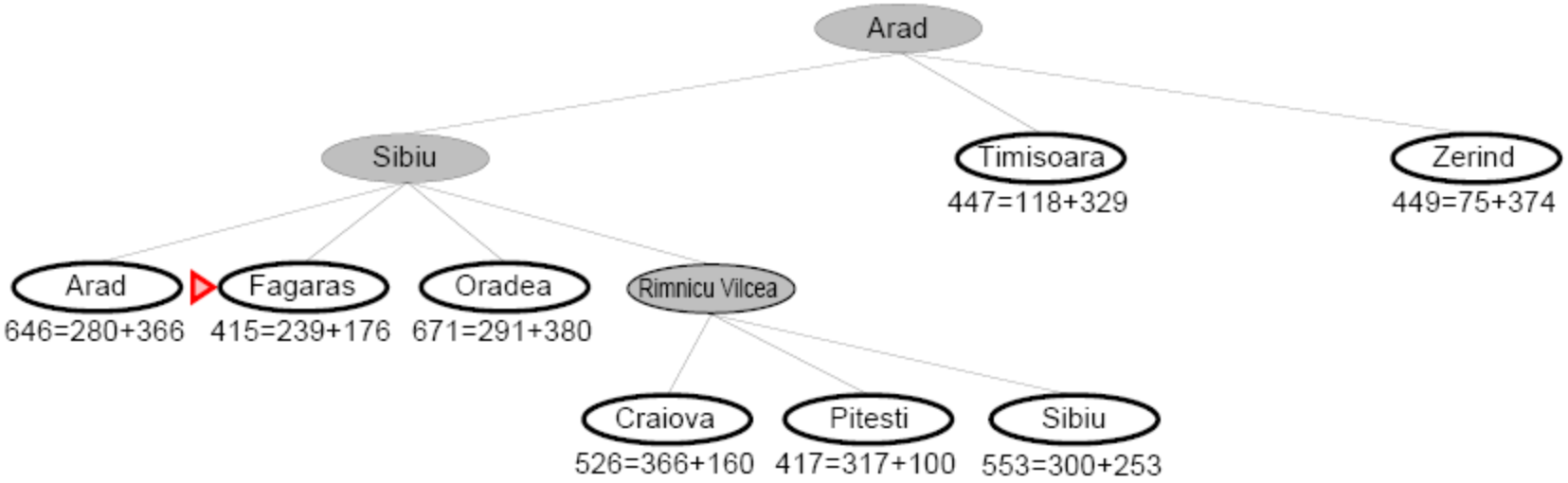
A* Search Example

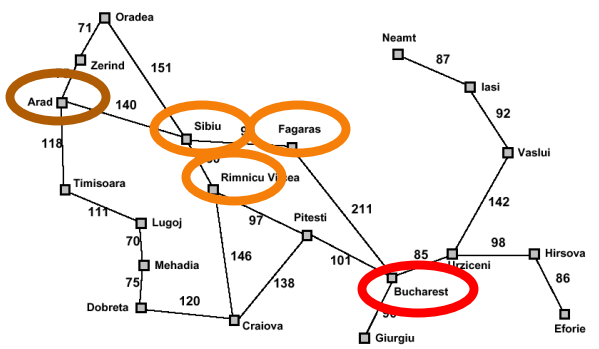




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search Example

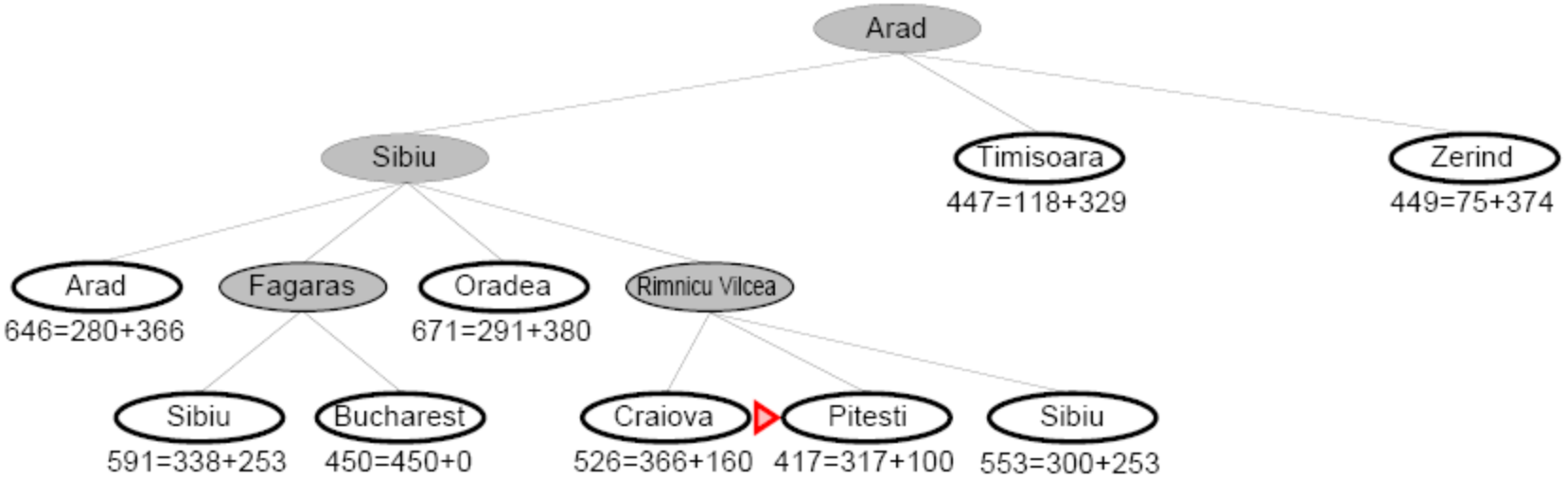


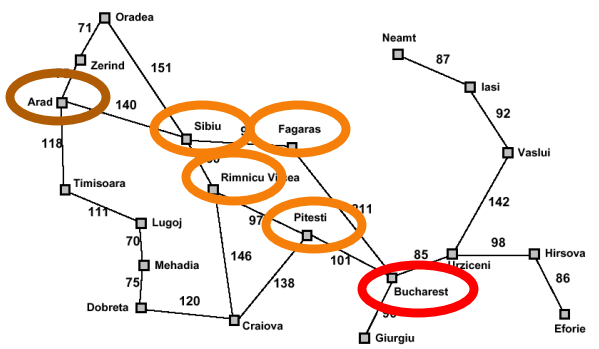


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	**100**
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

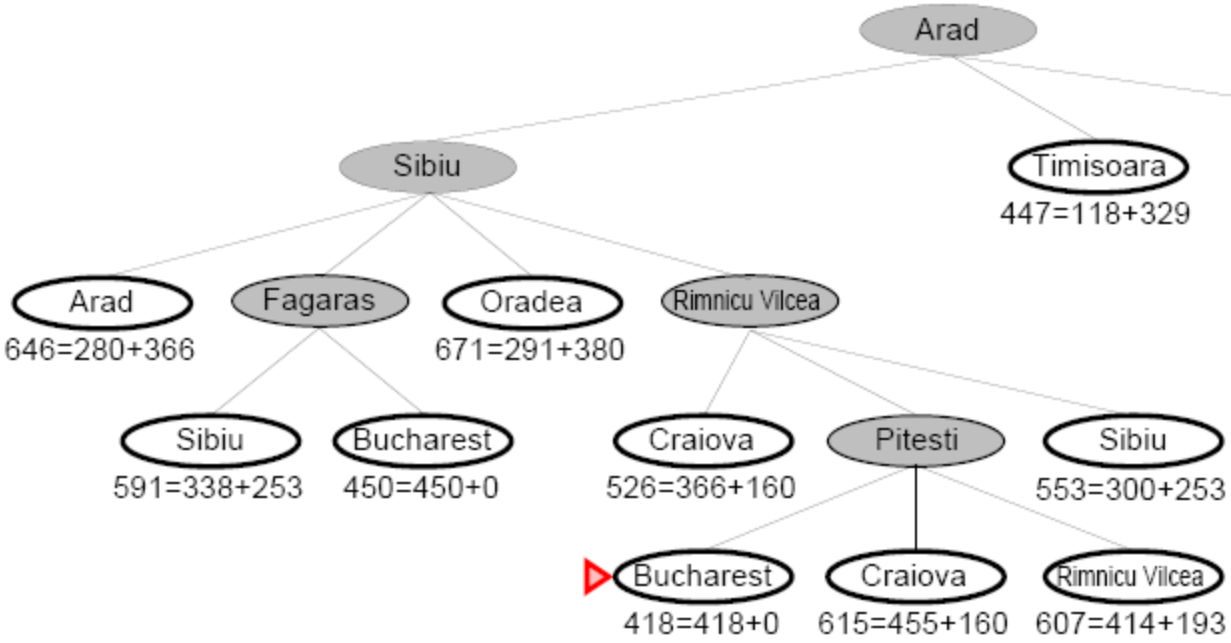
A* Search Example

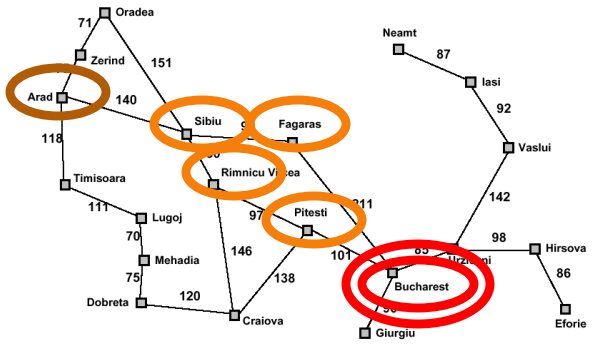




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search Example

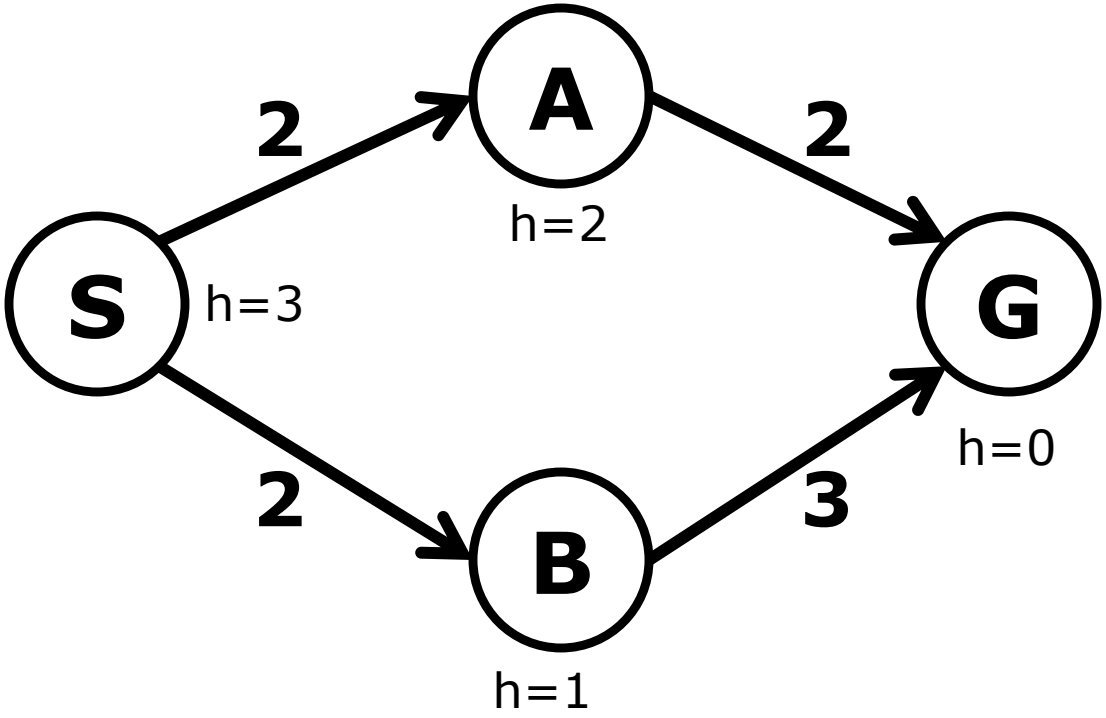




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	**100**
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Properties of A*

- Complete?

  Yes, unless there are infinitely many nodes with $f \leq f(G)$

- Time?

  Exponential in [relative error in h x length of soln.]

- Space?

  Keeps all nodes in memory.

- Optimal?

  Yes – cannot expand f_{i+1} until f_i is finished.

  A* expands all nodes with $f(n) < C^*$
  A* expands some nodes with $f(n) = C^*$
  A* expands no nodes with $f(n) > C^*$

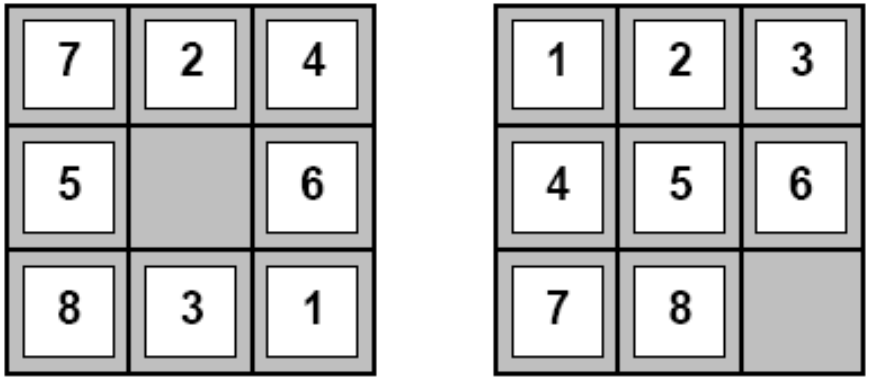
Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



$h_1(S) = ??$ 6

$h_2(S) = ??$ 4+0+3+3+1+0+2+1 = 14

A* Applications

- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

Summary: A*

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest h
  - incomplete and not always optimal
- A* search expands lowest $g + h$
  - complete and optimal
  - also optimally efficient (up to tie-breaks, for forward search)
- Admissible heuristics can be derived from exact solution of relaxed problems