TDT4136 Introduction to Artificial Intelligence

Lecture 3 : Problem Solving as Search

Chapter 3 in the textbook.

Pinar Öztürk

Norwegian University of Science and Technology
2021

Recap from last lecture

- PEAS
- Rationalty
- Environment characteristics and types
- Agent types

Representation of States and Transitions

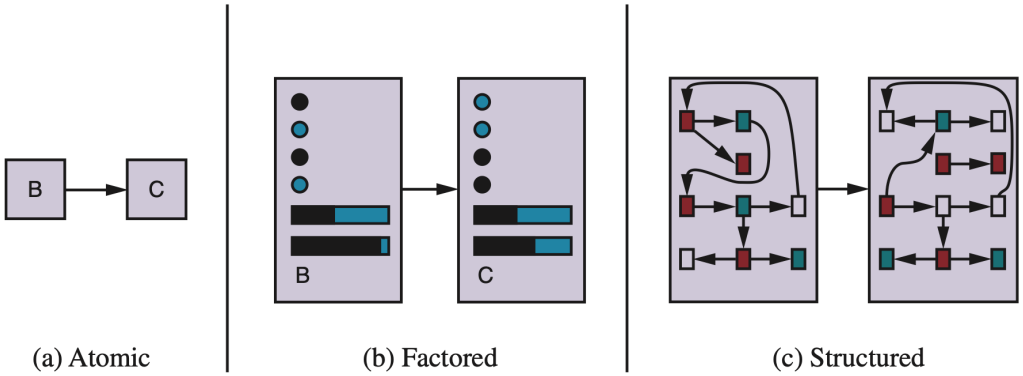


Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

Outline of Today

- Problem solving as state space search
- Uninformed search - Informed Search - Heuristic - Examples on heuristic search: Greedy Best Search and A* Search

Problem solving and search

- Some problems have straightforward solutions

  Solved by applying a formula, or a well-known procedure
  Example: differential equations

- Other problems require search:

  no single standardized method
  alternatives need to be explored to solve the problem
  the number of alternatives to search among can be very large, even infinite.

Example: Cabbage, goat, wolf and farmer

1

Brain Teasers

The Wolf, Goat, and Cabbage

This problem can be found in eighth-century writings.

A man has to take a wolf, a goat, and some cabbage across a river. His rowboat has enough room for the man plus either the wolf or the goat or the cabbage. If he takes the cabbage with him, the wolf will eat the goat. If he takes the wolf, the goat will eat the cabbage. Only when the man is present are the goat and the cabbage safe from their enemies. All the same, the man carries wolf, goat, and cabbage across the river. How?

Puzzle provided by Kordemsky: The Moscow Puzzles (Dover)

VIEW SOLUTION

DOWNLOAD PROBLEM AND SOLUTION AS PDF

How many river crossings does the farmer need? 4,5,6,7, or no solution?
How did you think to solve this problem?

[1] https://illuminations.nctm.org/BrainTeasers.aspx?id=4992

Cabbage, goat, wolf, and the farmer

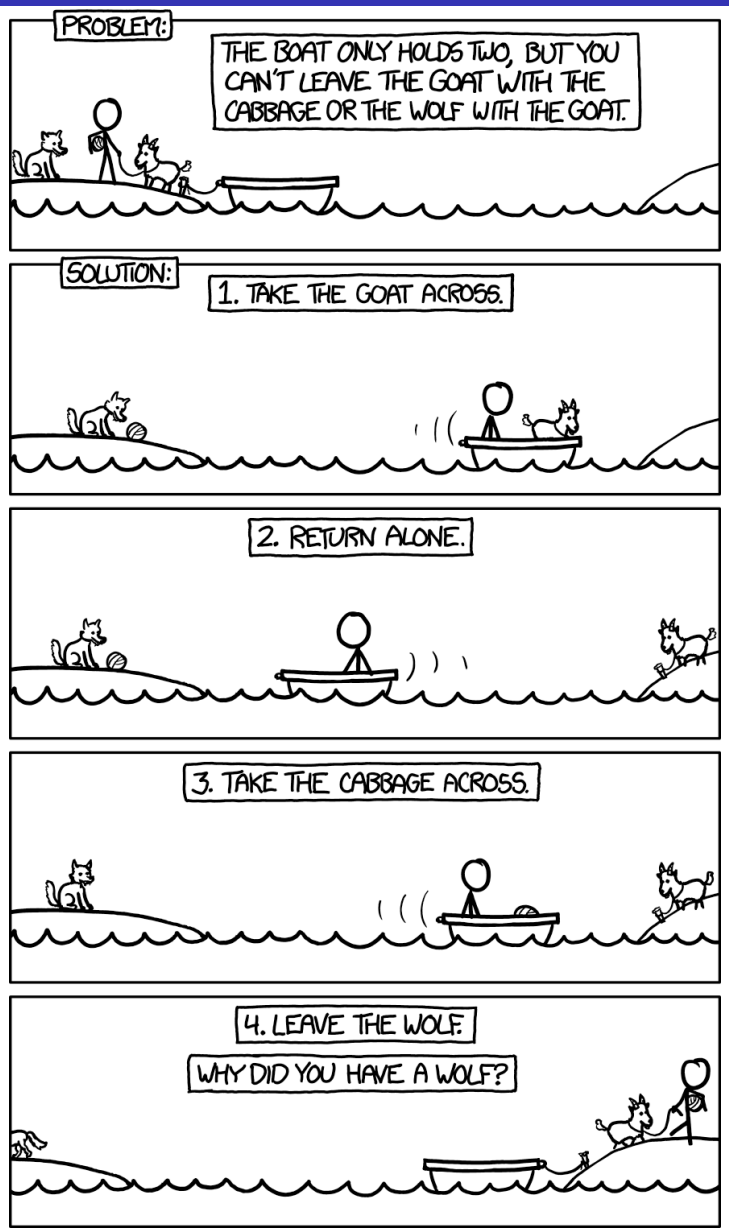


Figure from https://xkcd.com/1134/

Cabbage, goat, wolf and farmer - an answer

the river. How?

Puzzle provided by Kordemsky: The Moscow Puzzles (Dover)

VIEW SOLUTION

DOWNLOAD PROBLEM AND SOLUTION AS PDF

Solution:

The wolf does not eat cabbage, so the crossing can start with the goat.

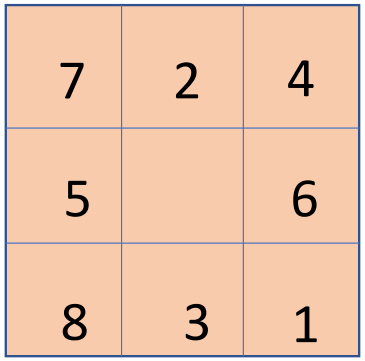
The man leaves the goat and returns, puts the cabbage in the boat and takes it across. On the other bank, he leaves the cabbage but takes the goat.

He leaves the goat on the first bank and takes the wolf across. He leaves the cabbage with the wolf and rows back alone.

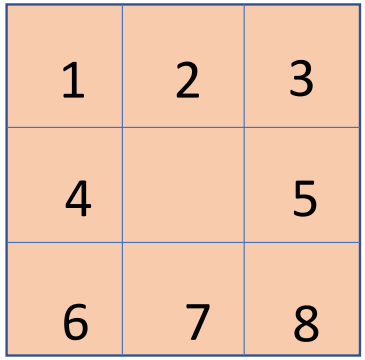
He takes the goat across.

- The important thing is not how you solved this manually
- Important is how we can make the machine to solve such problems automatically
- What kind of systematical approach in order to considers possibilities?
- Search is a suitable method to do this

Example : solving 8-puzzle



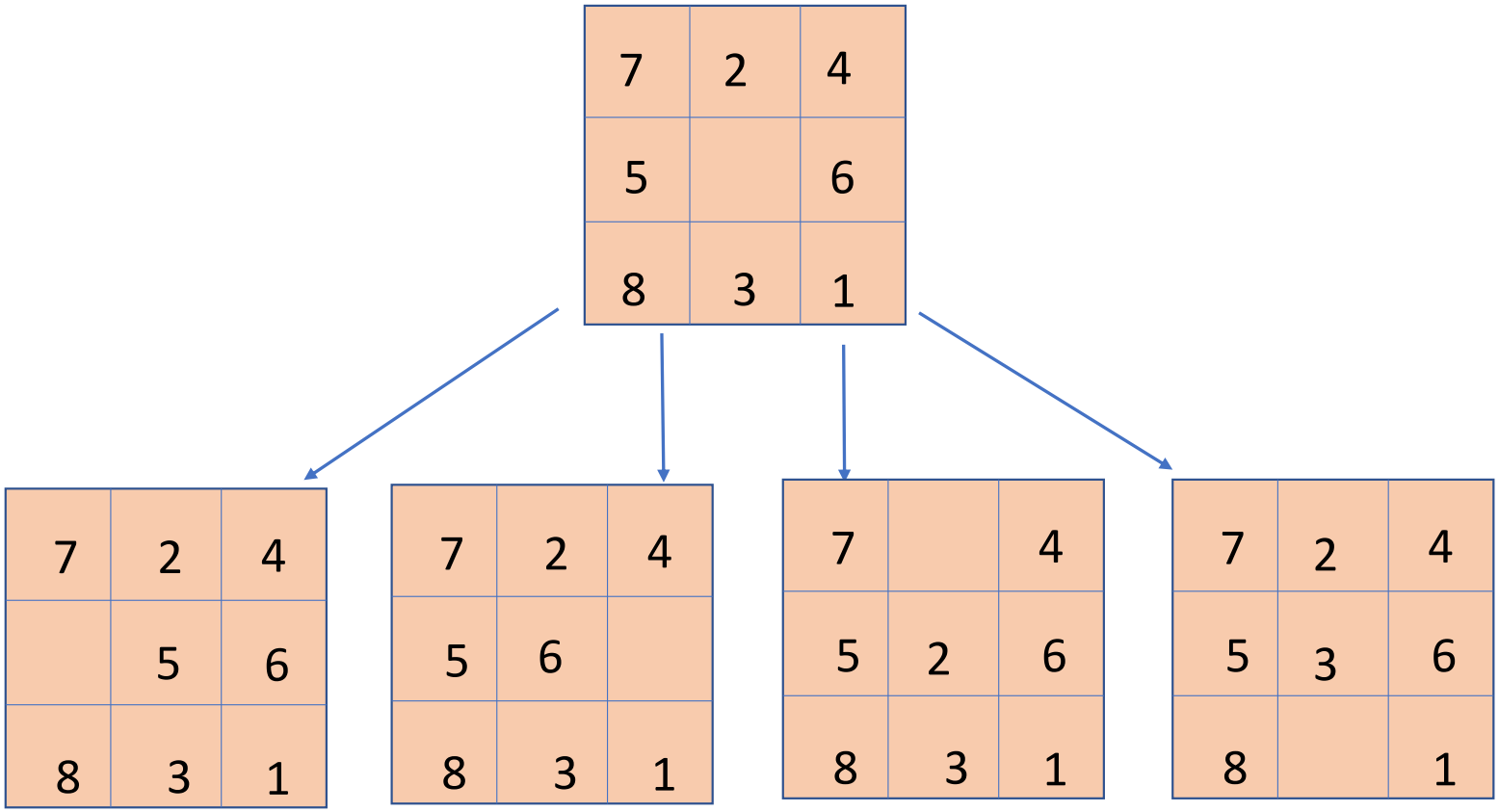
Initial state



Goal state

What is a **state**?

Example: 8-puzzle



Search is about exploring alternatives.

Problem solving process

Note that our current focus is fully observable and deterministic environments.

Problem-solving agents

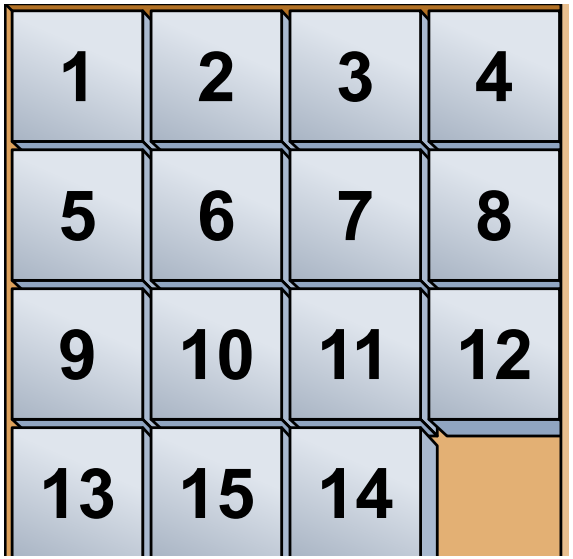
A type of goal-based agent.

```
function SIMPLE-PROBLEM-SOLVING-AGENT( percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH( problem)
    action ← FIRST(seq, state)
    seq ← REST(seq, state)
    return action
```

Solution

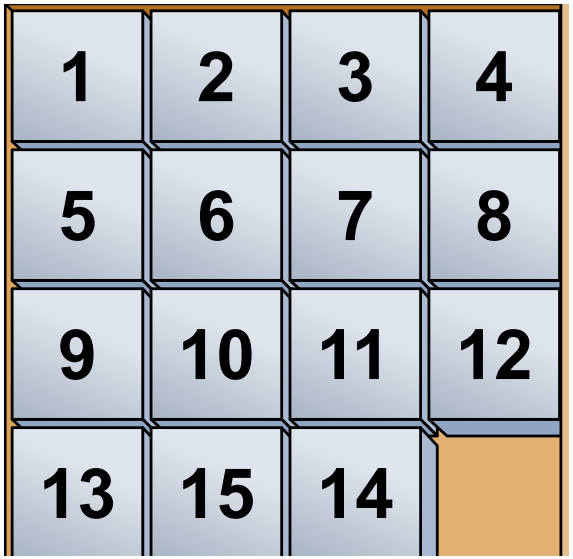
A solution is a sequence of actions that connect the initial state to a goal state.



Sam Loyd invented this puzzle in 1870's.

Solution

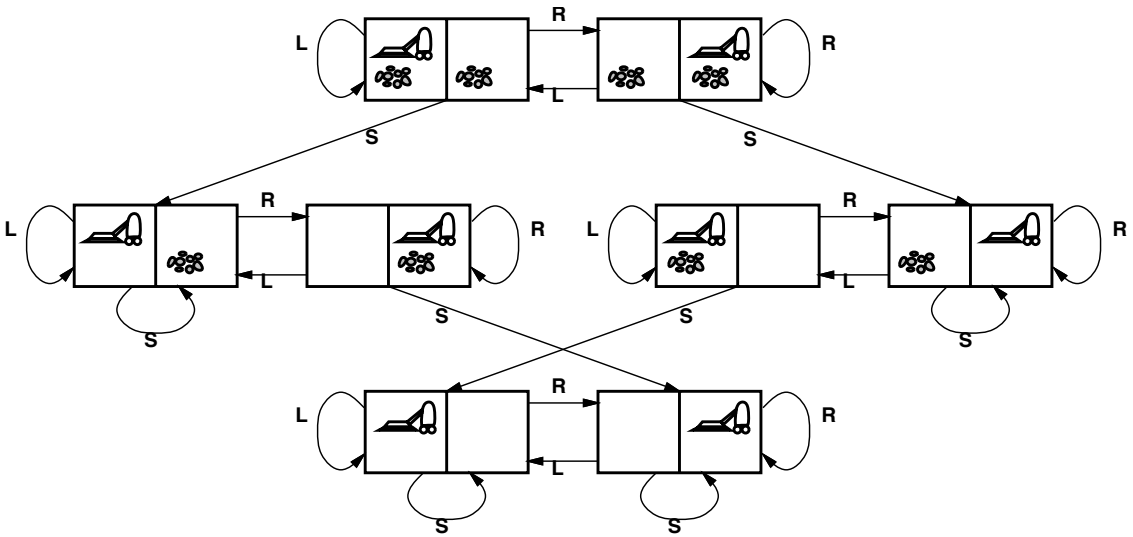
A solution is a sequence of actions that connect the initial state to a goal state.



Sam Loyd invented this puzzle in 1870's.

! Sometimes no solution exists

State Space



A state: the agent location and the dirt locations
State Space: All the possible states and the links/relationship (in terms of actions) between them

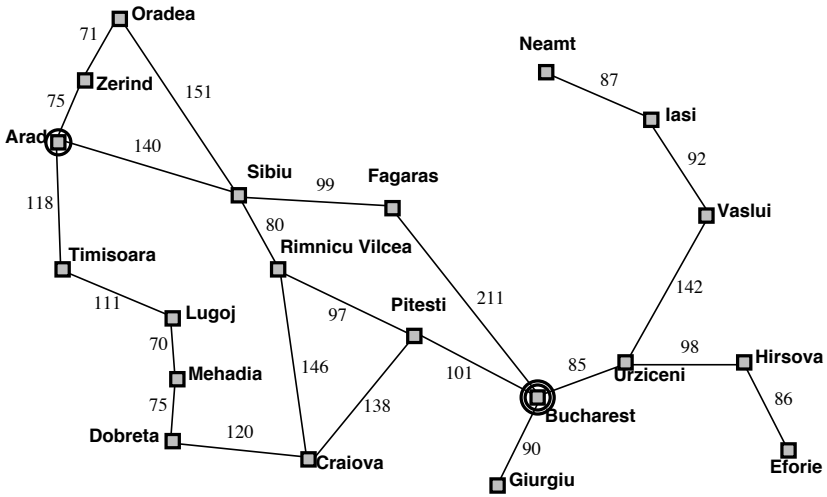
- How many states in this state space?
- The agent can be in one of the 2 locations, and each location may/not have dirt. So, 2×2^2 possible states.
- For an n-location environment, the state space has $n \times 2^n$ states

State space -Romania example

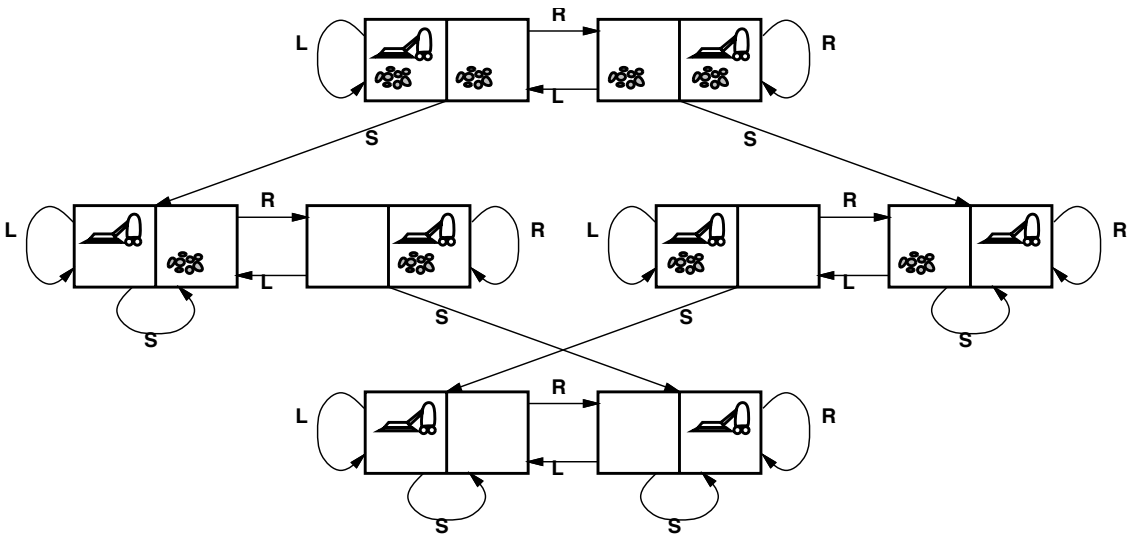
On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest.

Formulation of the search problem:

States: various cities
Initial state: in Arad
Goal: be in Bucharest
Actions: drive to a city, e.g.,ToZerind
Transition model: ToZerind moves the agent to Zerind.
Action cost: c(city1, action, city2); distance between city1 and city2



Example: vacuum world state space graph



states??:

initial state??:

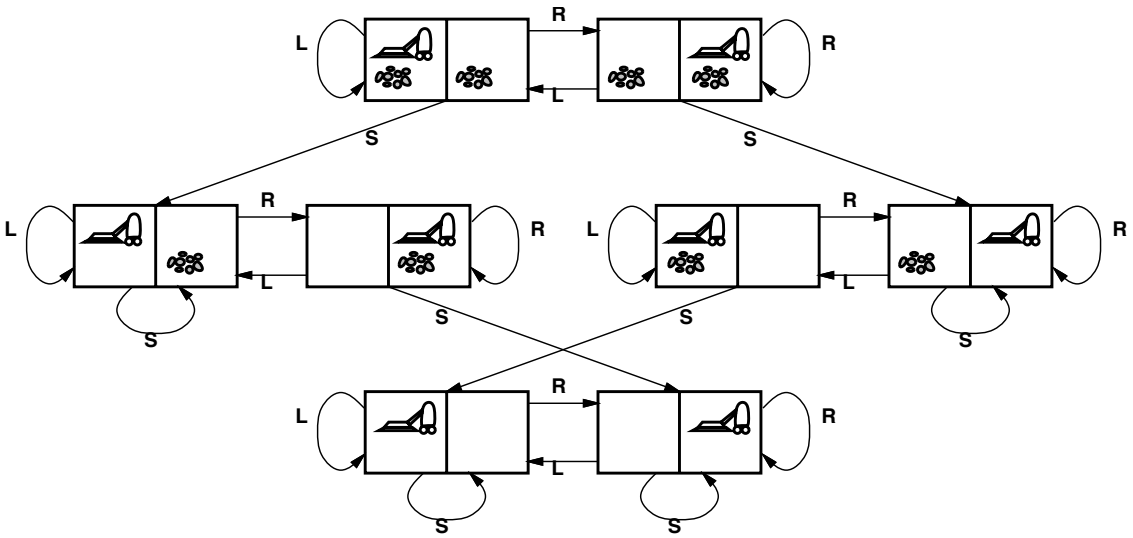
actions??:

transition model??:

goal states:??

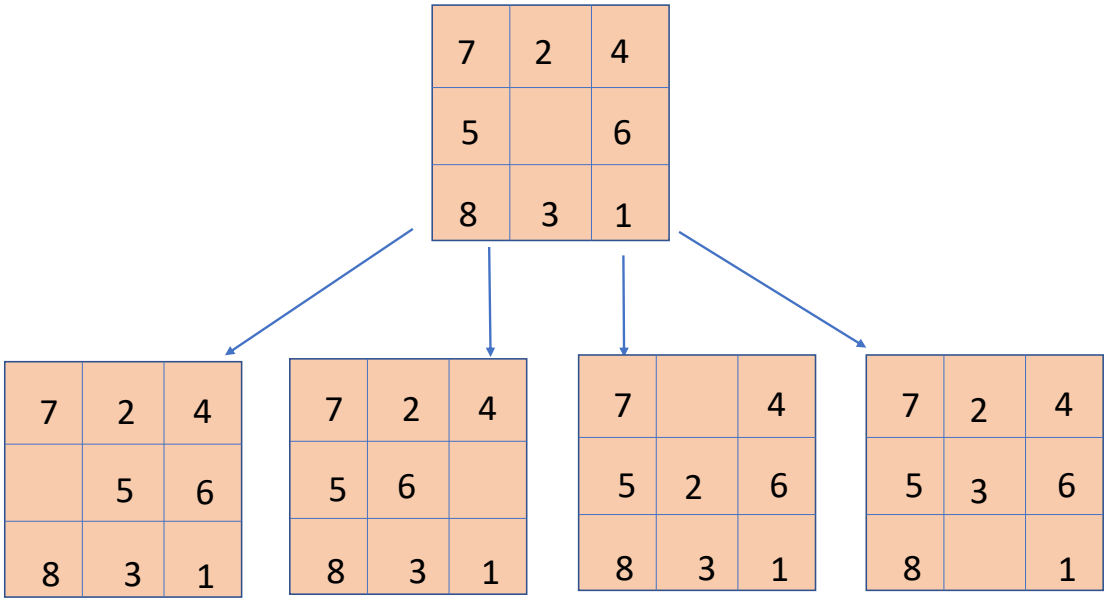
action cost:??

Example: vacuum world state space graph



states: integer dirt and robot locations (ignore dirt amount etc.)
initial state: any state can be defined as the initial state
actions: Suck, moveLeft,moveRight - or Suck, Forward, Backward, TurnLeft, TurnRight
transition model: Suck cleans the location, TurnLeft changes the direction of agent 90 degrees
goal states: no dirt
action cost: 1 unit

Example: The 8-puzzle



states??:

initial state??:

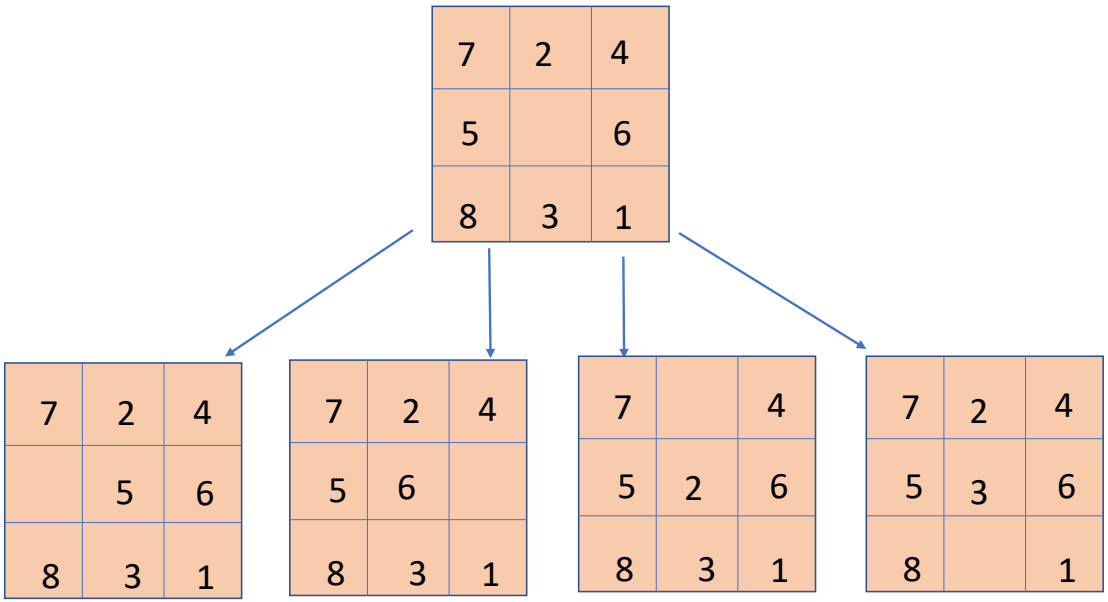
actions??:

transition model??:

goal??:

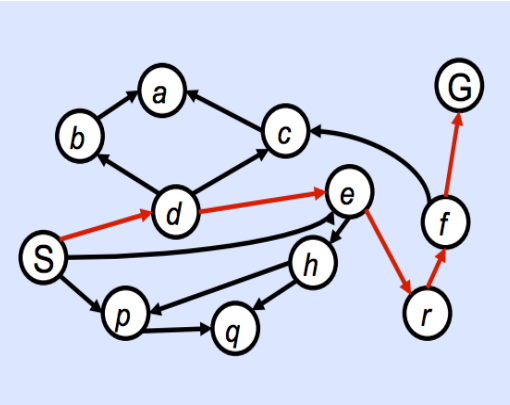
action cost??:

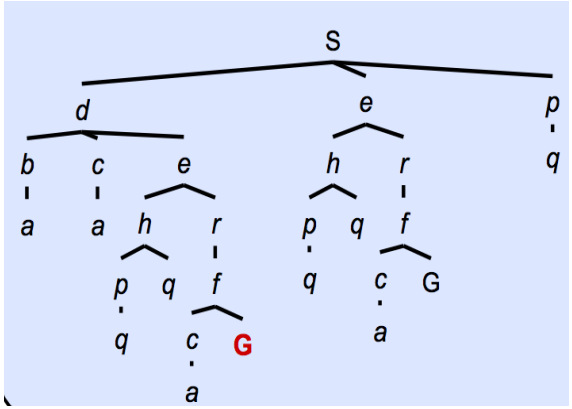
Example: The 8-puzzle



states: integer locations of tiles (ignore intermediate positions)
initial state: initial state (given)
actions: move blank left, right, up, down (ignore unjamming etc.)
transition model: the resultant configuration of the grid when an action is applied
goal: = goal state (given)
action cost: 1 unit per action

Search algorithms



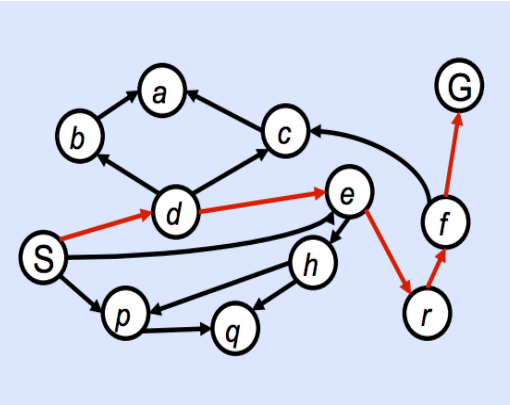


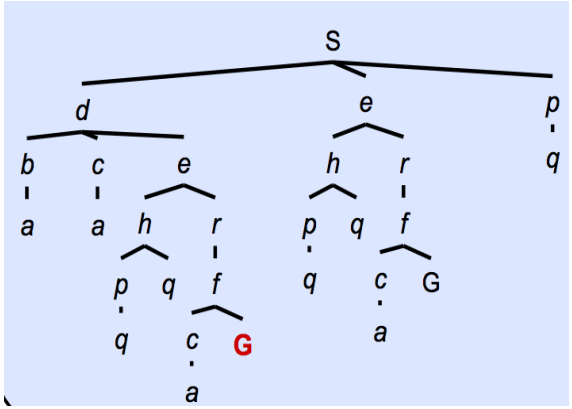
- A state is a (representation of) a physical configuration

- A state space graph represent all possible *states of the environment* and the transitions between the states.

- Search tree is superimposed on the state-space graph and shows how a particular search algorithm explores the state-space graph.

Search algorithms





- A state is a (representation of) a physical configuration

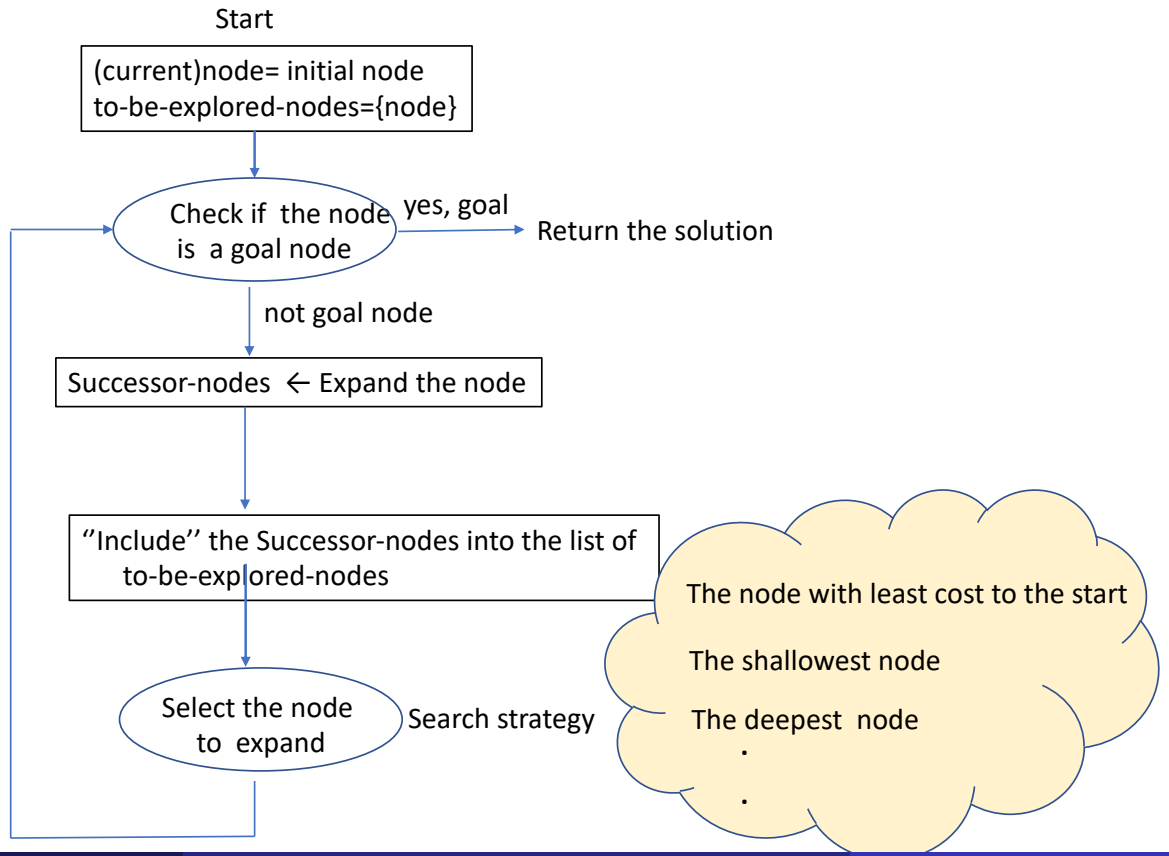
- A state space graph represent all possible *states of the environment* and the transitions between the states.

- Search tree is superimposed on the state-space graph and shows how a particular search algorithm explores the state-space graph.

What does a *node*, the *root node*, and an *edge* in the search tree correspond to in a *problem formulation*?

State graph is incrementally explored

- Most of the time it is not feasible or too expensive to build and represent the entire state graph. The problem solver agent (i.e., AI) generates a solution by **incrementally exploring** a small portion of the graph
- **Basic idea:** simulated exploration of state space by generating successors of already-explored states

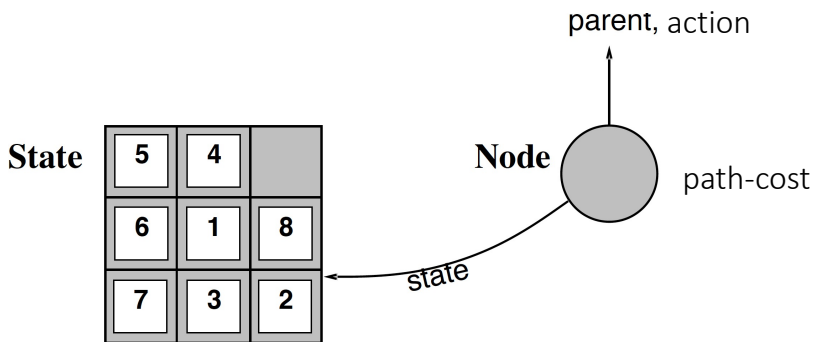


Best-first Search

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Data structure for the nodes in the search tree



A state is a (representation of) a physical configuration
A search strategy that uses the cost of the path as the evaluation metric and aims to minimize it uses a data structure for a node in the search tree with the following components:

- node.STATE
- node.PARENT
- node.ACTION
- node.PATH-COST

Some basic concepts

We need the following concepts for the search process:

- **Frontier**: the list of the nodes in the search tree that the search process considers worth to expand. Some of these will be expanded, some other not based on the progress of the search process
- The expanded nodes will have a special status. Some books including the 3rd eds of this book called the expanded nodes as **Closed** nodes where the ones in frontier are called as **open**.
- 4th edition adopts "frontier" and **reached** instead where the latter includes both the expanded nodes and nodes in the frontier.
- Hence nodes in the "reached" but not in the "frontier" are already expanded.

Data structure for Frontier

- Operations on a frontier
  - IS-EMPTY(frontier)
  - POP(frontier)
  - TOP(frontier).
  - ADD(node, frontier)
- The appropriate data structure for Frontier is a Que - different types:
  - Priority queue - pops the node with minimum cost defined by the evaluation function.
  - FIFO (first-in-first out) queue - pops the node that was added to the frontier first
  - LIFO (last-in-first-out) queue - pops the node that was added last.

Properties of search strategies

A strategy is defined by picking the order of node expansion

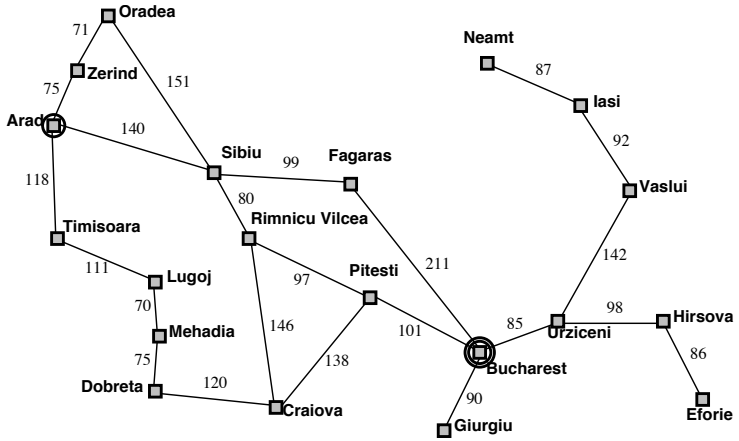
Strategies are evaluated along the following dimensions:

- completeness—does it always find a solution if one exists?
- time complexity—number of nodes generated/expanded
- space complexity—maximum number of nodes in memory
- cost optimality—does it always find a least-cost solution?

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 depth of the state space (may be ∞)

Redundant paths



- Arad is a *repeated* state in the search tree.
- Arad→ Sibiu→Arad. is a *loopy/cyclic* path ⇒*infinite* search tree ⇒ Problem!
- Loopy paths are a special case of *redundant paths* which need special treatment.
- There are also redundant but not loopy paths. Example: Two paths to Sibiu with difference path costs:
  1. Arad→Zerind→Oradea→Sibiu, and 2. Arad→Sibiu
- Need to eliminate redundant paths, keep only the best one.
- How?

How to Handle Redundant Paths

- Graph search (versus tree search). The *Best-first search* algorithm in the book is a *graph search* algorithm. Detects all redundant paths - can be found through checking *reached nodes* and nodes in the frontier in order to remove the costly paths.
- Don't care
- Check only cyclic loops - can be found by checking the chain of parent nodes

Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

Breadth-first search

Uniform-cost search

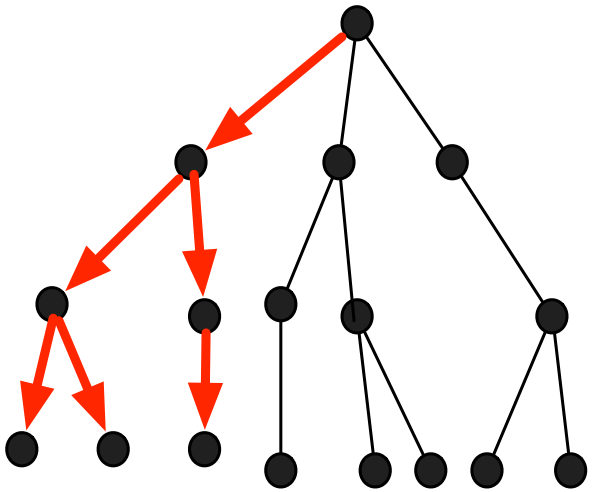
Depth-first search

Depth-limited search

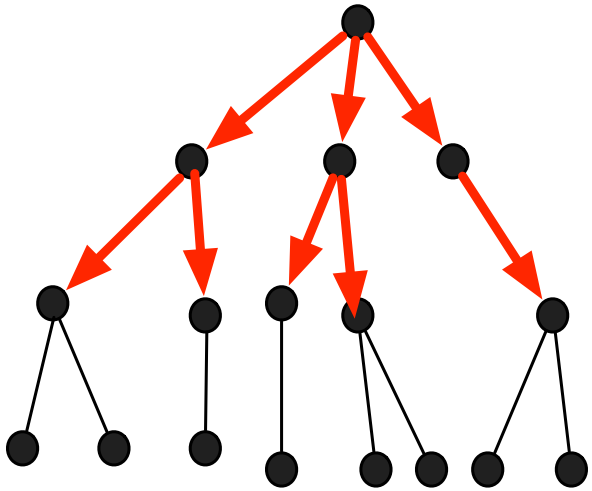
Iterative deepening search

Examples on Uninformed Search Strategies

Different "search strategies" explore the possibilities in different order



Depth-First



Breadth-First

Breadth-first search

Idea: explore the search space "horizontally", i.e. expanding all nodes at level k before taking into account nodes at level $k+1$

Implementation: The intended behavior can be ensured by using a FIFO policy for storing "to be expanded" nodes

Breadth-first search

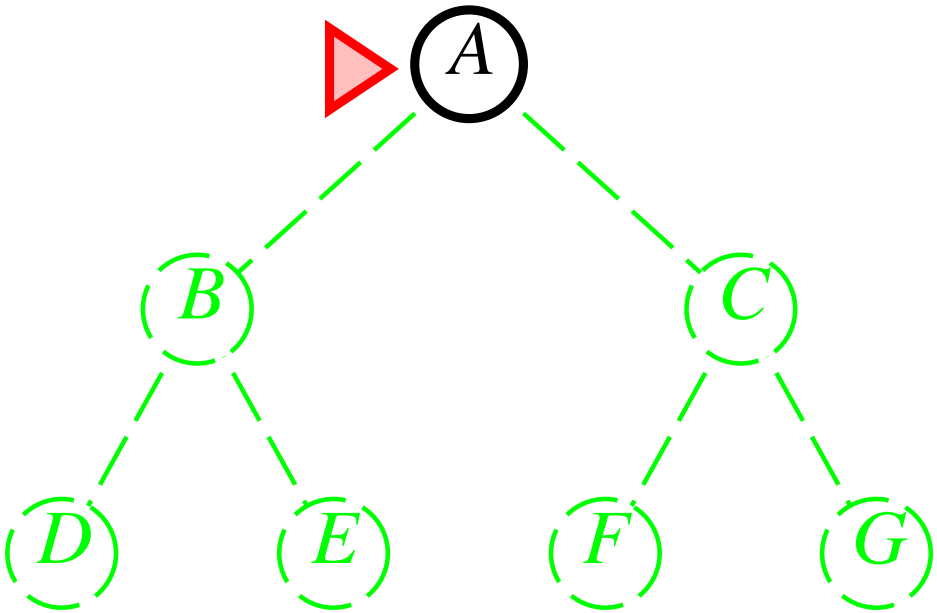
```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Breadth-first search

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

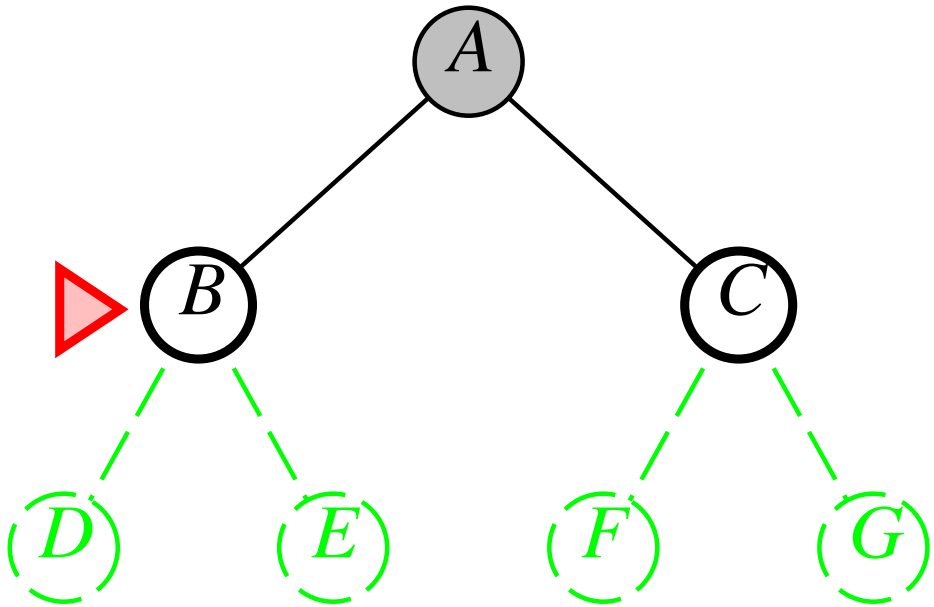


Breadth-first search

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

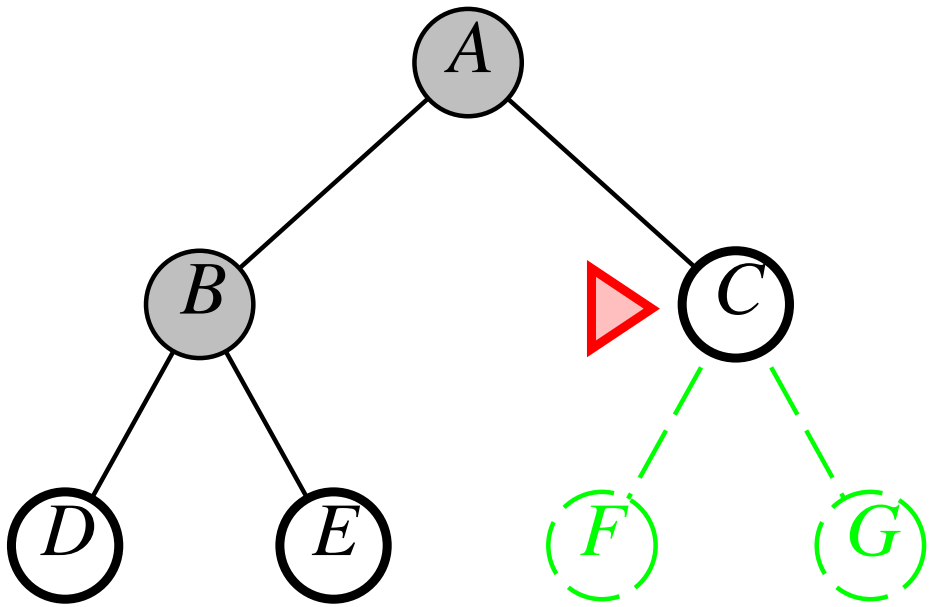


Breadth-first search

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

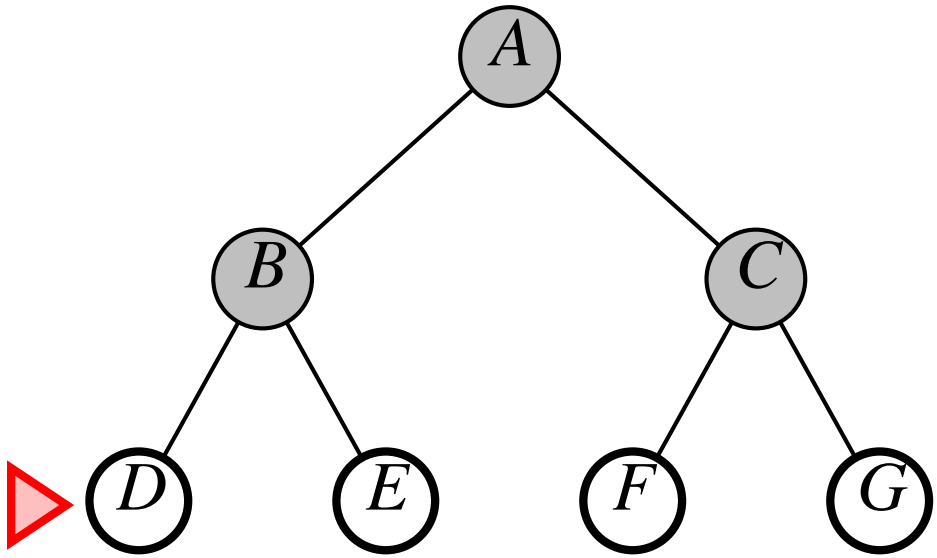


Breadth-first search

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

Complete?? Yes (if b is finite, and the state space either has a solution or is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exp. in d

Space?? $O(b^d)$ (keeps every node in memory)

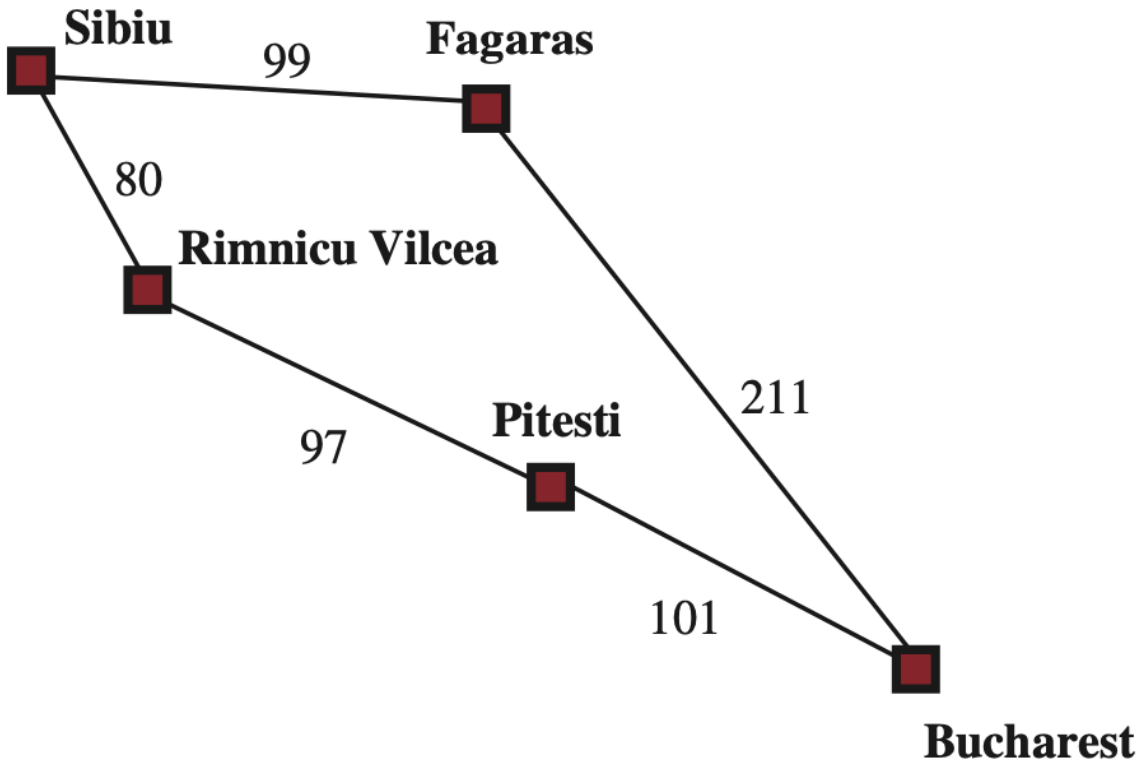
Cost optimal?? Yes (if actions have the same cost);

Space is the big problem

Uniform-cost search

An algorithm cost optimal with varying positive step costs.

Uniform-cost search



Uniform-cost search

Expand least-cost unexpanded node

Implementation:

frontier = queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

Complete?? Yes if b is finite, the state space either has a solution or is finite, and step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{1+\lfloor C^*/\epsilon \rfloor})$

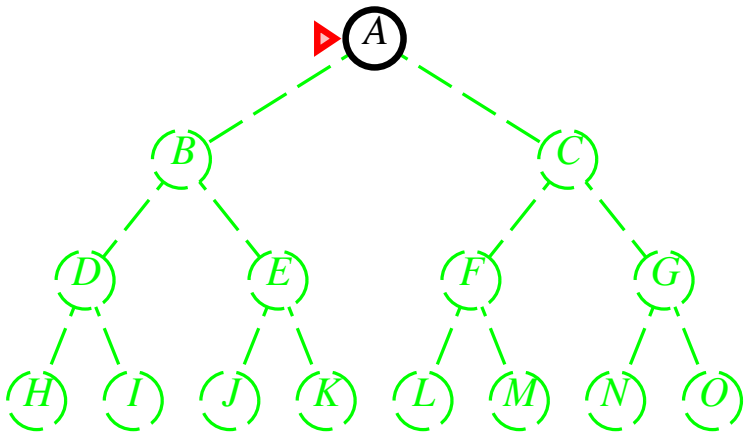
Cost optimal?? Yes - nodes expanded in increasing order of $g(n)$

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Initial state = A
Is A a goal state?
Put A at front of queue.
frontier = $[A]$

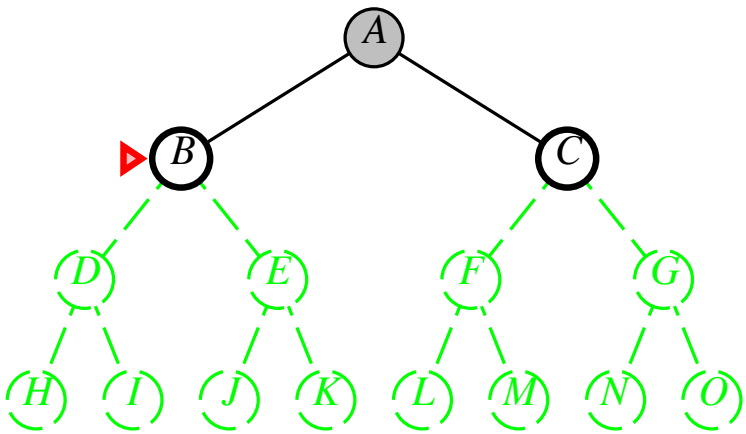
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand A to B, C.
Put B, C at front of queue.
frontier = $[B, C]$

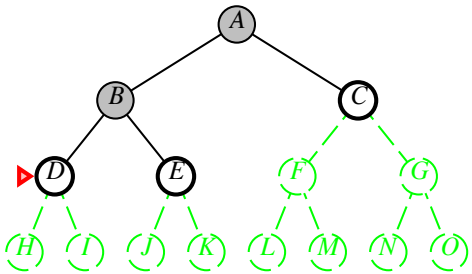
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand B to D, E. Put D, E at front of queue. frontier = $[D, E, C]$

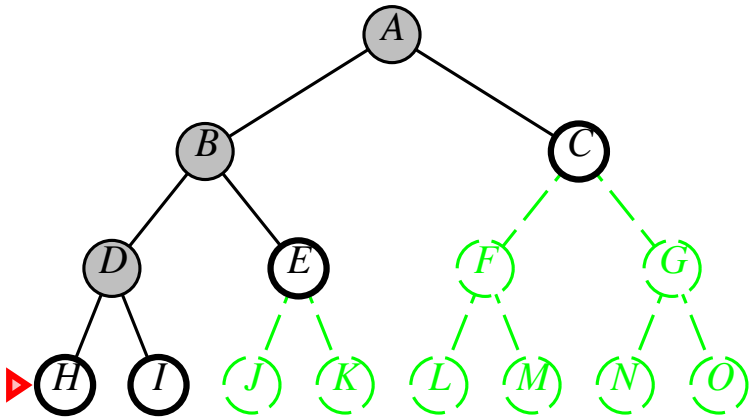
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand D to H, I.
IPut H, I at front of queue.
frontier $= [H, I, E, C]$

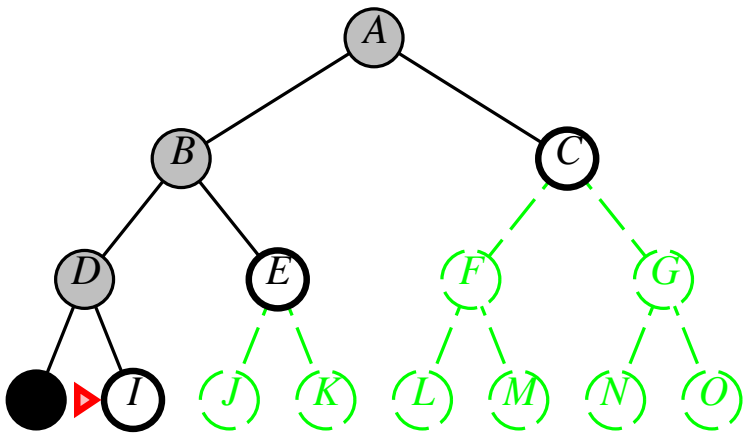
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand H to no children.
Forget H.
frontier = $[I, E, C]$

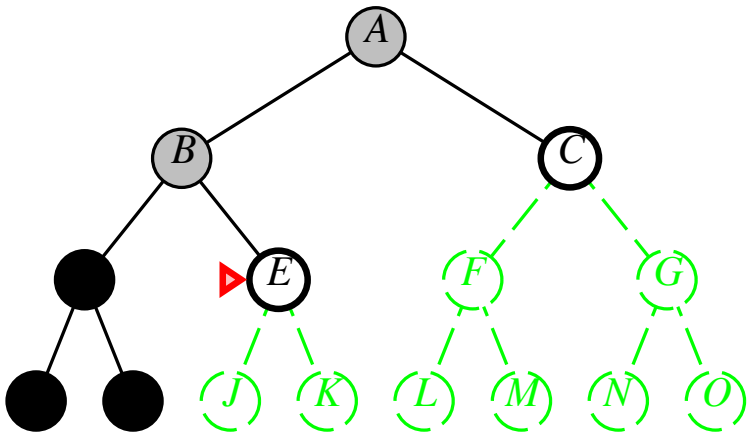
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand I to no children.
Forget D, I.
frontier = $[E, C]$

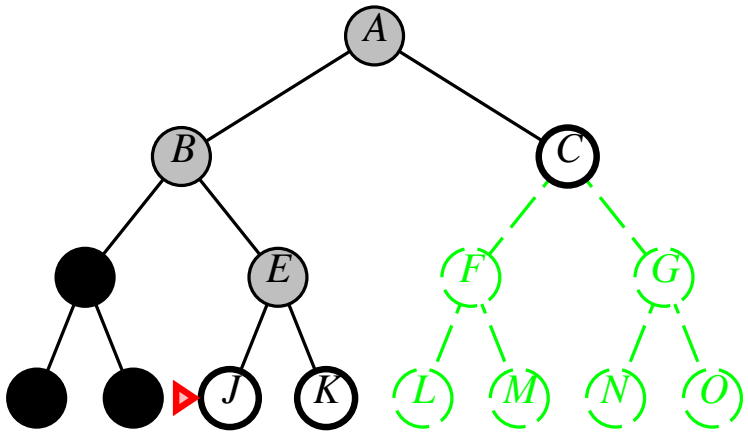
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand E to J, K.
Put J, K at front of queue.
frontier = $[J, K, C]$

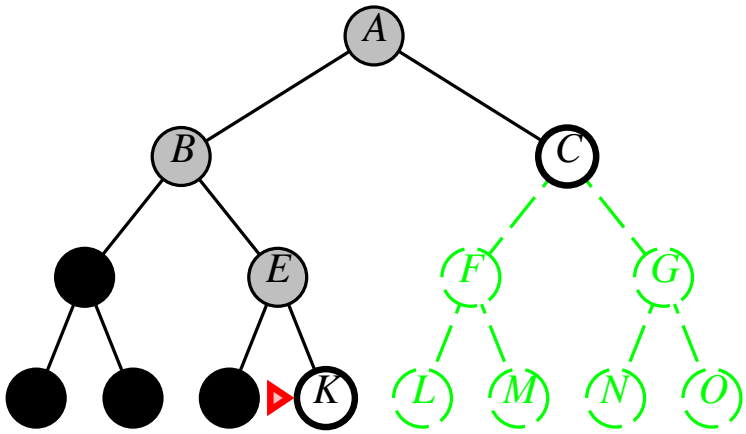
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand J to no children.
Forget J.
frontier = $[K, C]$

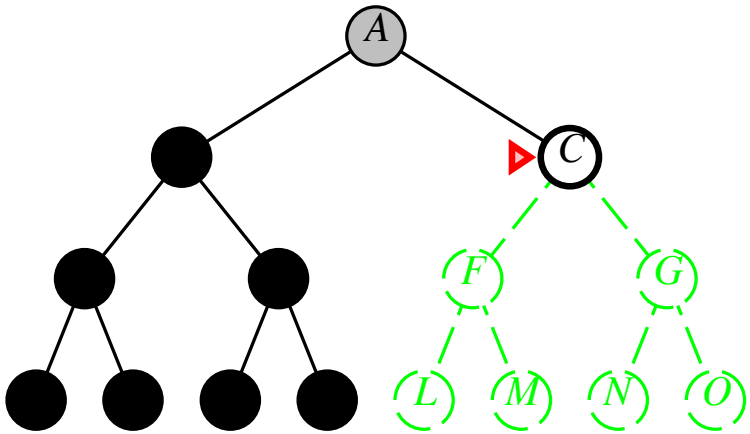
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand K to no children.
Forget B, E, K.
frontier = [C]

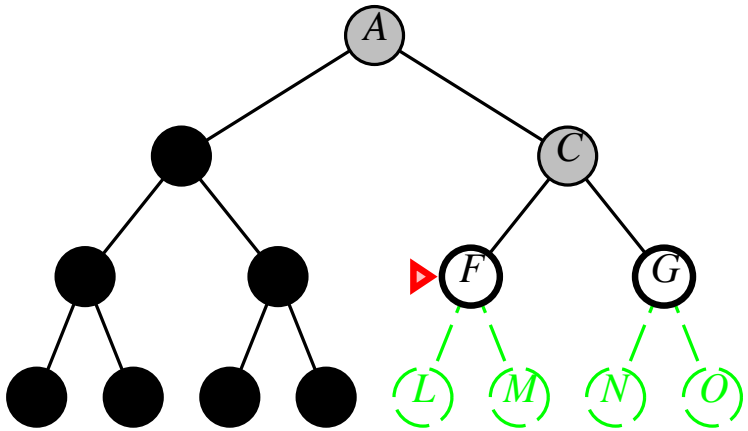
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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



Expand C to F, G.

Put F, G at front of queue.
frontier $= [F, G]$

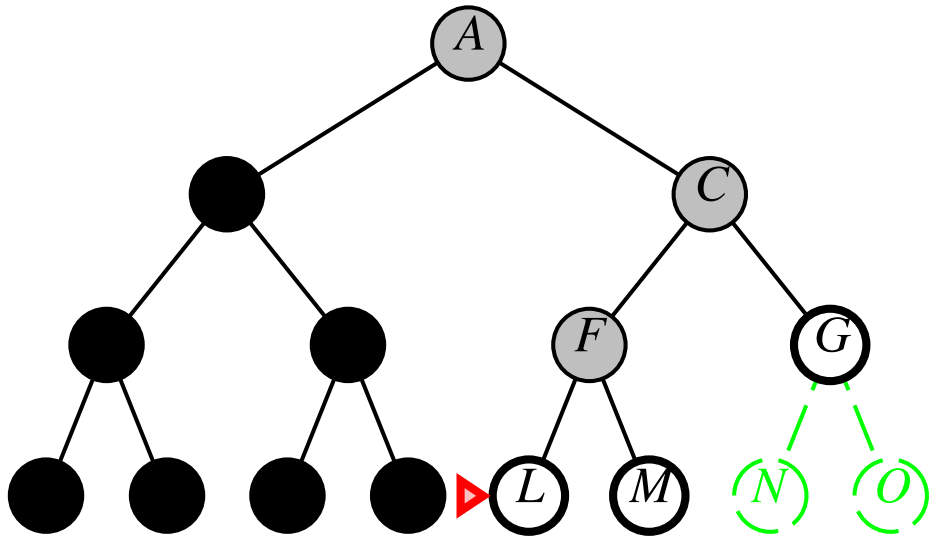
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes

Depth-first search

Expand deepest unexpanded node

Implementation:

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

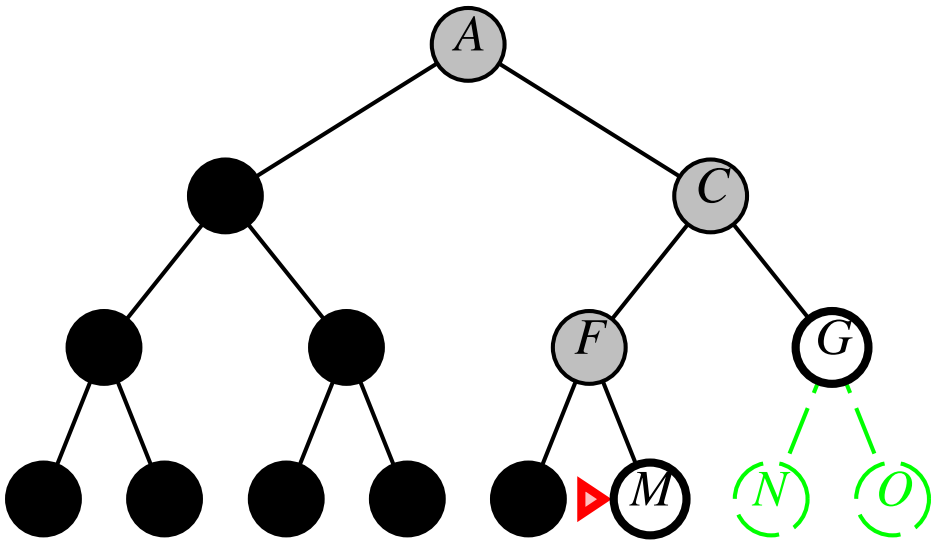


Depth-first search

Expand deepest unexpanded node

Implementation:

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



Properties of depth-first search

Complete?? No: tree search fails in infinite-depth spaces, e.g., spaces with loops

1. Modify to avoid repeated states along path
   Check if current nodes occurred before on path to root.
   No extra memory cost but it avoids only infiite loops, not redundant paths.
2. Can use graph search (remember all nodes ever seen).
   It is complete in finite spaces.
   Problem with graph search: space is exponential, not linear
   Still fails in infinite-depth spaces (may miss goal)

Properties of depth-first search

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

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

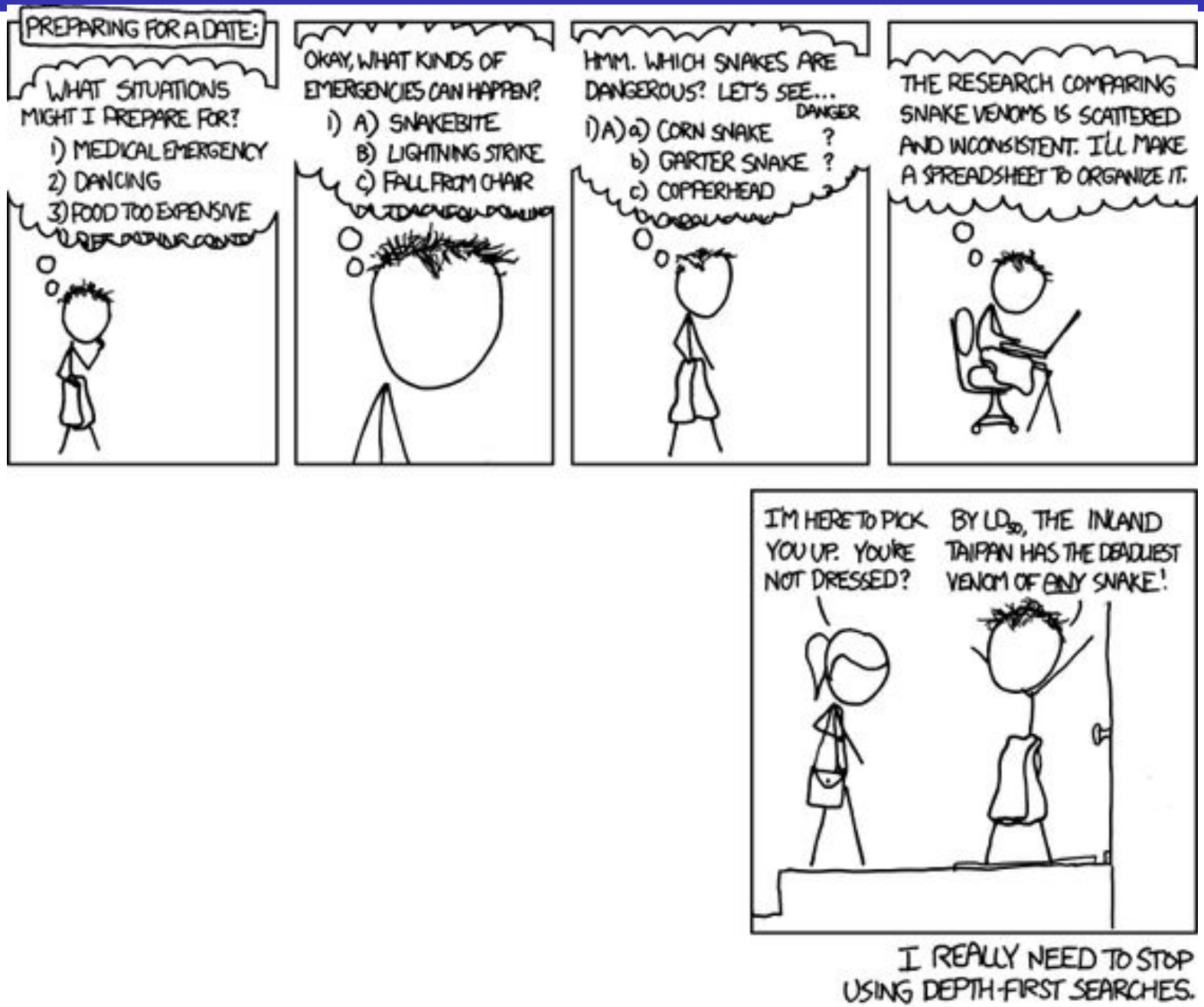
Time?? $O(b^m)$: terrible if m is much larger than d

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

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

Cost optimal?? No

DFS


PREPARING FOR A DATE:
WHAT SITUATIONS MIGHT I PREPARE FOR?
1) MEDICAL EMERGENCY
2) DANCING
3) FOOD TOO EXPENSIVE
OKAY, WHAT KINDS OF EMERGENCIES CAN HAPPEN?
1) A) SNAKEBITE
B) LIGHTNING STRIKE
C) FALL FROM CHAIR
HMM. WHICH SNAKES ARE DANGEROUS? LET'S SEE...
1) A) a) CORN SNAKE ?
b) GARTER SNAKE ?
c) COPPERHEAD
DANGER
THE RESEARCH COMPARING SNAKE VENOMS IS SCATTERED AND INCONSISTENT. I'LL MAKE A SPREADSHEET TO ORGANIZE IT.
I'M HERE TO PICK YOU UP. YOU'RE NOT DRESSED?
BY LD_{50}, THE INLAND TAIPAN HAS THE DEADLIEST VENOM OF ANY SNAKE!
I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

Depth-limited search

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors

Recursive implementation:

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
```

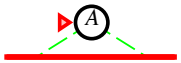
Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

Iterative deepening search $l = 0$

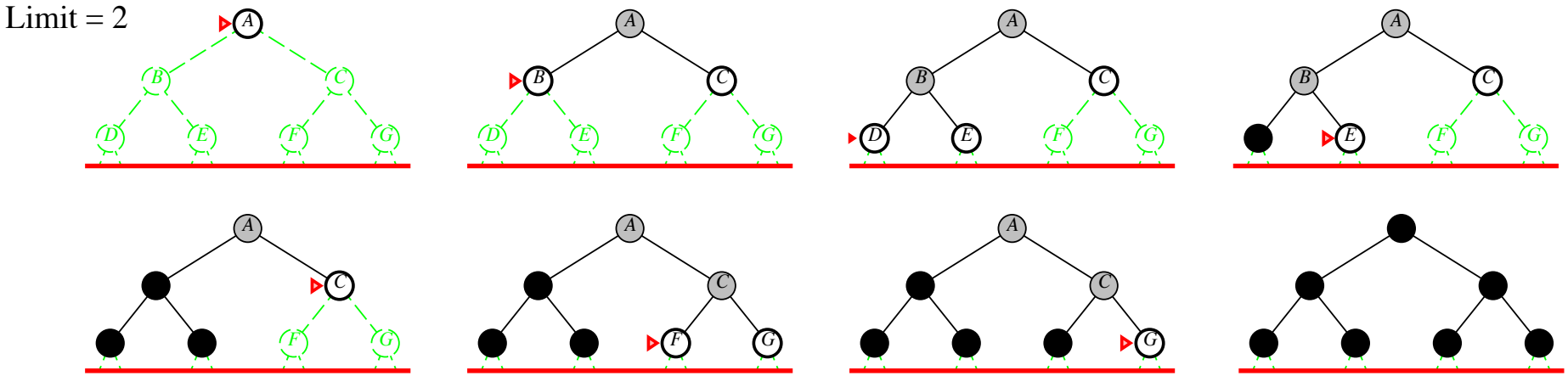
Limit = 0



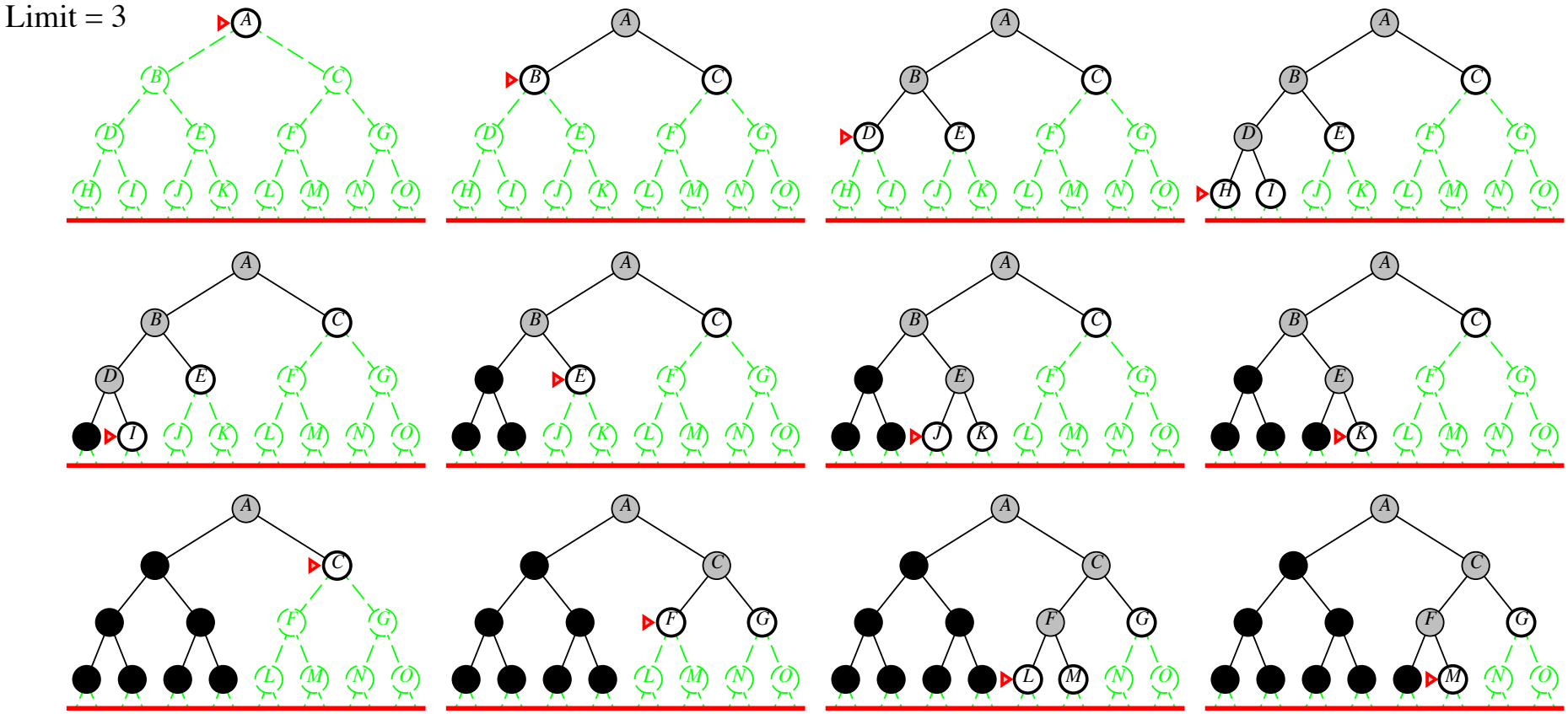
Iterative deepening search $l = 1$

Limit = 1

Iterative deepening search $l = 2$



Iterative deepening search $l = 3$



Properties of iterative deepening search

Complete?? Yes

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

Space?? $O(bd)$

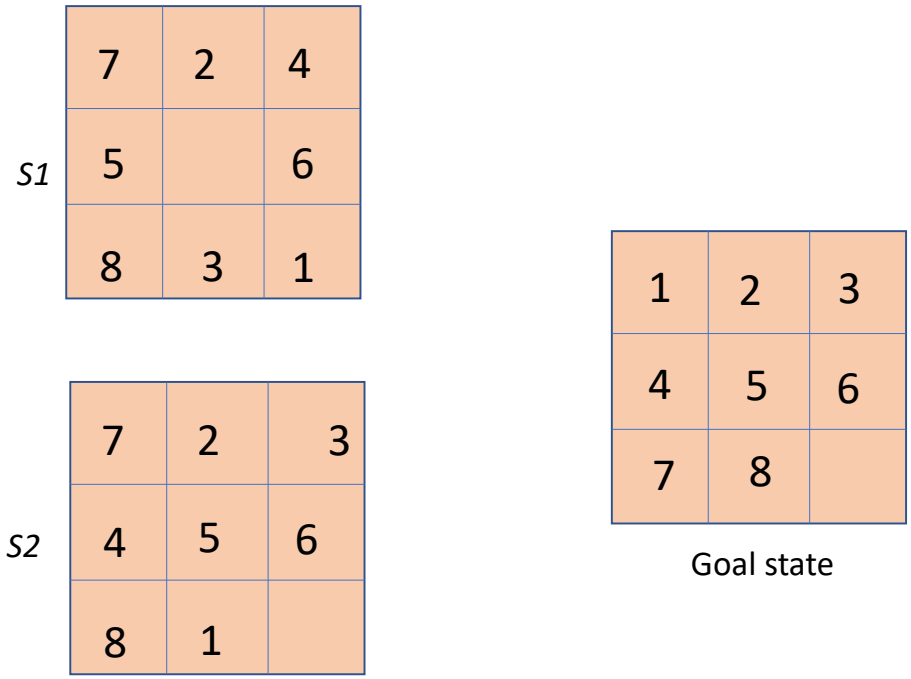
Cost optimal?? Yes, if step cost are equal

Uninformed strategies - summary of the properties

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

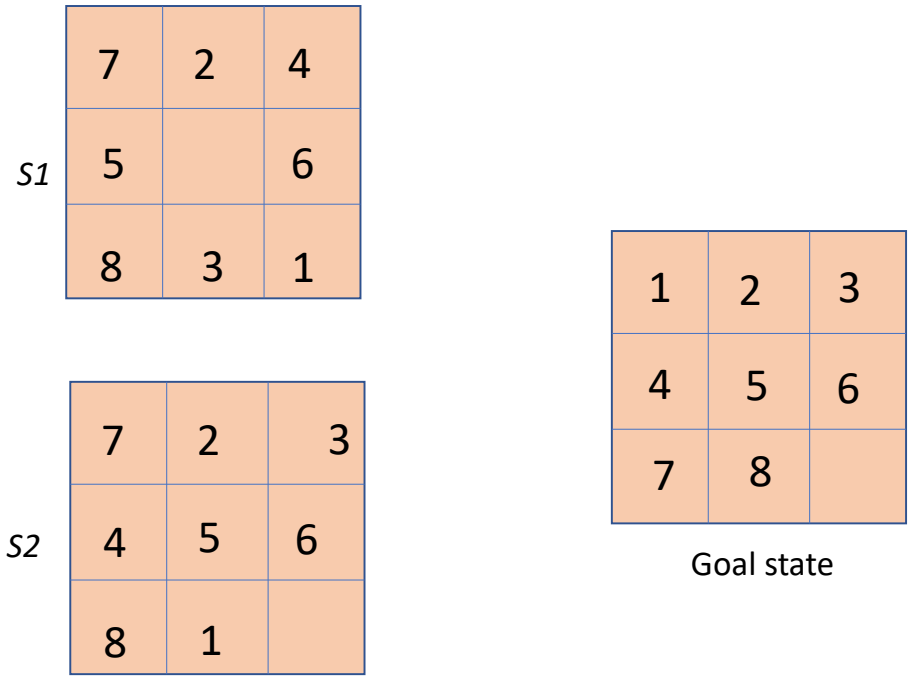
Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

Uninformed vs informed search



- For an uninformed search strategy , states S1 and S2 are two indifferent nodes (e.g., at the same level in the search tree
- Uninformed search strategies uses only the position of the nodes in the tree, not the state descriptions

Uninformed vs informed search



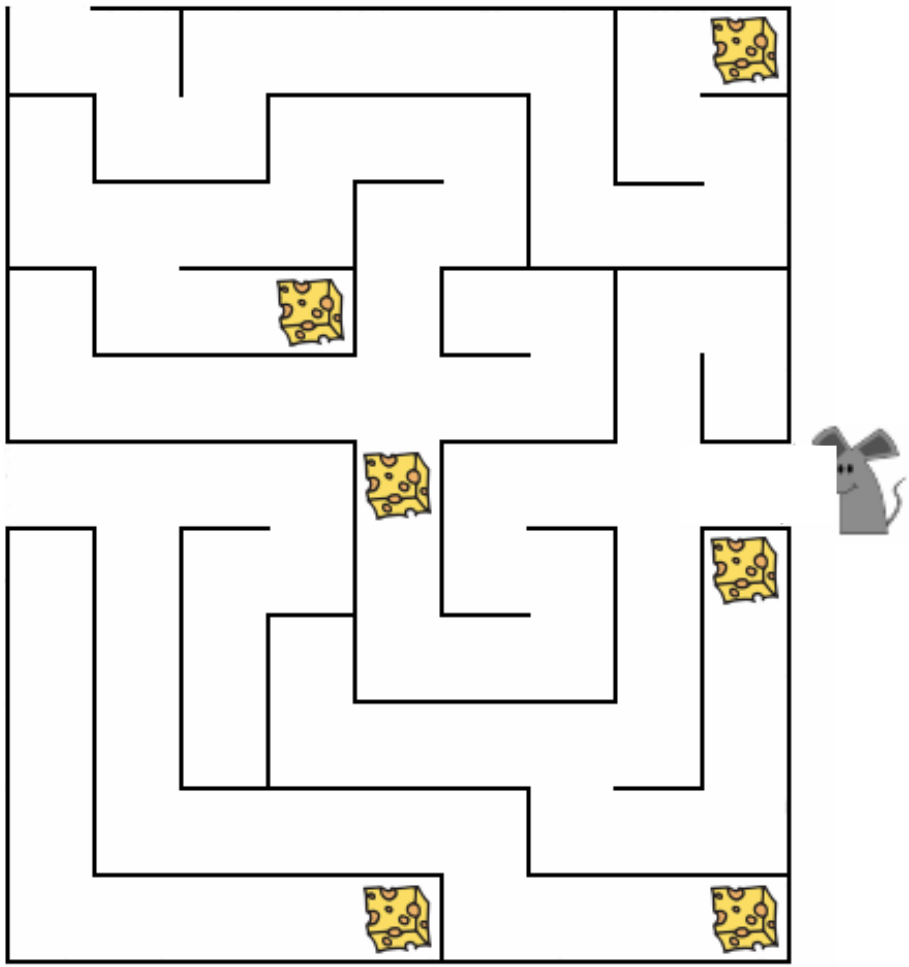
- S1 is more promising than S2 for an informed (heuristic) search strategy that counts the number of misplaced tiles
- Heuristic strategies use the information in the state description.

Uninformed Search disadvantageous

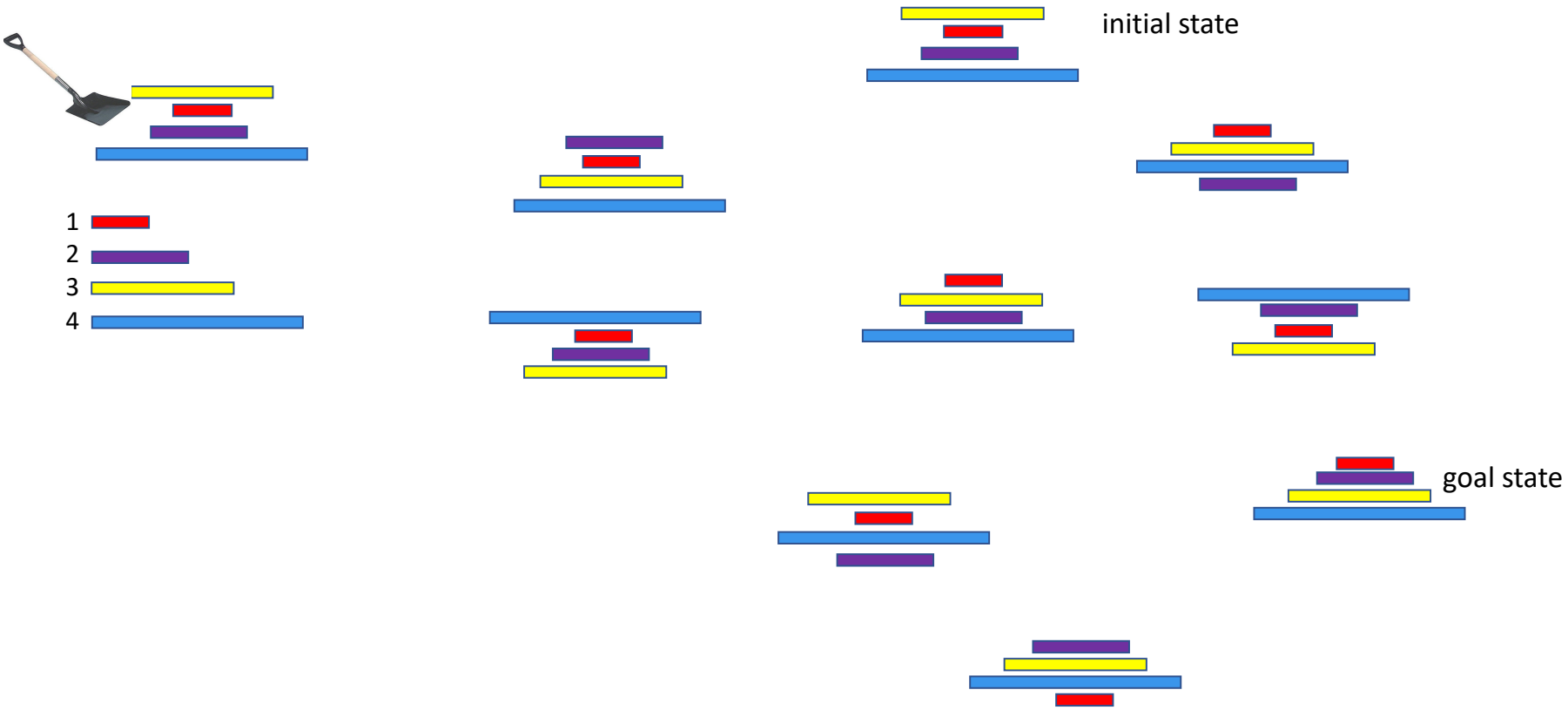
- systematically searching the search space blindly- not questioning where the goal may be in the space
- search space is often very large. Time/space problems with such exhaustive search - think of chess.
- in such situations, better to use algorithms which do a more informed search

Heuristic

Informed Search algorithms uses "heuristics".
What is a heuristic?

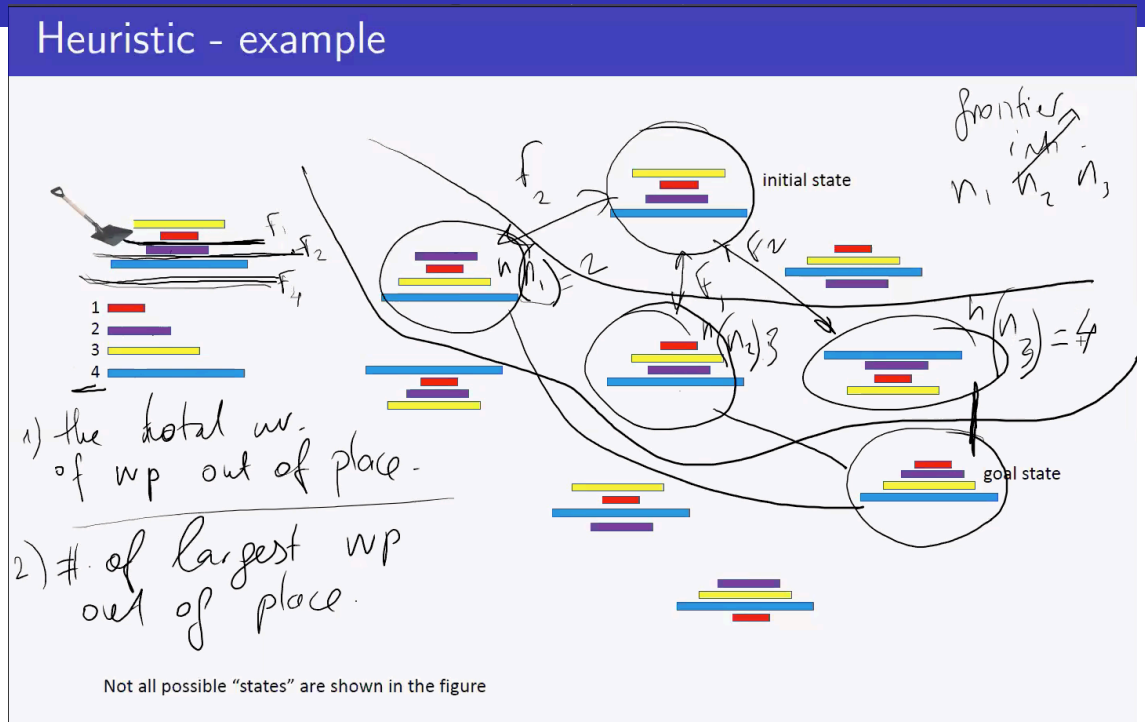


Heuristic - example



Not all possible “states” are shown in the figure

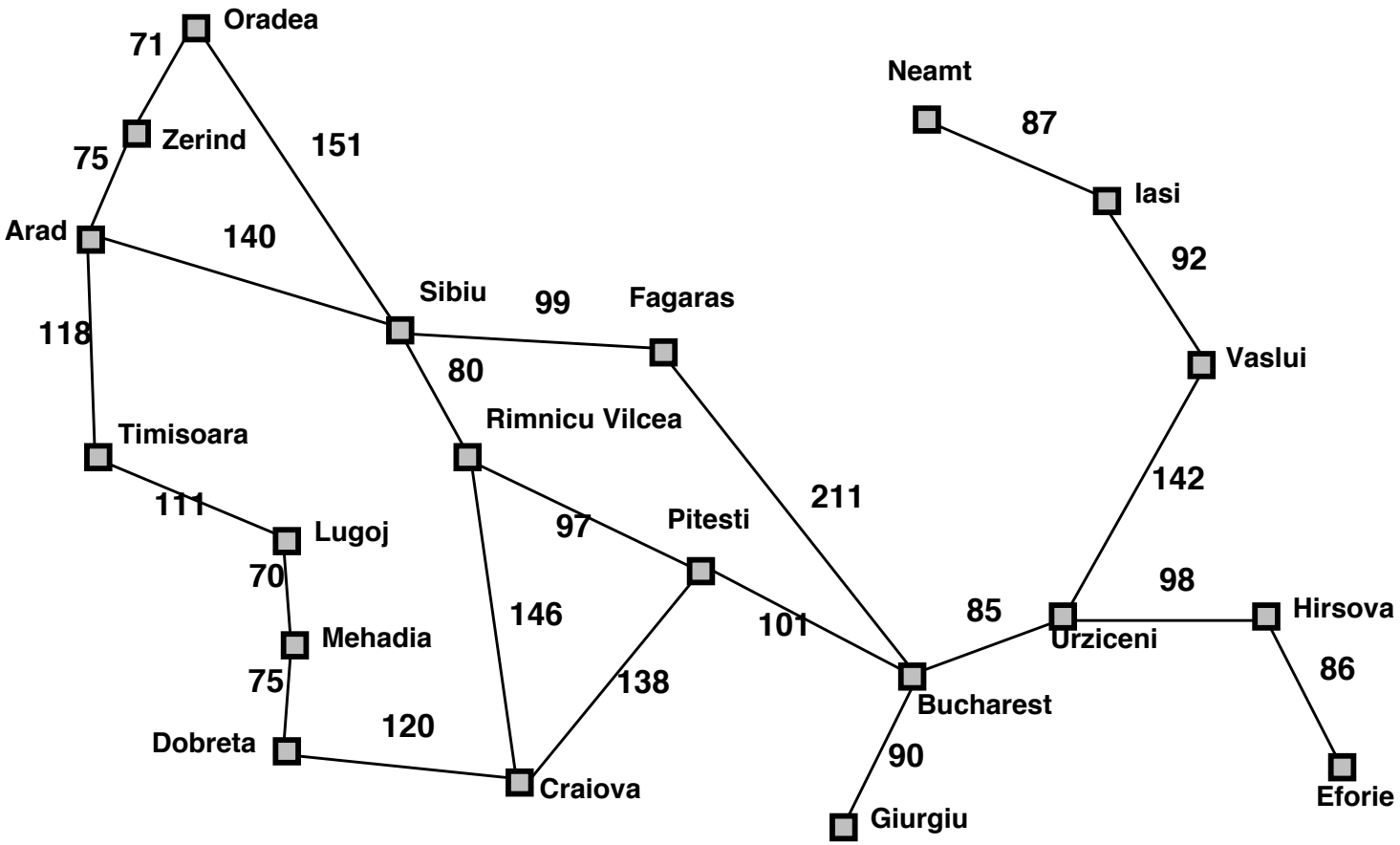
Heuristic - example



Wp: wood piece
Find the sequence of actions that brings the environment fro the initial state to the goal state. The agent can use a showel to flip the wood pieces.
The numbers show the color (and indeed size of the wood pieces). For example 2 is the purple wp and it is the second smallest wp.

Actions: F1, F2,.... these are Flipping actions.
heuristic 1: the total nr of wp out of place
heuristic 2: the nr of largest wp out of place (the nr is the one that shows the color of wps. The nr of the largest wp out of place in the initial state is 3, the yellow wp.

Heuristic-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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Heuristic Search Algorithms

Today we look at two well-known informed search algorithms:

- Greedy Best First search
- A*

Greedy best-first search

Idea: use an evaluation function for each node
– estimate of "desirability"

$\Rightarrow$ Expand most desirable unexpanded node

Implementation:
frontier is a queue sorted in decreasing order of desirability

Special cases:
Greedy search
A* search

Greedy search

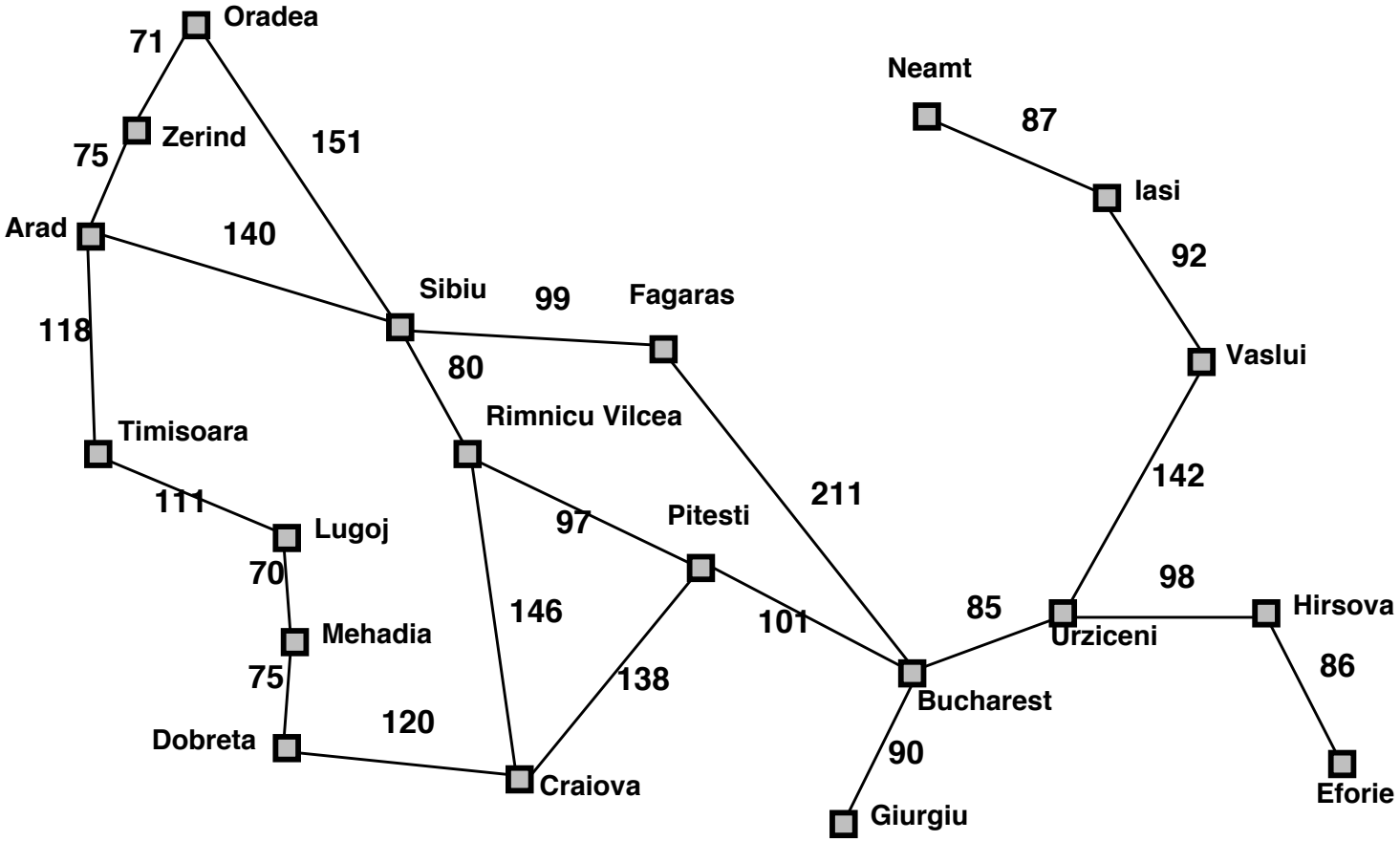
Evaluation function $= h(n)$ (heuristic)

estimate of cost from n to the closest goal

E.g., $h_{\mathrm{SLD}}(n) =$ straight-line distance from n to Bucharest

Greedy search expands the node that appears to be closest to goal

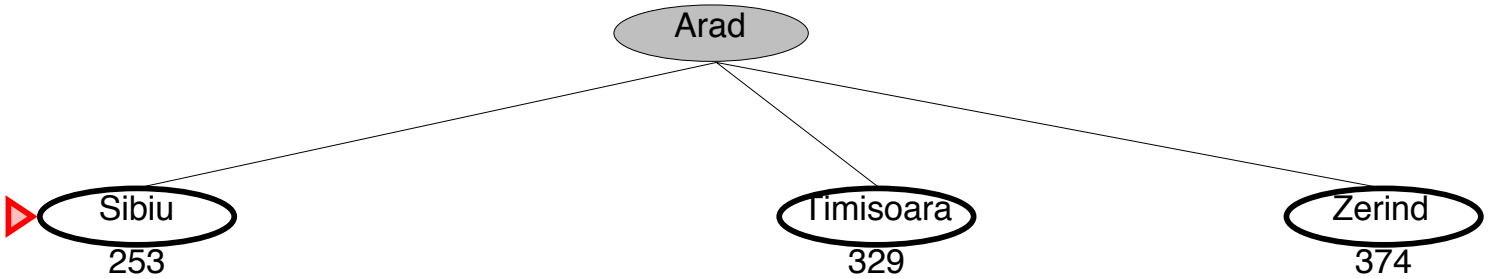
Greedy Search, Romania Example

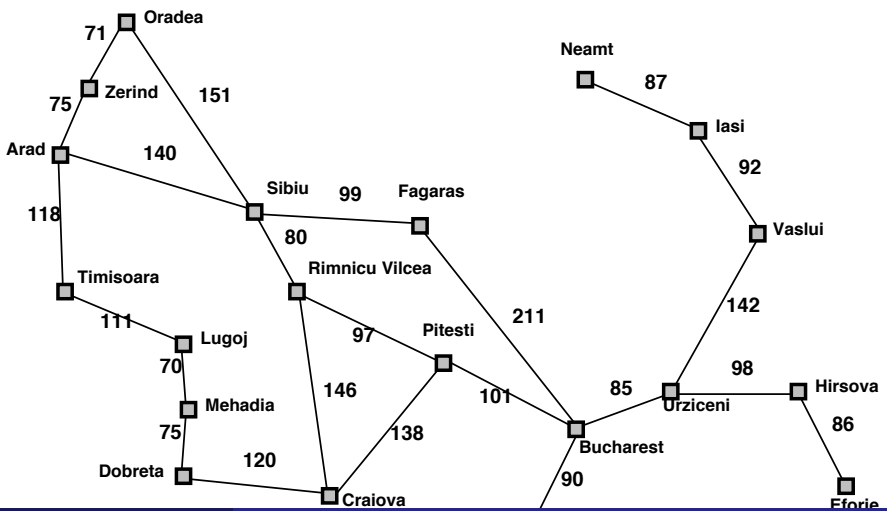


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search for Romania example

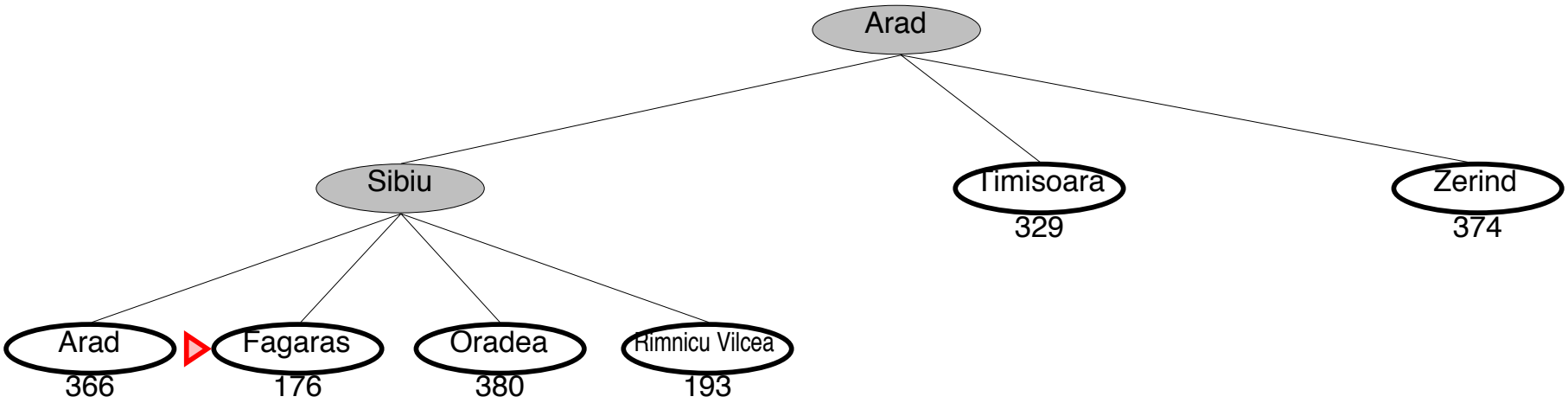
Greedy search example

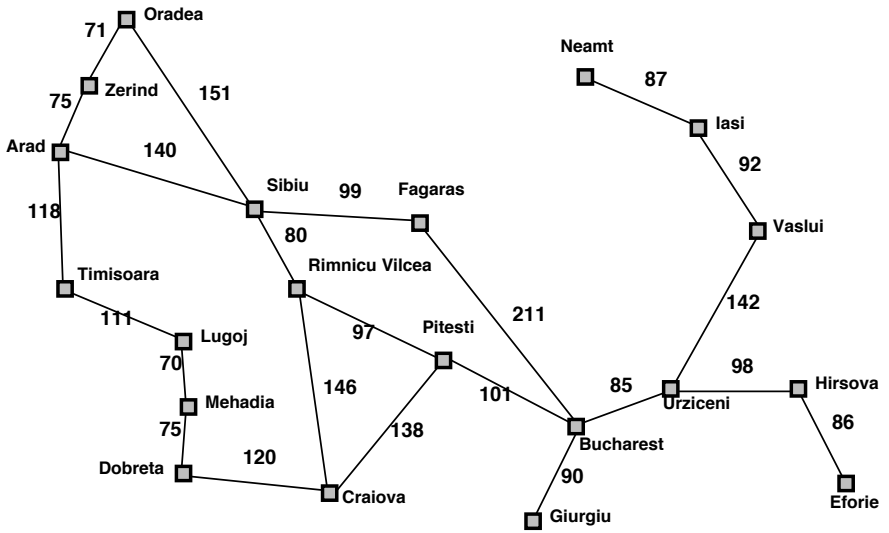




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199

Greedy search example

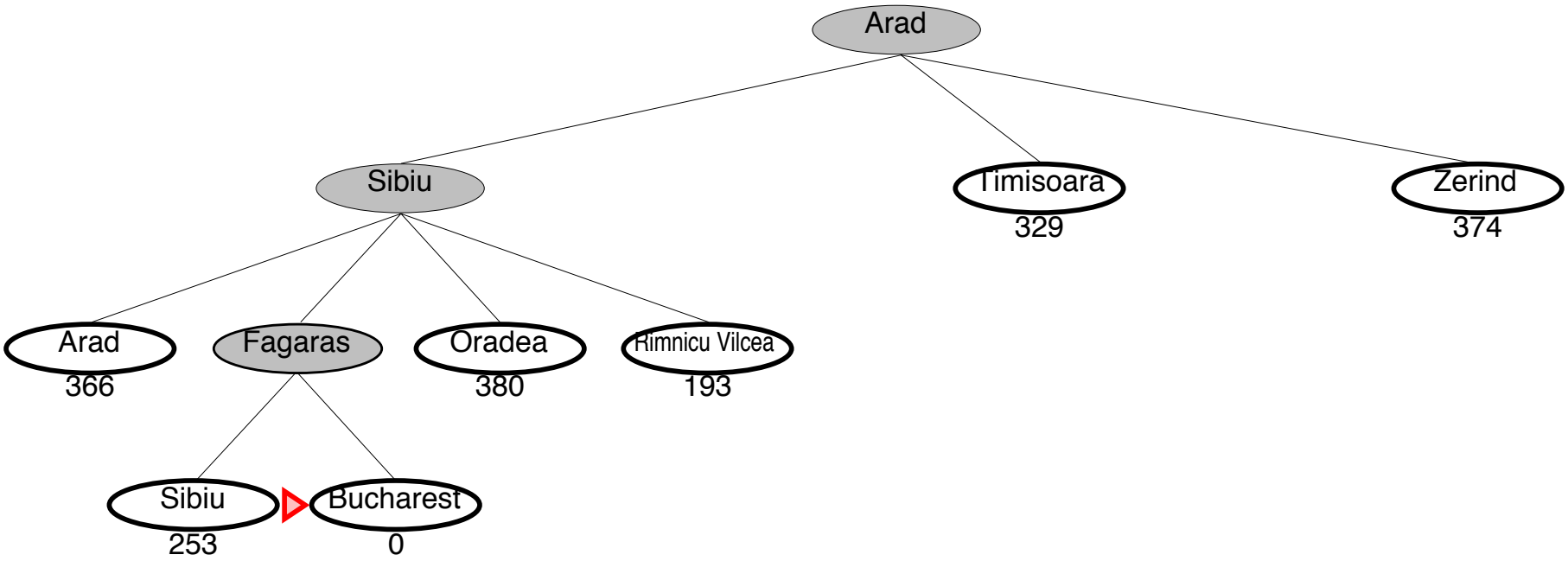


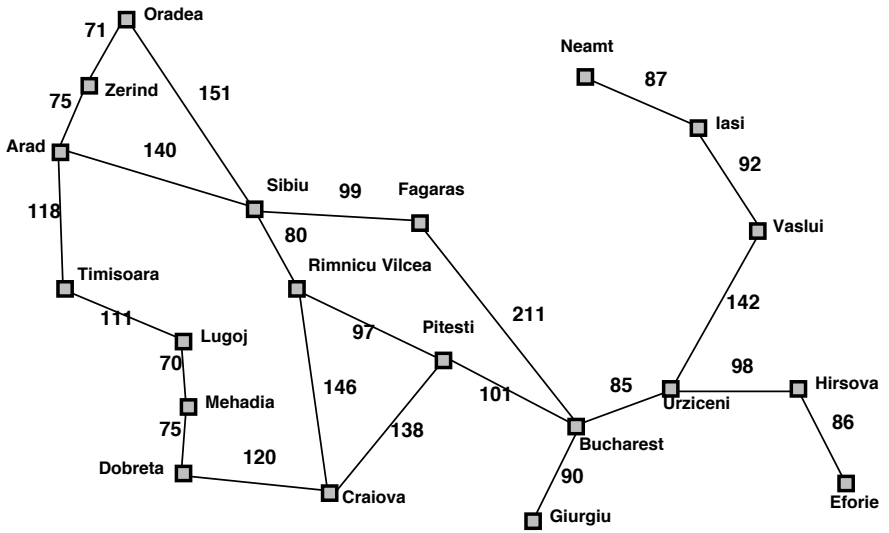


Straight–line distance to Bucharest

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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search example



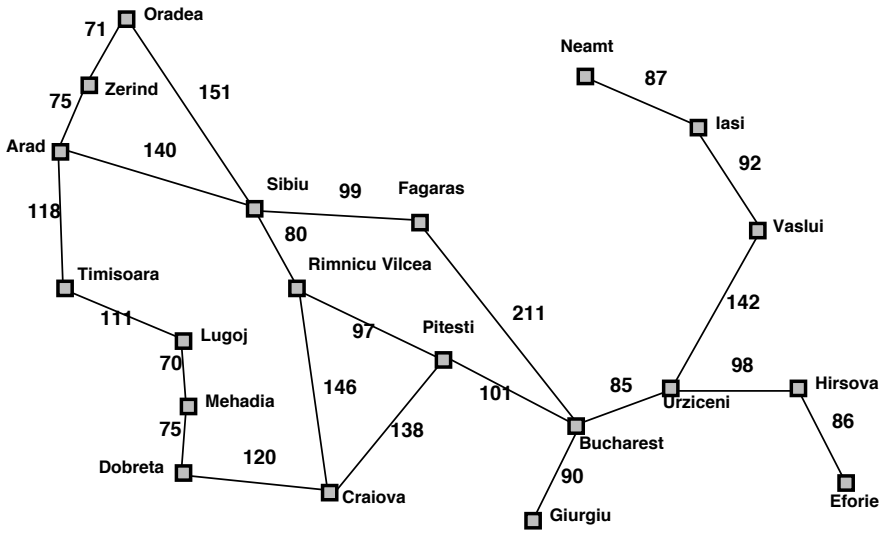


Straight–line distance to Bucharest

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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy search

Complete?? No for tree search –can get stuck in loops, e.g., assume getting from Iasi to Fagaras,
Straight line distance-wise, Neamt is closest to Fagaras.
Hence, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ ...
Complete in graph version (i.e., with repeated-state checking) for finite spaces)



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

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., assume getting from Iasi to Fagaras,
Straight line distance-wise, Neamt is closest to Fagaras.
Hence, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ ...
Complete in graph version (i.e., with repeated-state checking) for finite spaces)

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., assume getting from Iasi to Fagaras,
Straight line distance-wise, Neamt is closest to Fagaras.
Hence, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ ...
Complete in graph version (i.e., with repeated-state checking) for finite spaces)

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal??

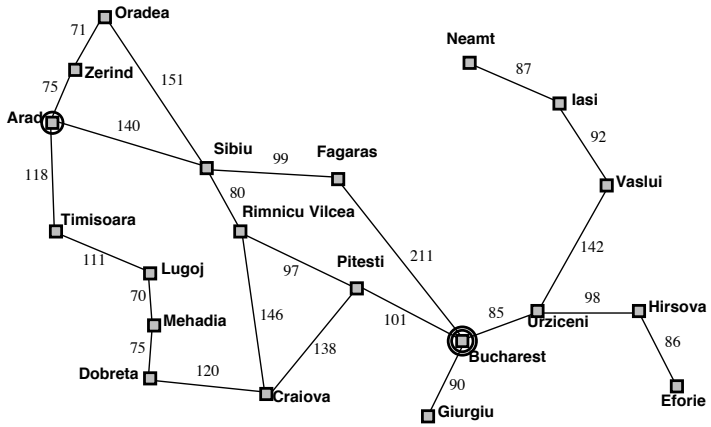
Properties of greedy search

Complete?? Not in tree version – can get stuck in loops, e.g., assume getting from Iasi to Fagaras. Straight line distance-wise, Neamt is closest to Fagaras. Hence, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ ...
Complete in graph version for finite spaces.

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No. E.g., the path it found (Sibiu-Fagaras-Bucharest) is longer than the path Sibiu-Rimnicu Vilcea- Pitesti -Buchares.



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	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

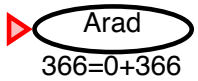
A* search

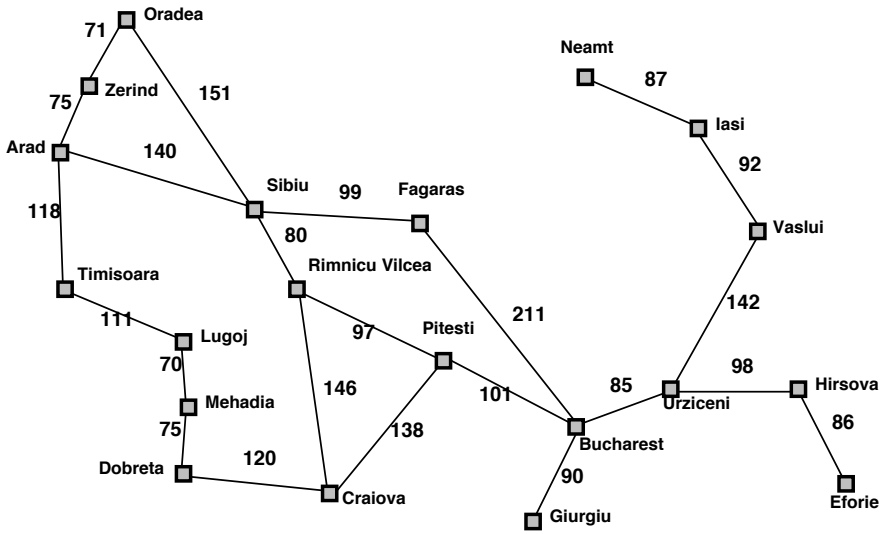
Idea: avoid expanding paths that are already expensive

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

$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost of the cheapest path from n to goal node
$f(n)$ = estimated cost of the cheapest solution through n to goal

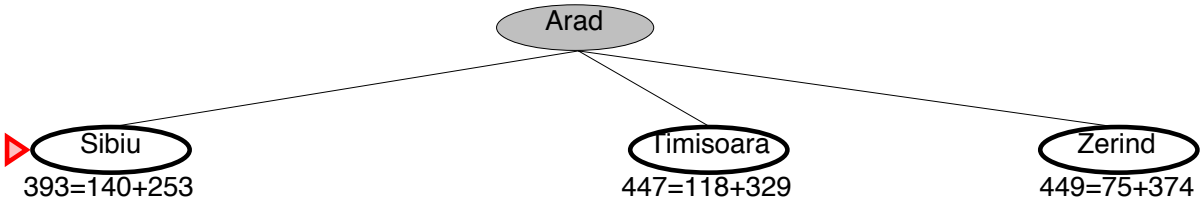
A* search example

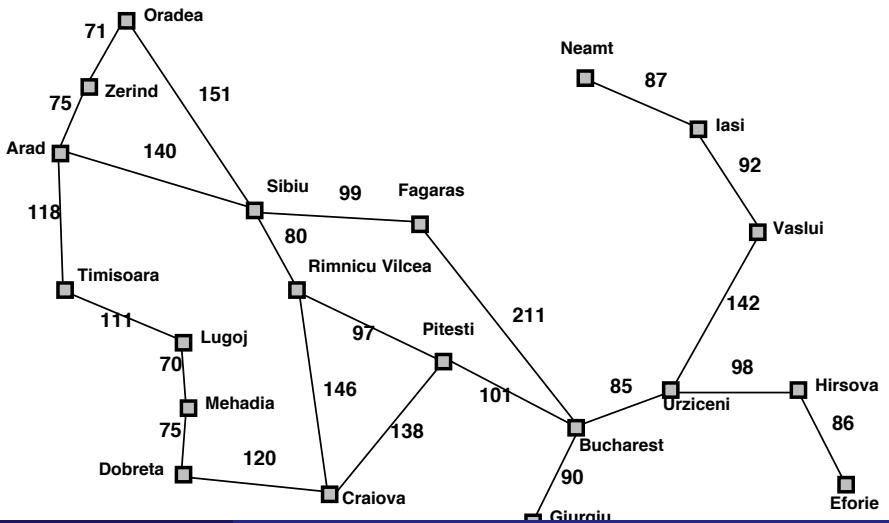




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

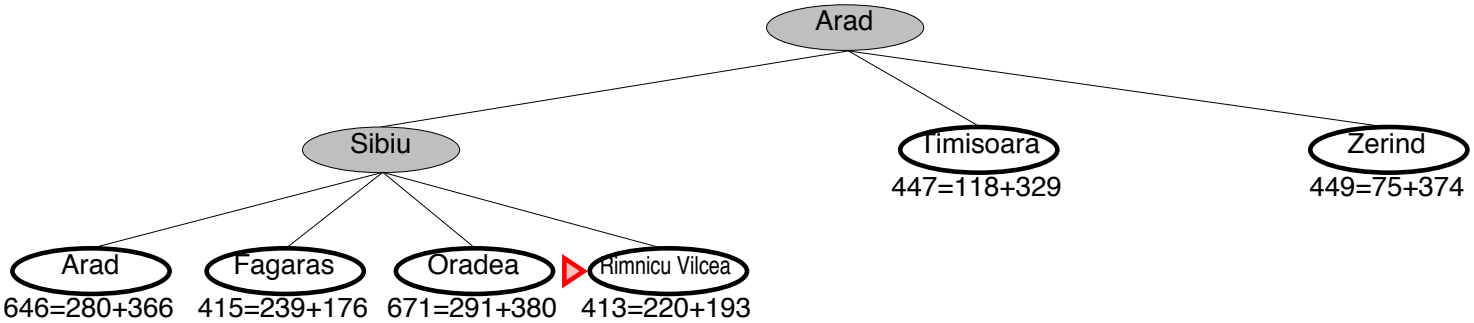


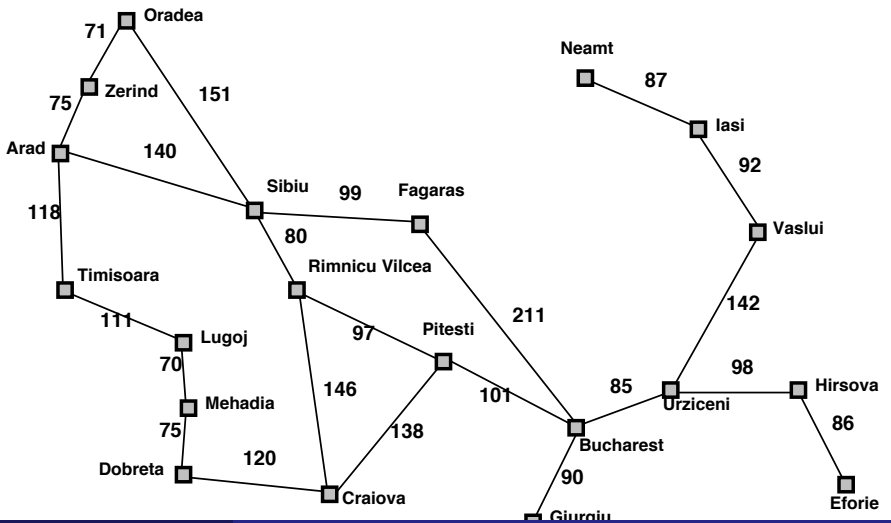


Straight–line distance to Bucharest

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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	

A* search example

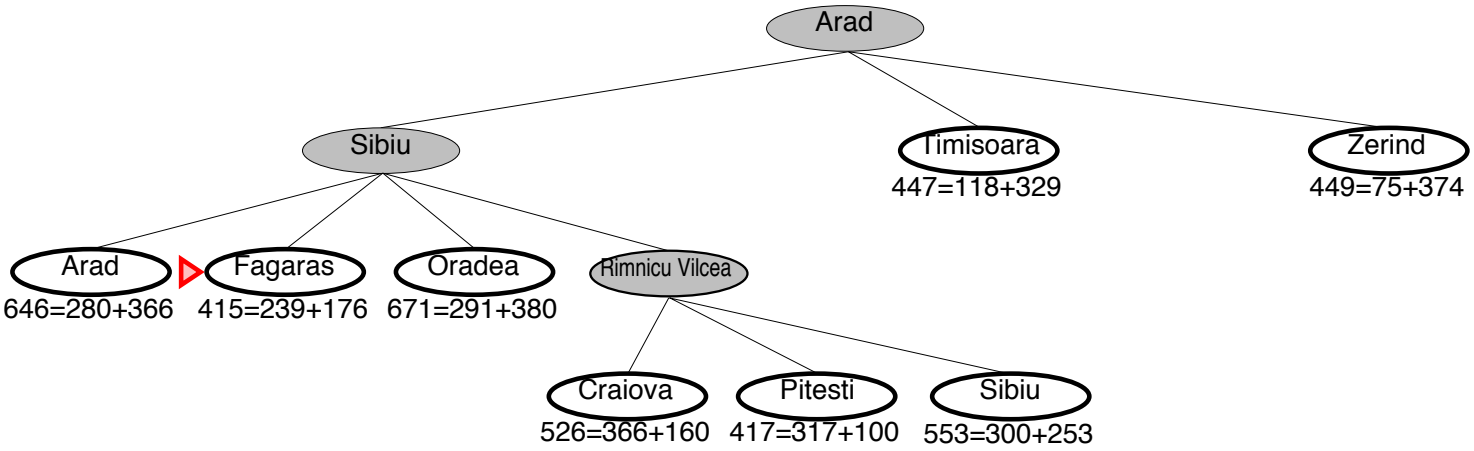




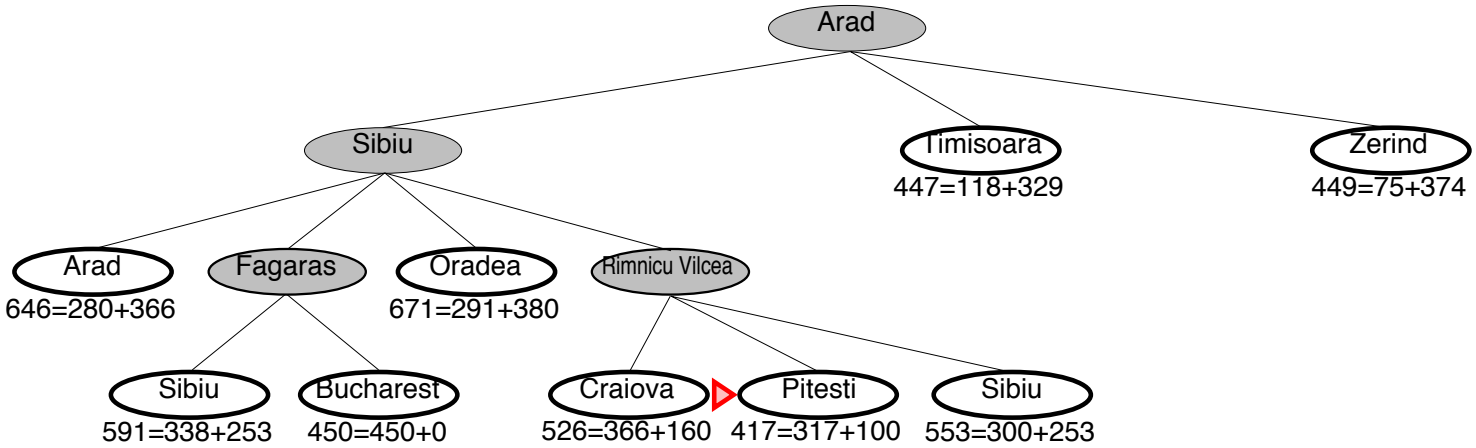
Straight–line distance to Bucharest

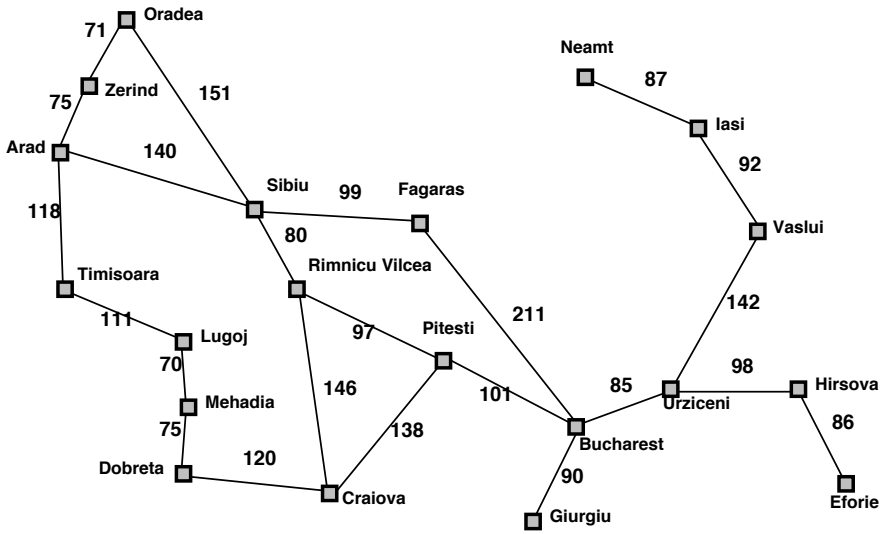
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	

A* search example



A* search example

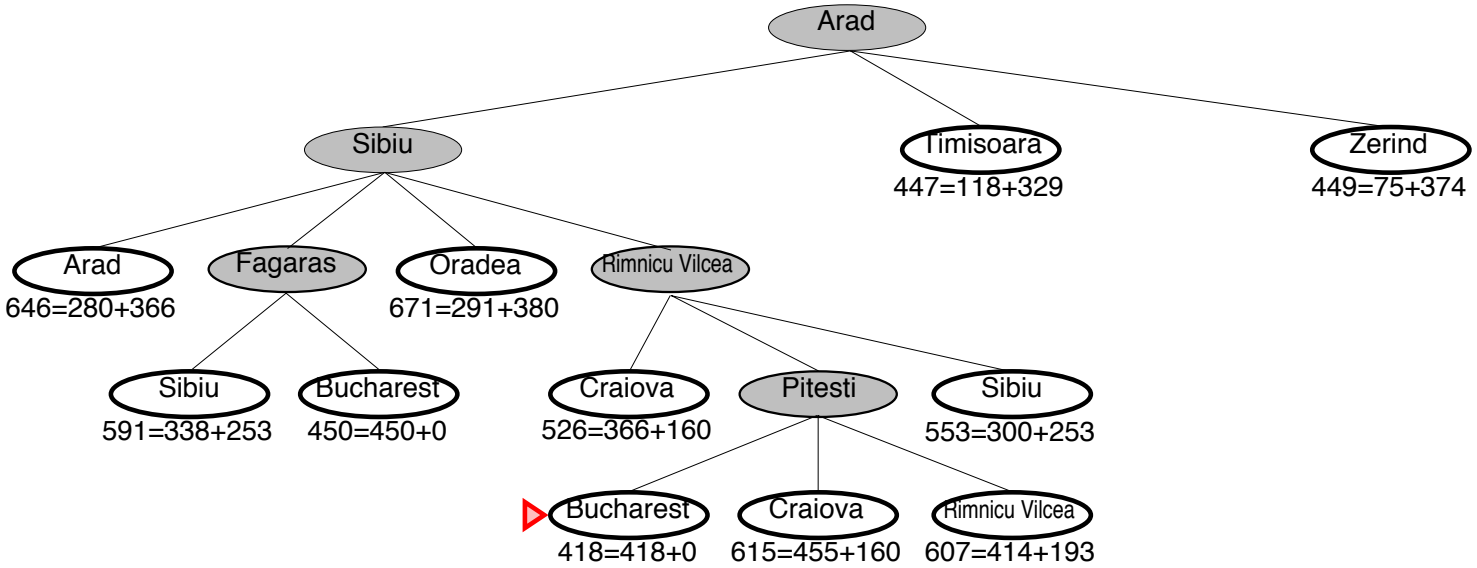


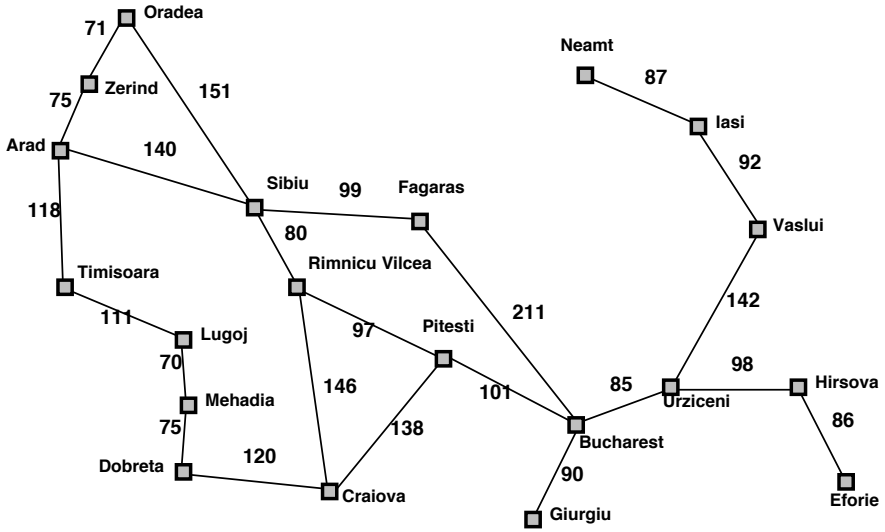


Straight–line distance to Bucharest

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	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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Optimal path?

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$.
I.e., if all step costs are $> \epsilon$ and b is finite.

Time??