Problem Solving by Search

CHAPTER 3

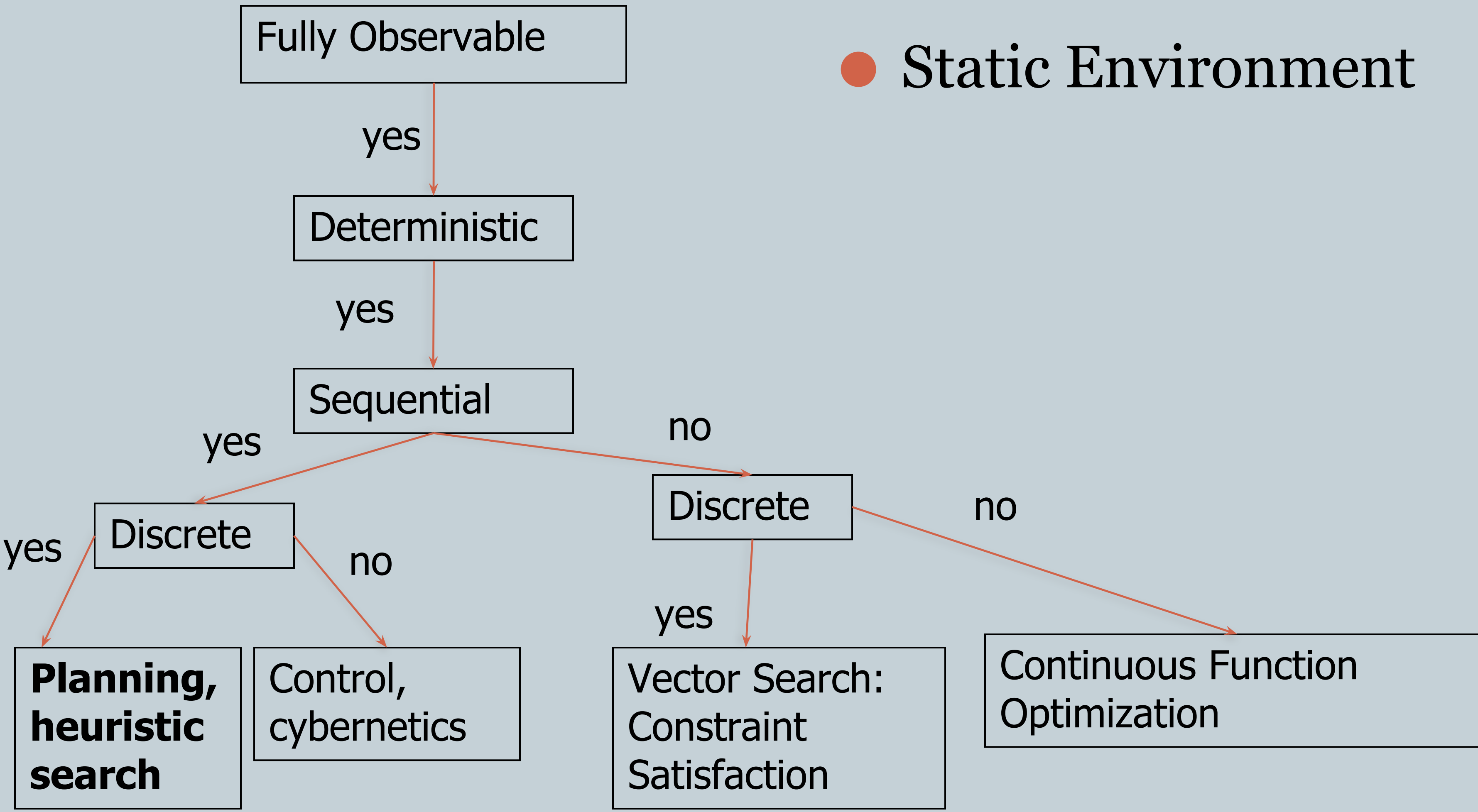
Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Global Edition 3/E

Outline

- Problem formulation: representing sequential problems.
- Example problems.
- Planning for solving sequential problems without uncertainty.
- Basic search algorithms

Environment Type Discussed In this Lecture

- Static Environment



Choice in a Deterministic Known Environment

- Without uncertainty, choice is trivial in principle: choose what you know to be the best option.
- Trivial if the problem is represented in a look-up table.

Option	Value
Chocolate	10
Coffee	20
Book	15

This is the standard problem representation in decision theory (economics).

Computational Choice Under Certainty

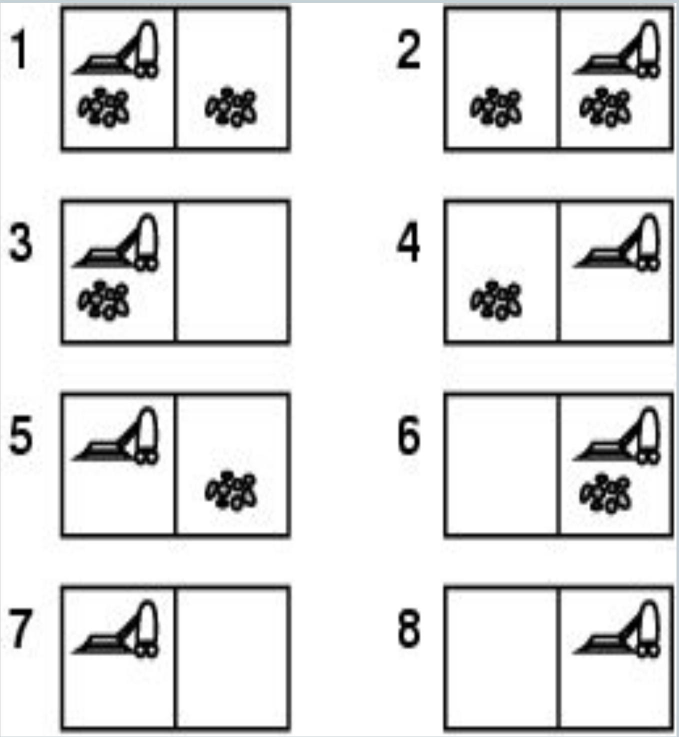
- But choice can be *computationally* hard if the problem information is represented differently.
- Options may be **structured** and the best option needs to be constructed.
  - E.g., an option may consist of a path, sequence of actions, plan, or strategy.
- The value of options may be given **implicitly** rather than explicitly.
  - E.g., cost of paths need to be computed from map.

Problem Types

- **Deterministic**, fully observable -> single-state problem
  - Agent knows exactly which state it will be in; solution is a **sequence**
- **Non-observable** -> conformant problem
  - Agent may have no idea where it is; solution (if any) is a **sequence**
- **Nondeterministic** and/or partially observable -> contingency problem
  - percepts provide new information about current state
    solution is a contingent plan or a policy
    often **interleave search, execution**
- **Unknown state space** -> exploration problem ("online")

Sequential Action Example

- Deterministic, fully observable: single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence
  - Vacuum world: everything observed
  - Romania: The full map is observed



- Single-state:

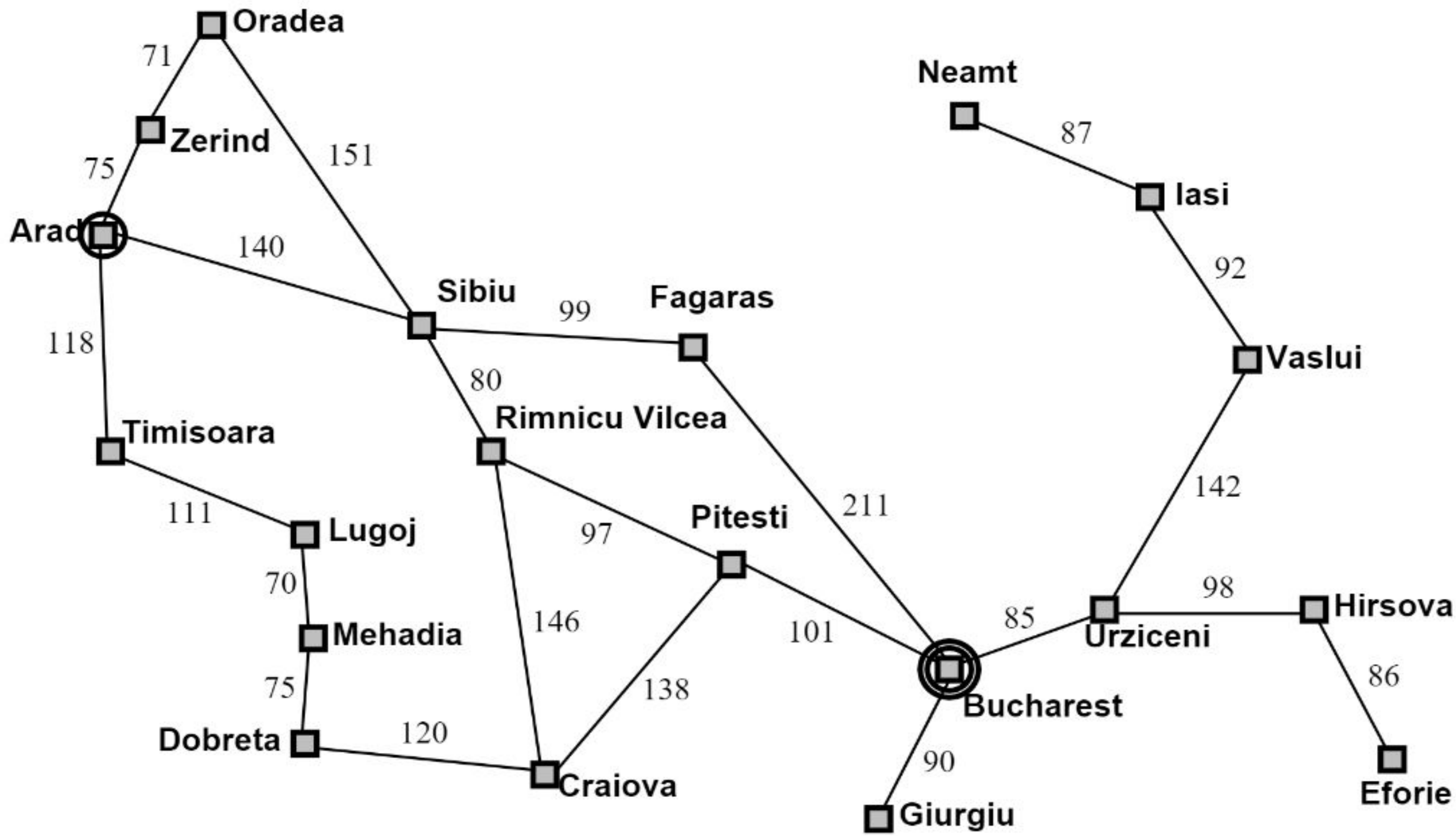
Start in #5. Solution??
  - [Right, Suck]

Example: Romania

- On holiday in Romania; currently in Arad.

- Formulate goal:
  - be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania



Abstraction: The process of removing details from a representation Is the map a good representation of the problem? What is a good replacement?

Single-state problem formulation

A problem is defined by 4 items:

- **initial state** e.g., "at Arad"
- **Successor function** S(x)= set of action–state
- **Goal test**, can be
  - explicit, e.g., x = "at Bucharest", or "checkmate" in chess
  - 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

The successor function

- Successor function: for a given state, returns a set of action/new-state pairs.
- Vacuum-cleaner world: (A, dirty, clean) → ('Left', (A, dirty, clean)),('Right', (B, dirty, clean)), ('Suck', (A, clean, dirty)), ('NoOp, (A, dirty, clean))
- Romania: In(Arad) → ((Go(Timisoara), In(Timisoara), (Go(Sibiu), In(Sibiu)), (Go(Zerind), In(Zerind))

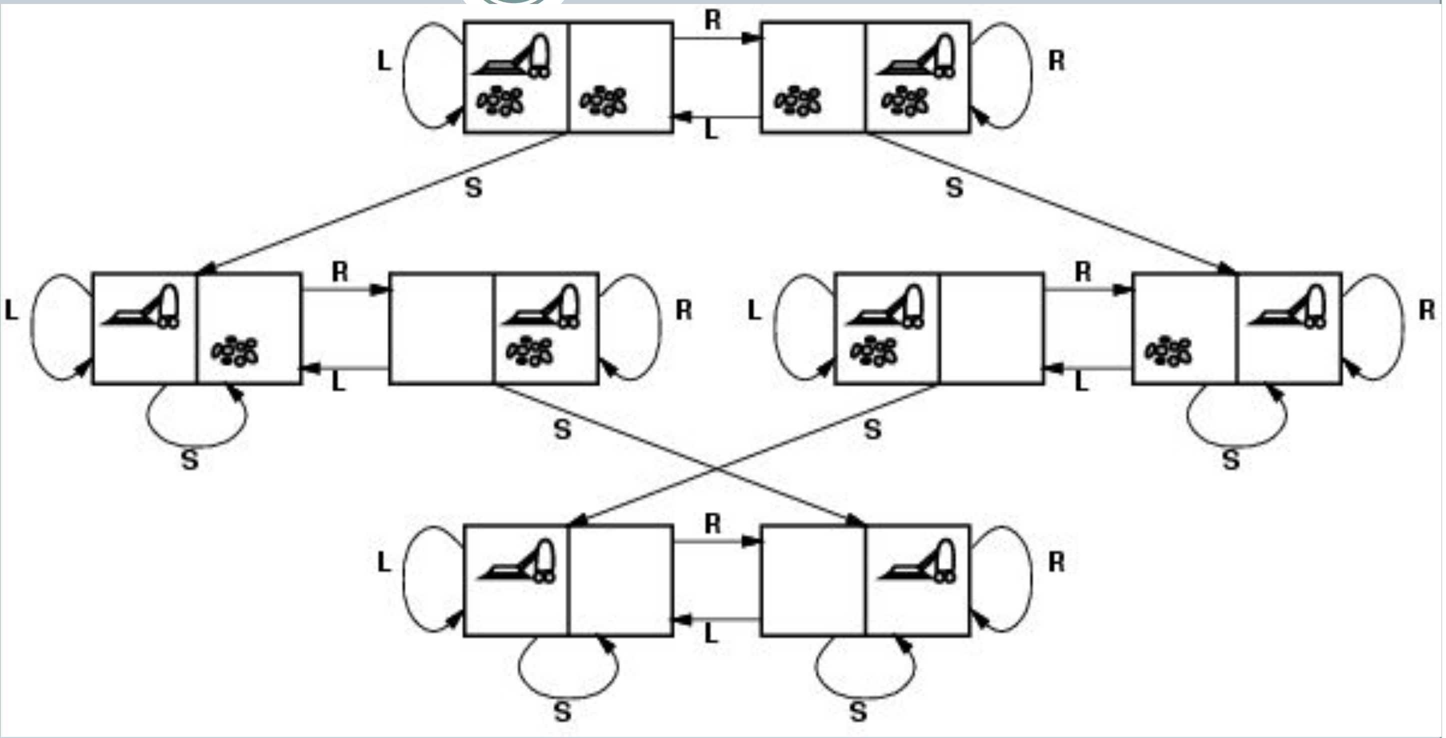
Size of space

- 8-puzzle: 9!/2 = 181, 000 states (easy)
- 15-puzzle: ~ 1.3 trillion states (pretty easy)
- 24-puzzle: ~ 1025 states (hard)
- TSP, 20 cities: 20! = 2.43 × 1018 states (hard)

Selecting a state space

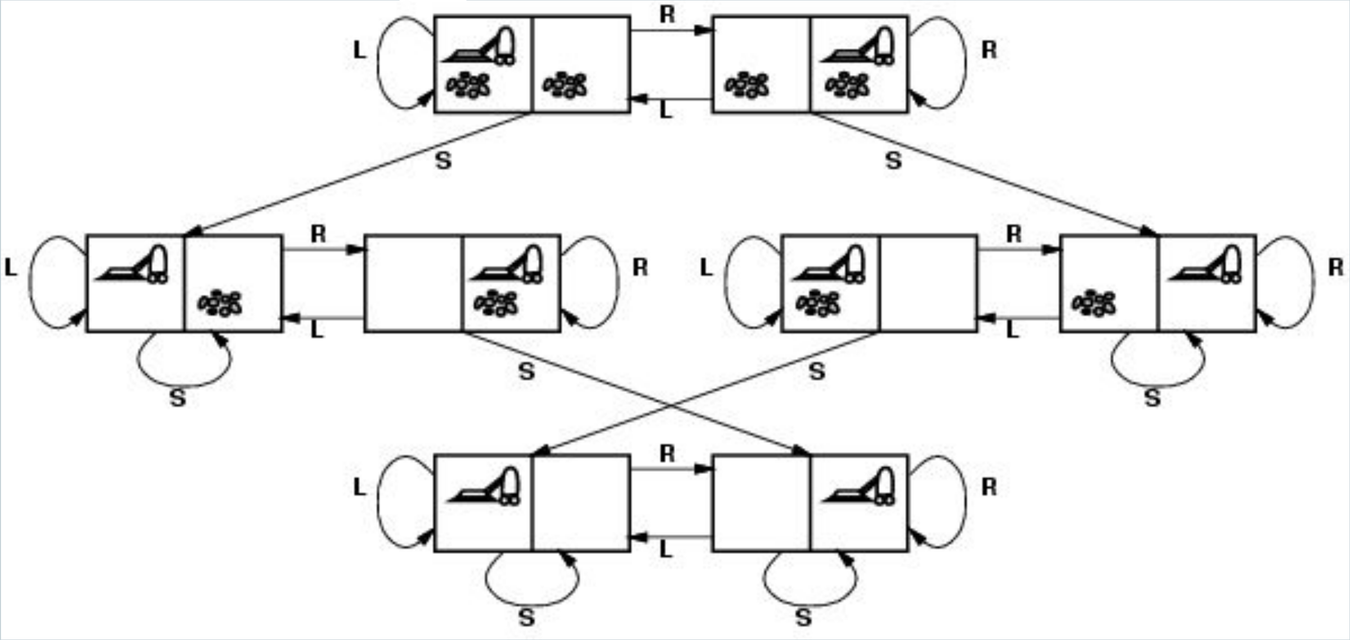
- Real world is complex
  - state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., "Arad -> Zerind" represents a complex set of possible routes, detours, rest stops, etc.
- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original problem

Vacuum world state space graph



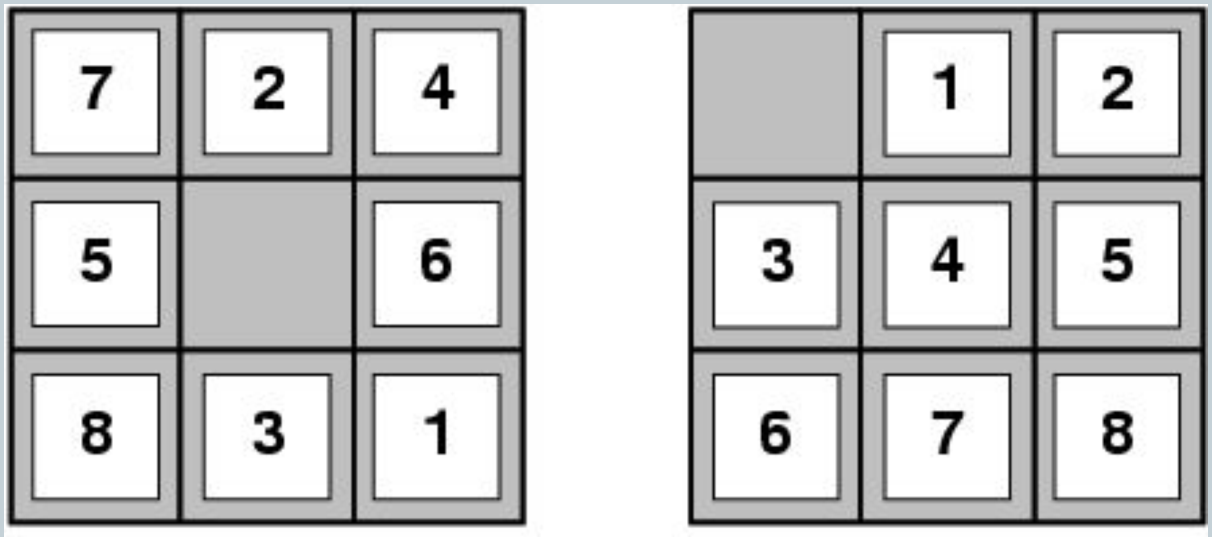
- states?
- actions?
- goal test?
- path cost?

Vacuum world state space graph



- states? integer dirt and robot location
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

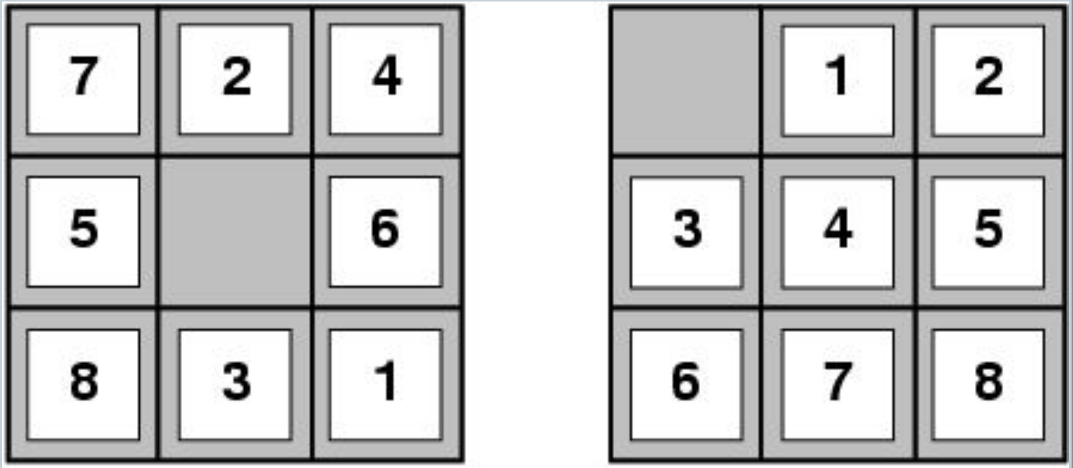
Example: The 8-puzzle



Start State Goal State

- states?
- actions?
- goal test?
- path cost?

Example: The 8-puzzle

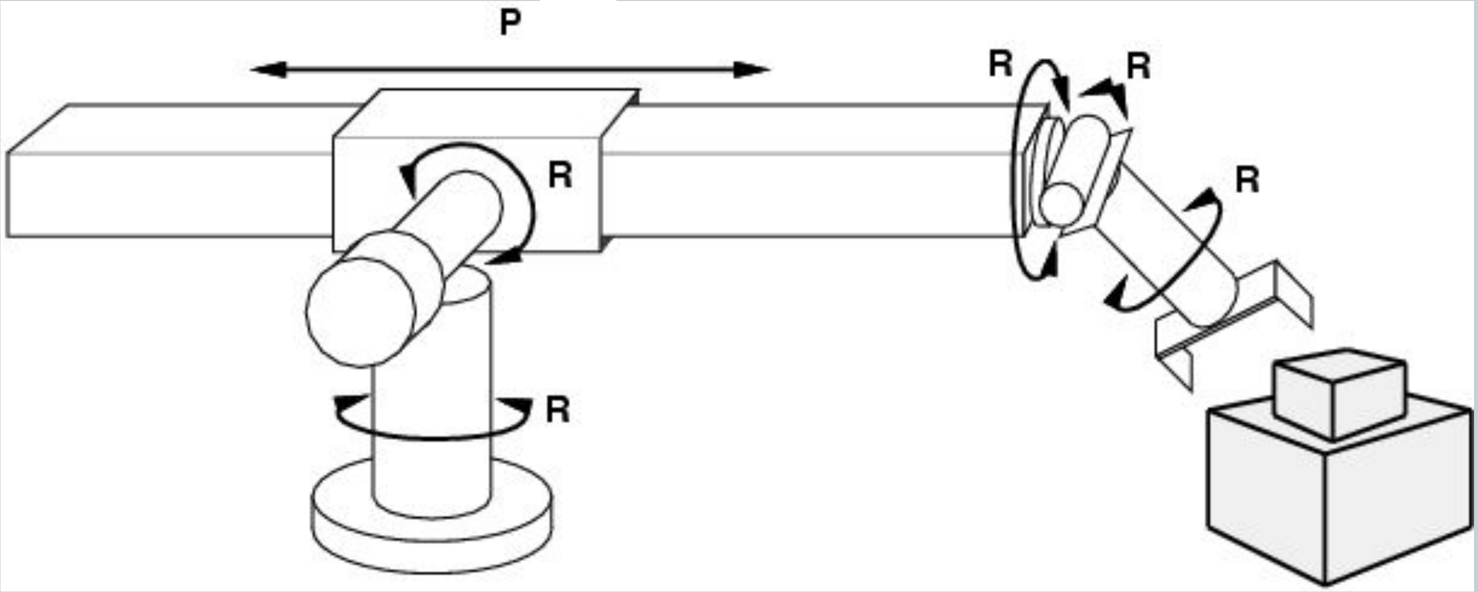


Start State   Goal State

- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: robotic assembly



- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

Problem-solving agents


```
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)
    seq ← REST(seq)
    return action
```

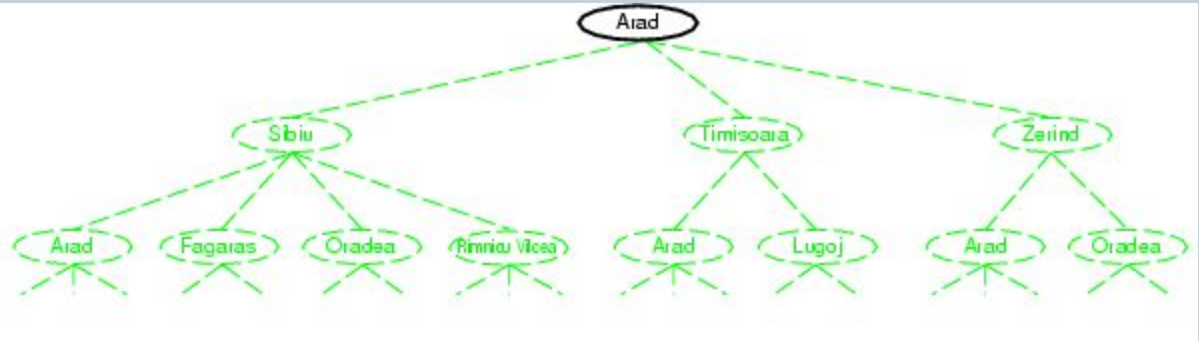
Note: this is offline problem solving; solution executed “eyes closed.”

Tree search algorithms

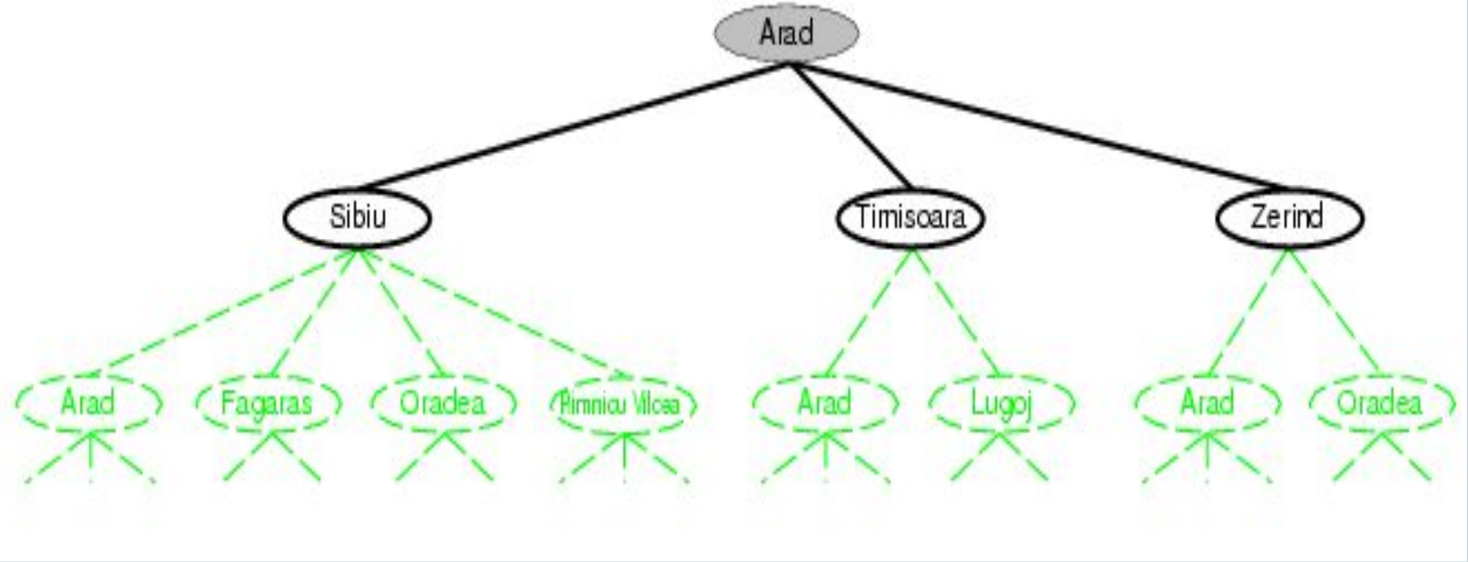
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

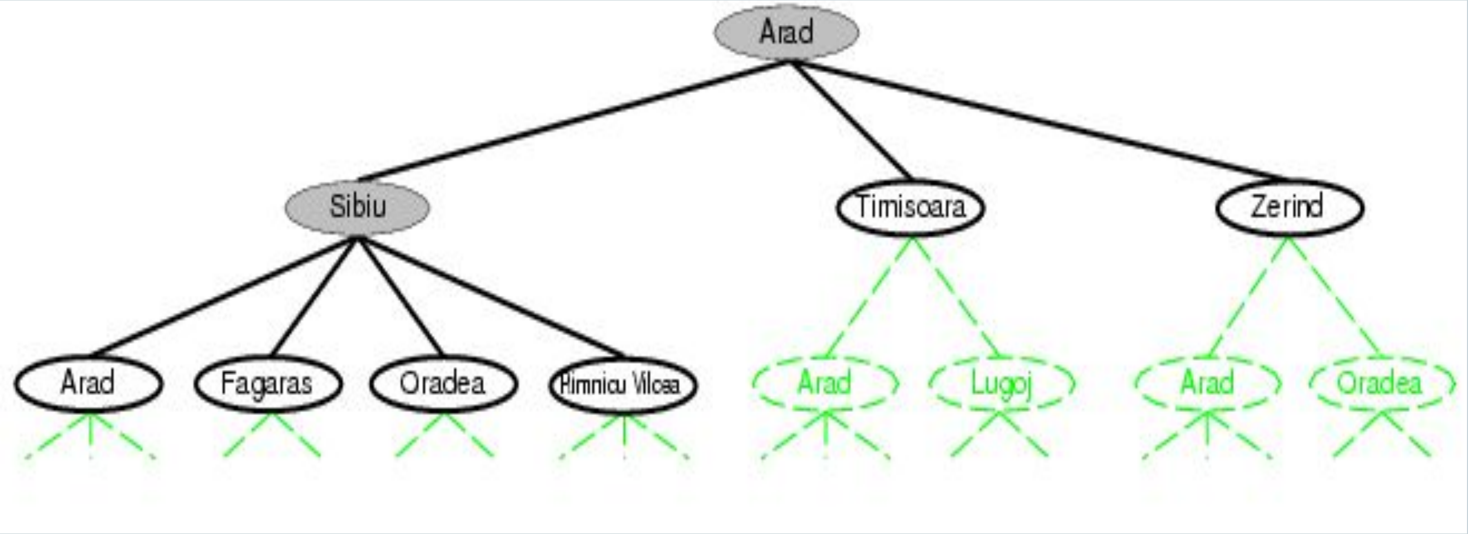
Tree search example



Tree search example



Tree search example



Search Graph vs. State Graph

- Be careful to distinguish
  - Search tree: nodes are **sequences of actions.**
  - State Graph: Nodes are states of the environment.
  - We will also consider soon **search graphs**.
- Demo: http://aispace.org/search/

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?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - 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 ∞)

Search Strategies

- Uninformed (blind) search
- Informed Search
- Adversarial Search (Game Theory)

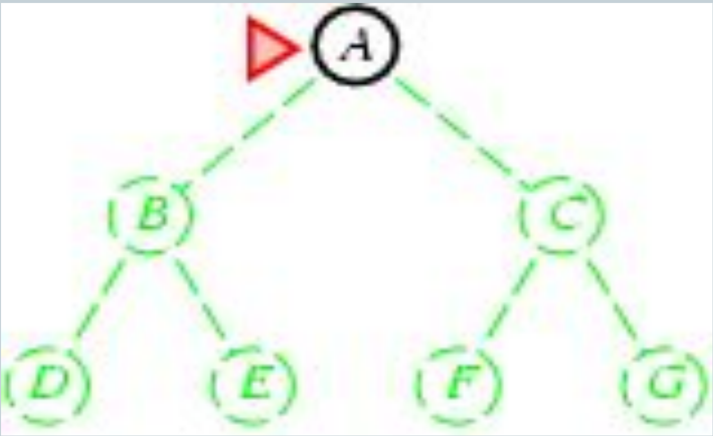
Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Uninformed search (blind search)
  - Breadth-first search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

Breadth-first search

- Expand shallowest unexpanded node

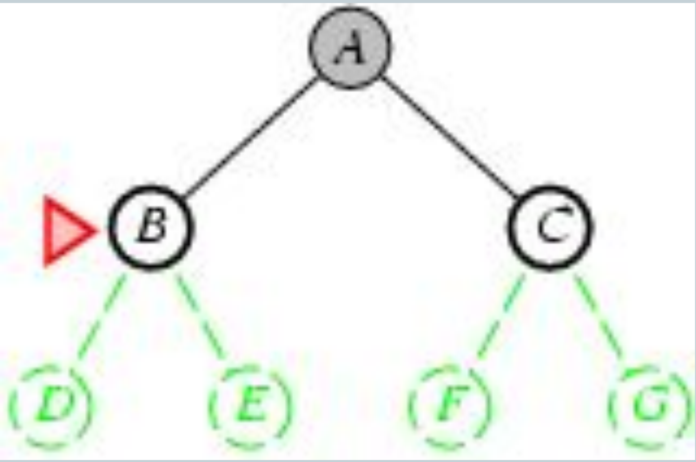
- Implementation:
  - is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node

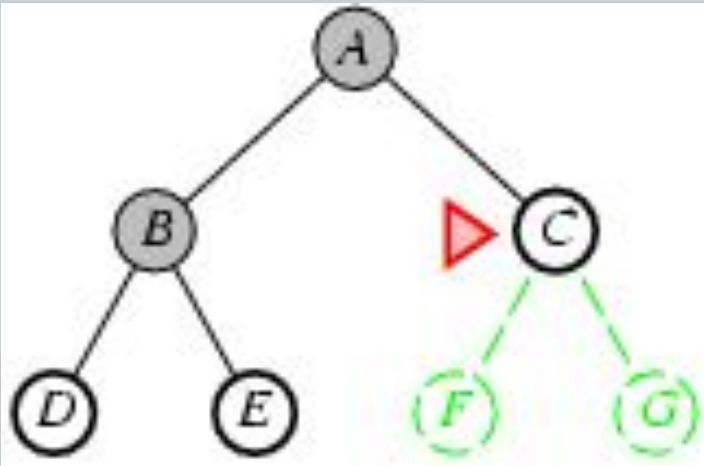
- Implementation:
  - is a FIFO queue, i.e., new successors go at end



Breadth-first search

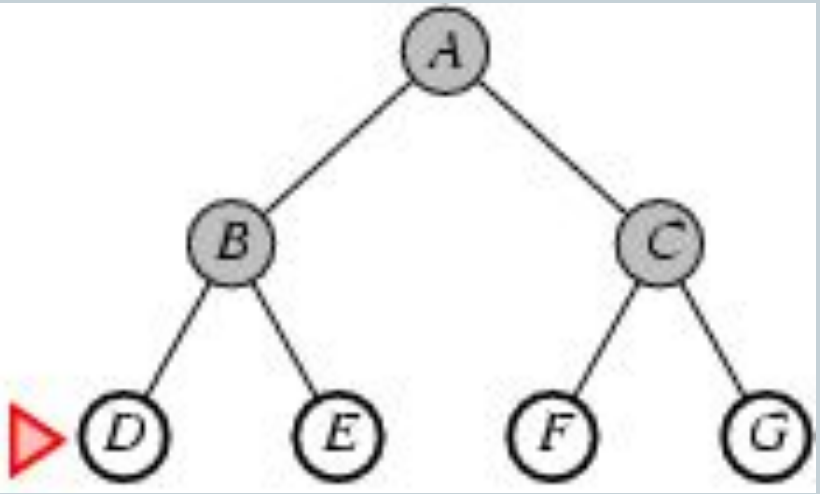
- Expand shallowest unexpanded node

- Implementation:
  - is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node
  http://aispace.org/search/
- Implementation:
  - is a FIFO queue, i.e., new successors go at end

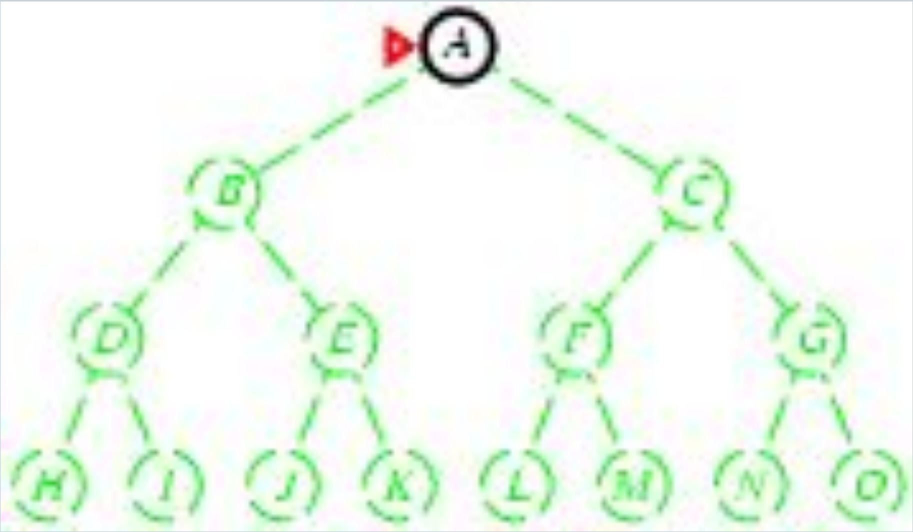


Properties of breadth-first search

- Complete? Time? Space? Optimal?
- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)
- Space is the bigger problem (more than time)

Depth-first search

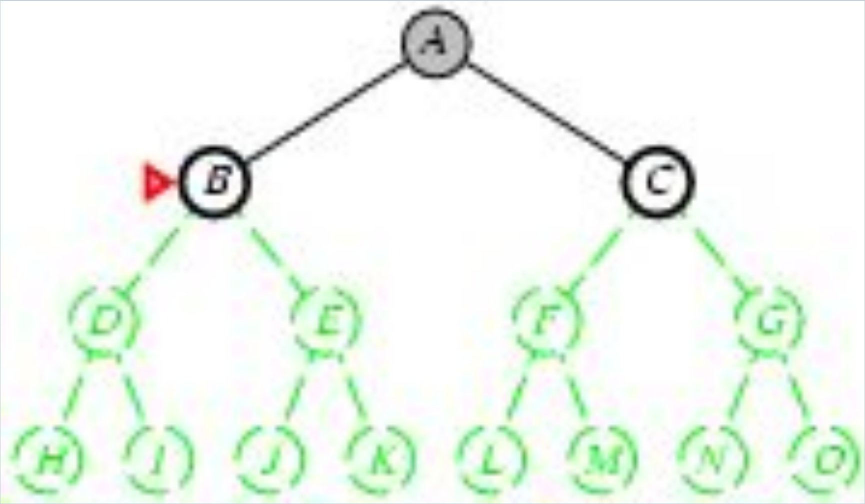
- Expand deepest unexpanded node
- Implementation:
  - LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node

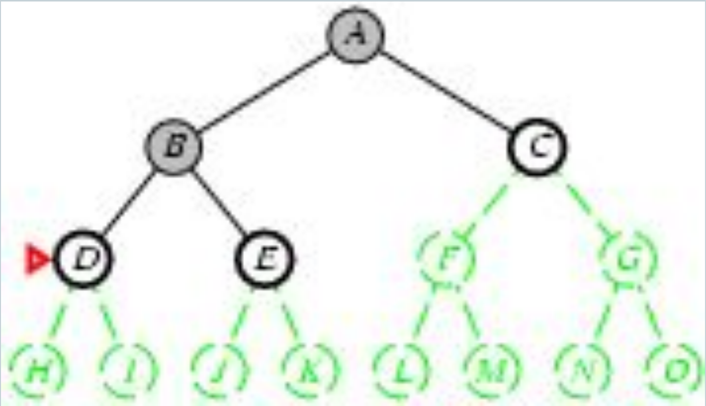
- Implementation:
  - LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node

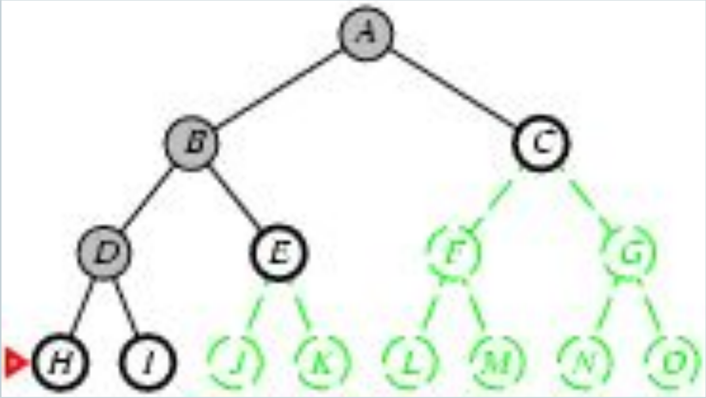
- Implementation:
  - LIFO queue, i.e., put successors at front



Depth-first search

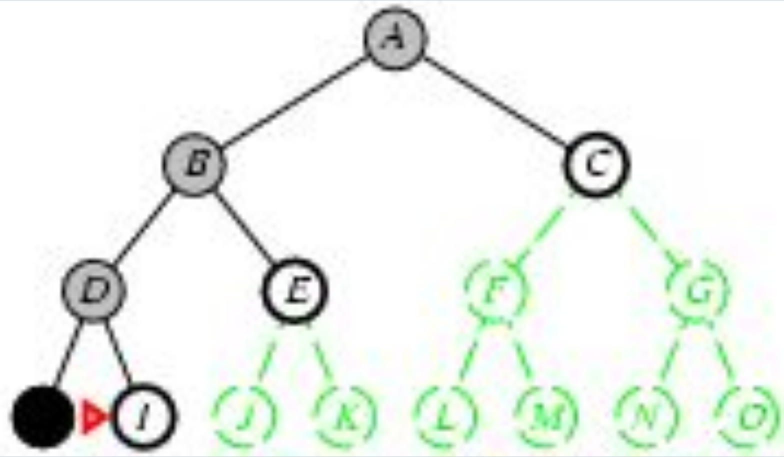
- Expand deepest unexpanded node

- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



Depth-first search

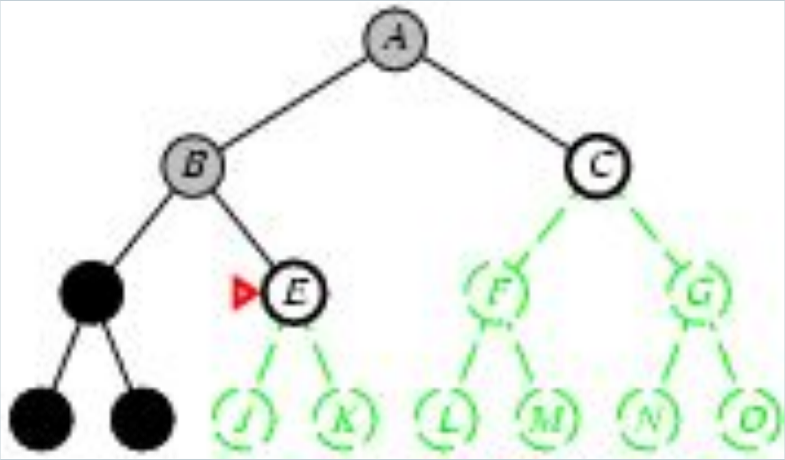
- Expand deepest unexpanded node
- Implementation:
  - LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node

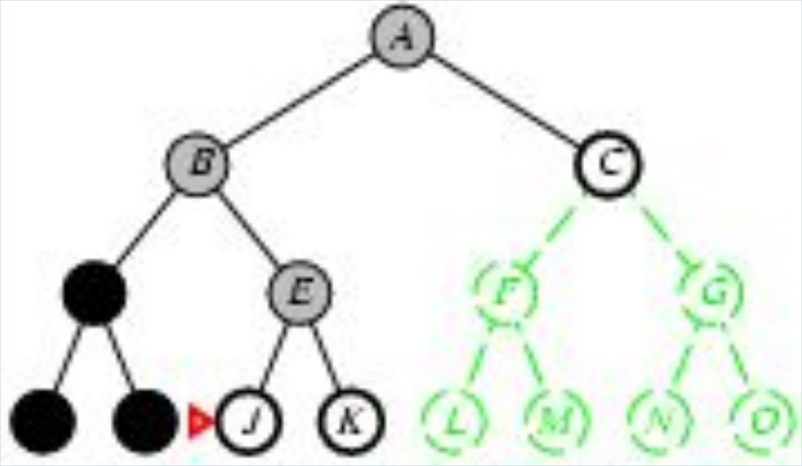
- Implementation:
  - LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node

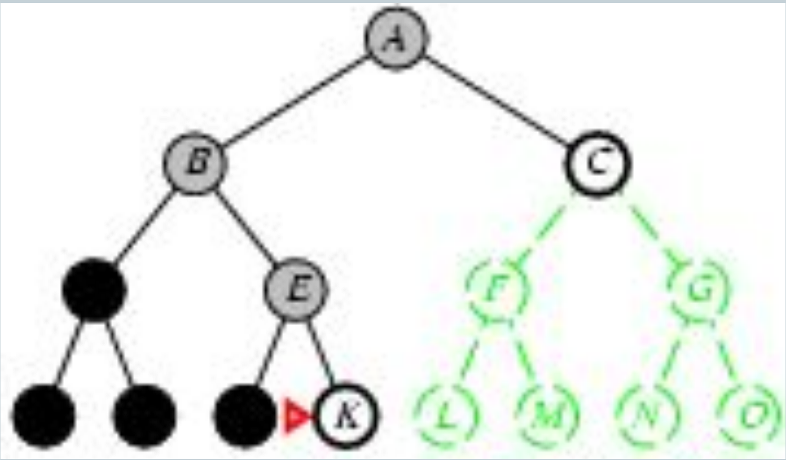
- Implementation:
  - LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node

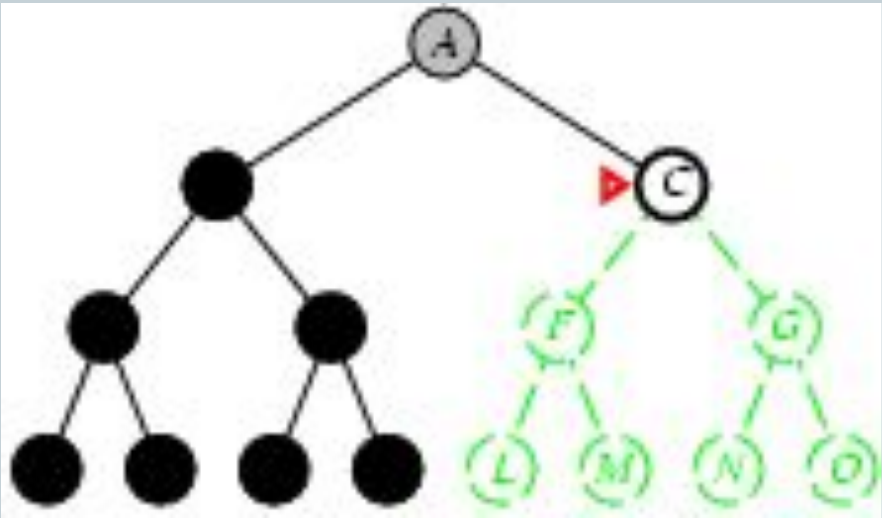
- Implementation:
  - LIFO queue, i.e., put successors at front



Depth-first search

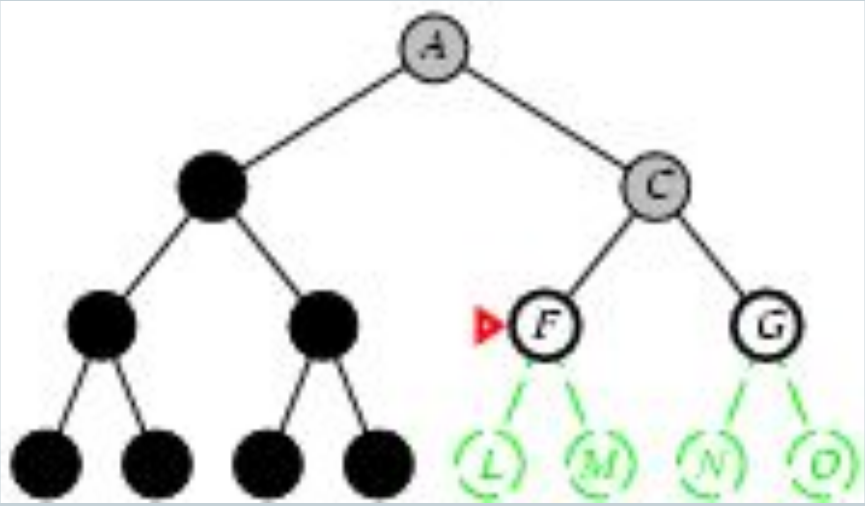
- Expand deepest unexpanded node

- Implementation:
  - LIFO queue, i.e., put successors at front



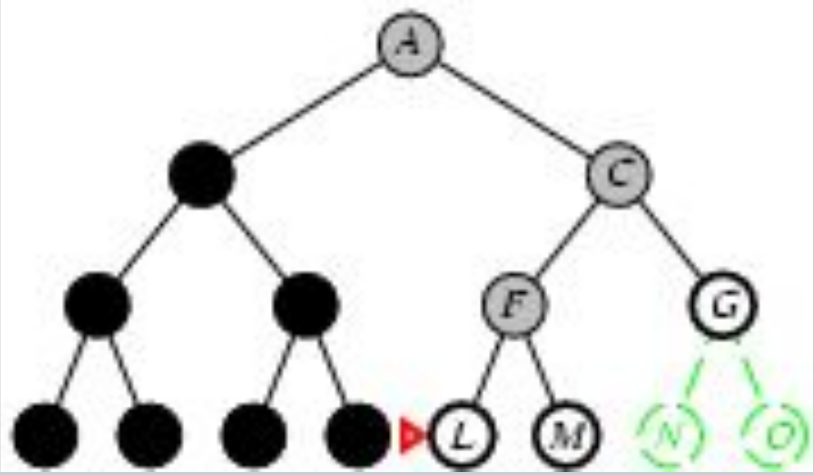
Depth-first search

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



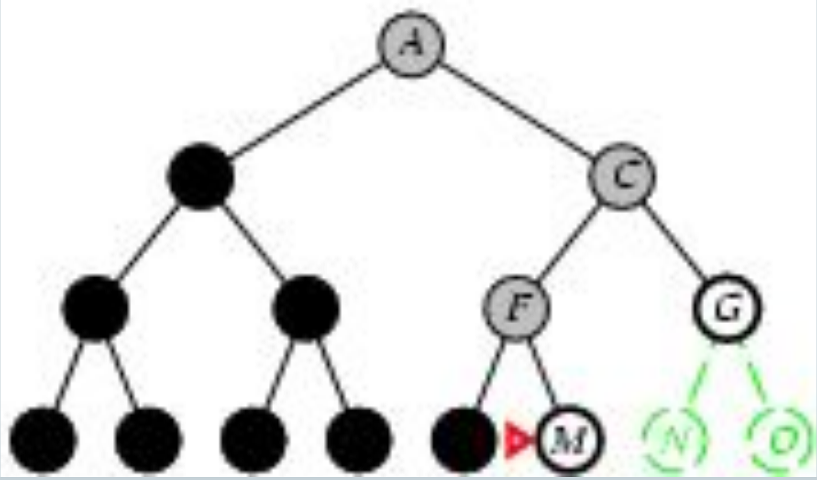
Depth-first search

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



Depth-first search

- Expand deepest unexpanded node
  http://aispace.org/search/
- Implementation:
  - LIFO queue, i.e., put successors at front



Properties of depth-first search

- Complete? Time? Space? Optimal?
- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path (graph search)
    - ☐ complete in finite spaces
- Time? $O(b^m)$: terrible if maximum depth m is much larger than solution depth d
  - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space! Store single path with unexpanded siblings.
  - Seems to be common in animals and humans.
- Optimal? No.

  Important for exploration (on-line search)

Depth-limited search

- depth-first search with depth limit *l*,
  - i.e., nodes at depth *l* have no successors
  - Solves infinite loop problem
- Common AI strategy: let user choose search/resource bound.

  Complete? No if l < d:

- Time? $O(b^l)$:
- Space? $O(bl)$, i.e., linear space!
- Optimal? No if l > b

Iterative deepening search

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure

inputs: *problem*, a problem

for *depth* ← 0 **to** ∞ **do**

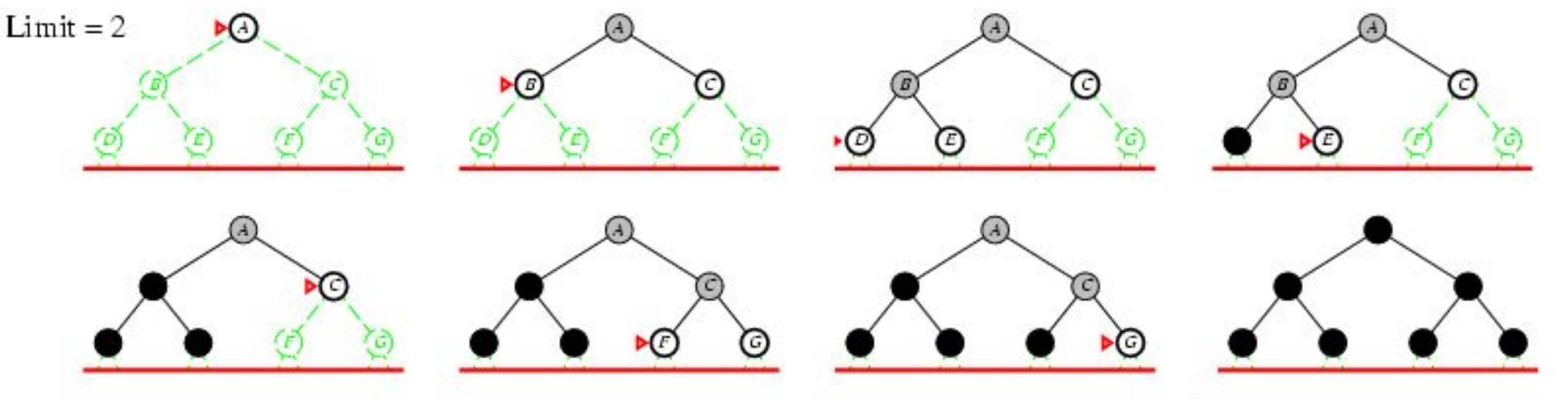
result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)

if *result* ≠ cutoff **then return** *result*

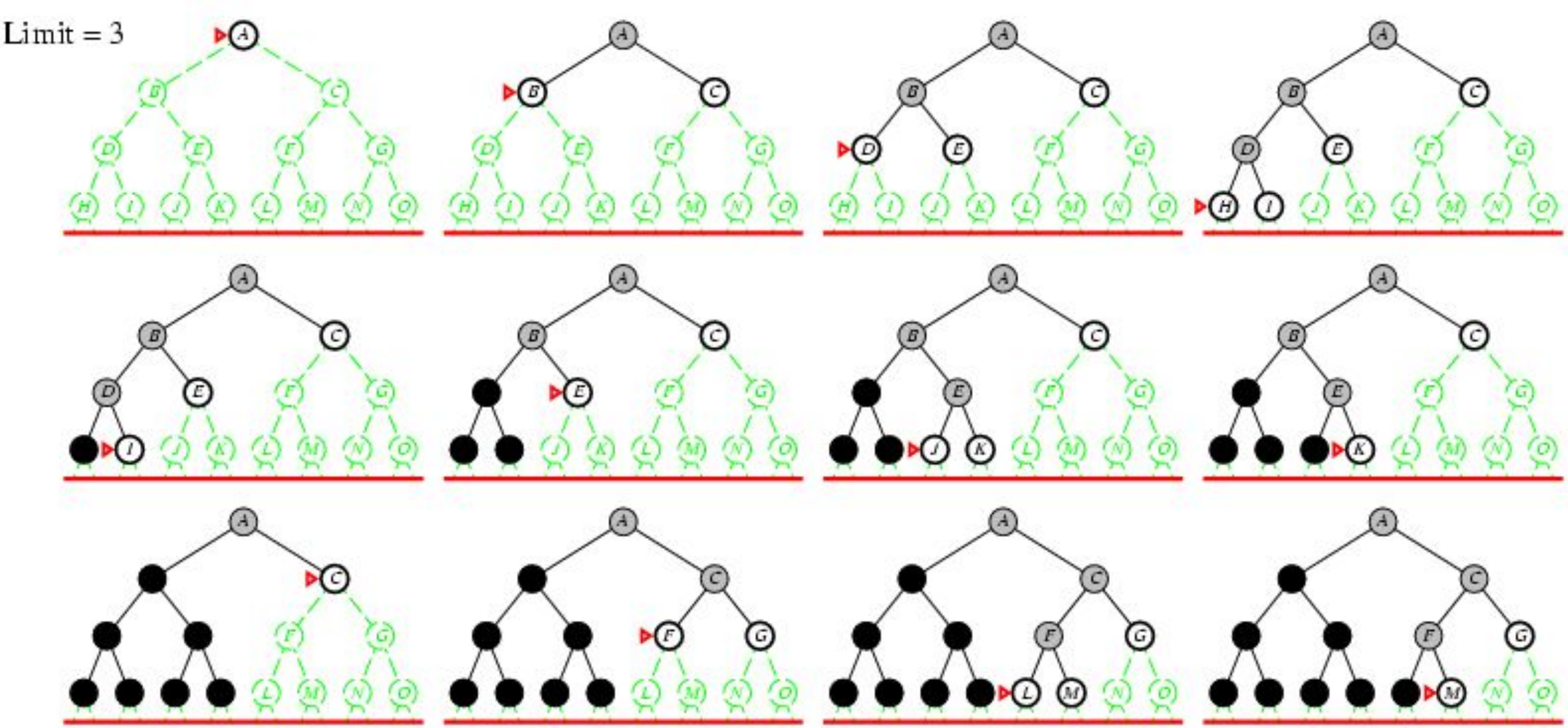
Iterative deepening search l =0

Iterative deepening search l =1

Iterative deepening search l =2



Iterative deepening search l =3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

$$N_{IDS} = (d+1)b^0 + d\ b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- For $b = 10$, $d = 5$,

$N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$

$N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$

- Overhead = (123,456 - 111,111)/111,111 = 11%

Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

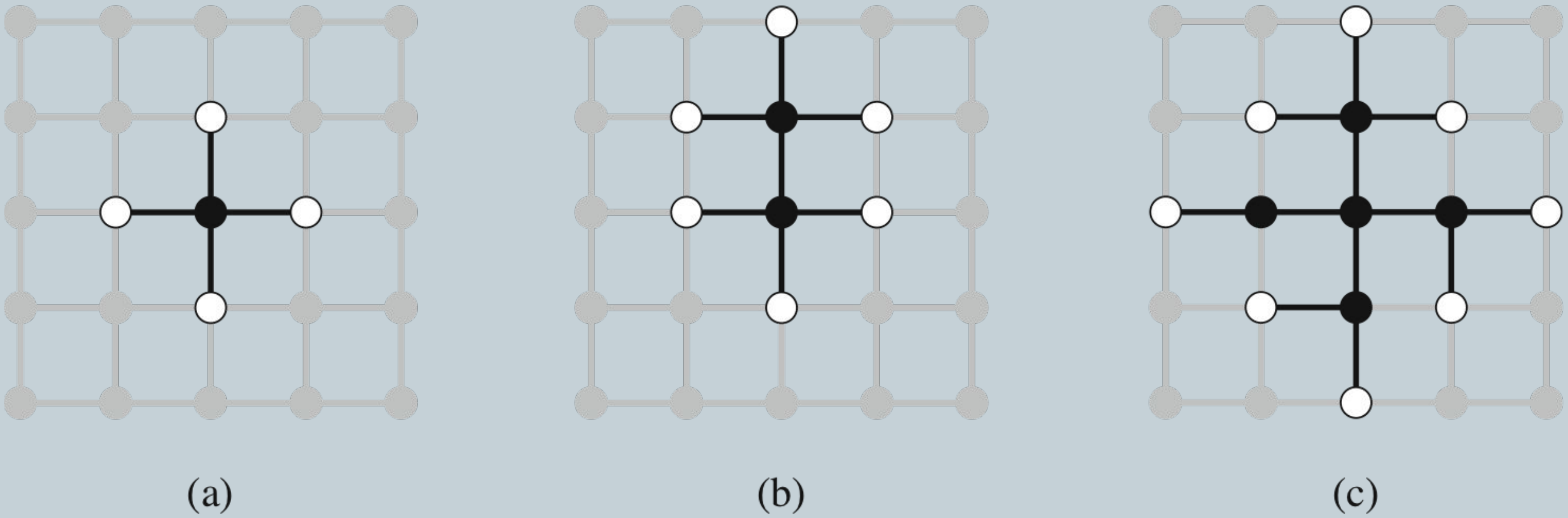
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Graph search


```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

- Simple solution: just keep track of which states you have visited.
- Usually easy to implement in modern computers.

The Separation Property of Graph Search



- Black: expanded nodes.
- White: frontier nodes.
- Grey: unexplored nodes.

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

End of Chapter 3

Informed search algorithms

CHAPTER 4

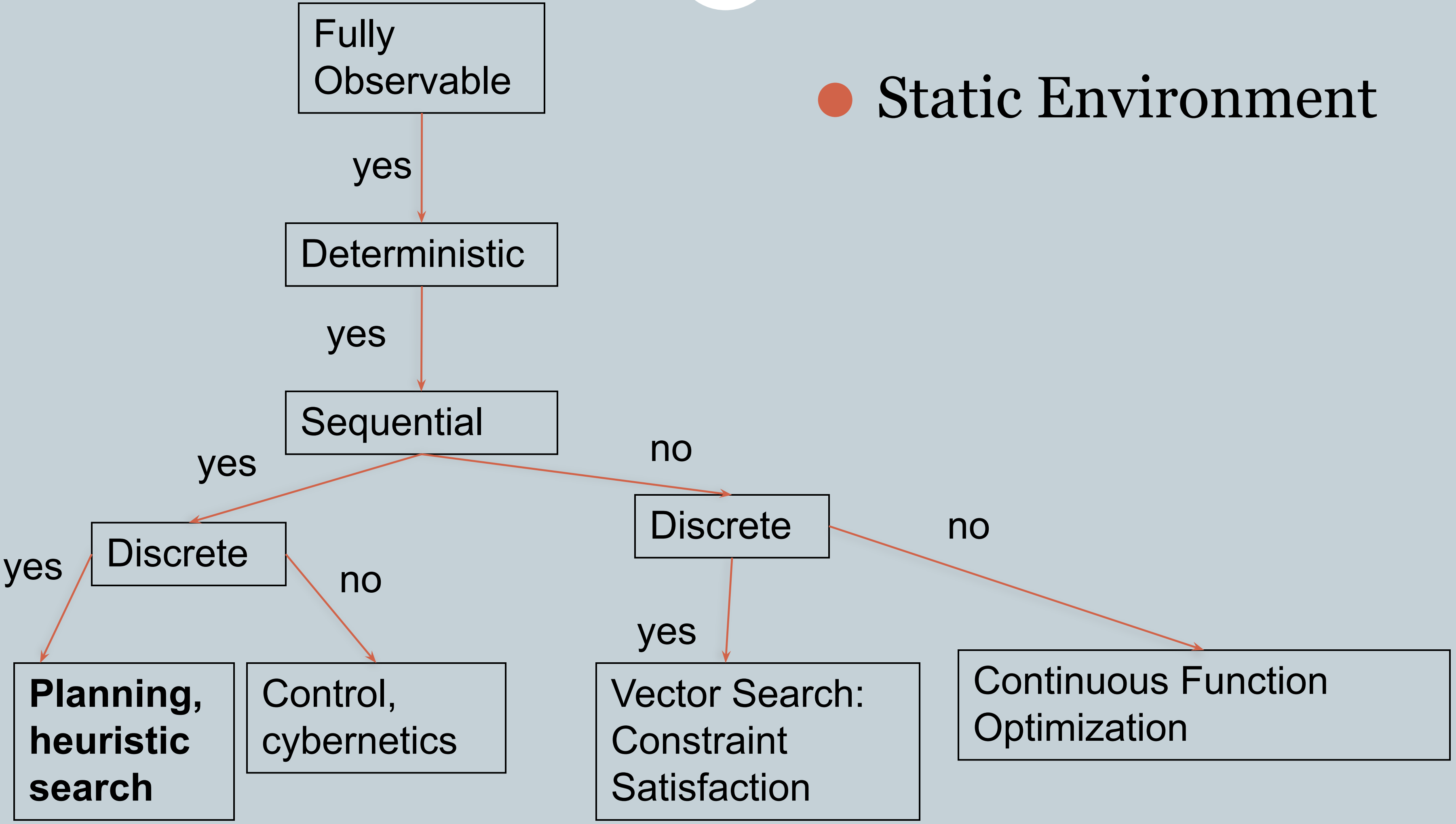
Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Global Edition 3/E

Outline

- Best-first search
- A^* search
- Heuristics

Environment Type Discussed In this Lecture

- Static Environment



Review: Tree search


```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem] applied to STATE(node) succeeds return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

- A search strategy is defined by picking the order of node expansion
- Which nodes to check first?

Knowledge and Heuristics

- Simon and Newell, *Human Problem Solving*, 1972.
- S&N: intelligence comes from **heuristics** that help find promising states fast.

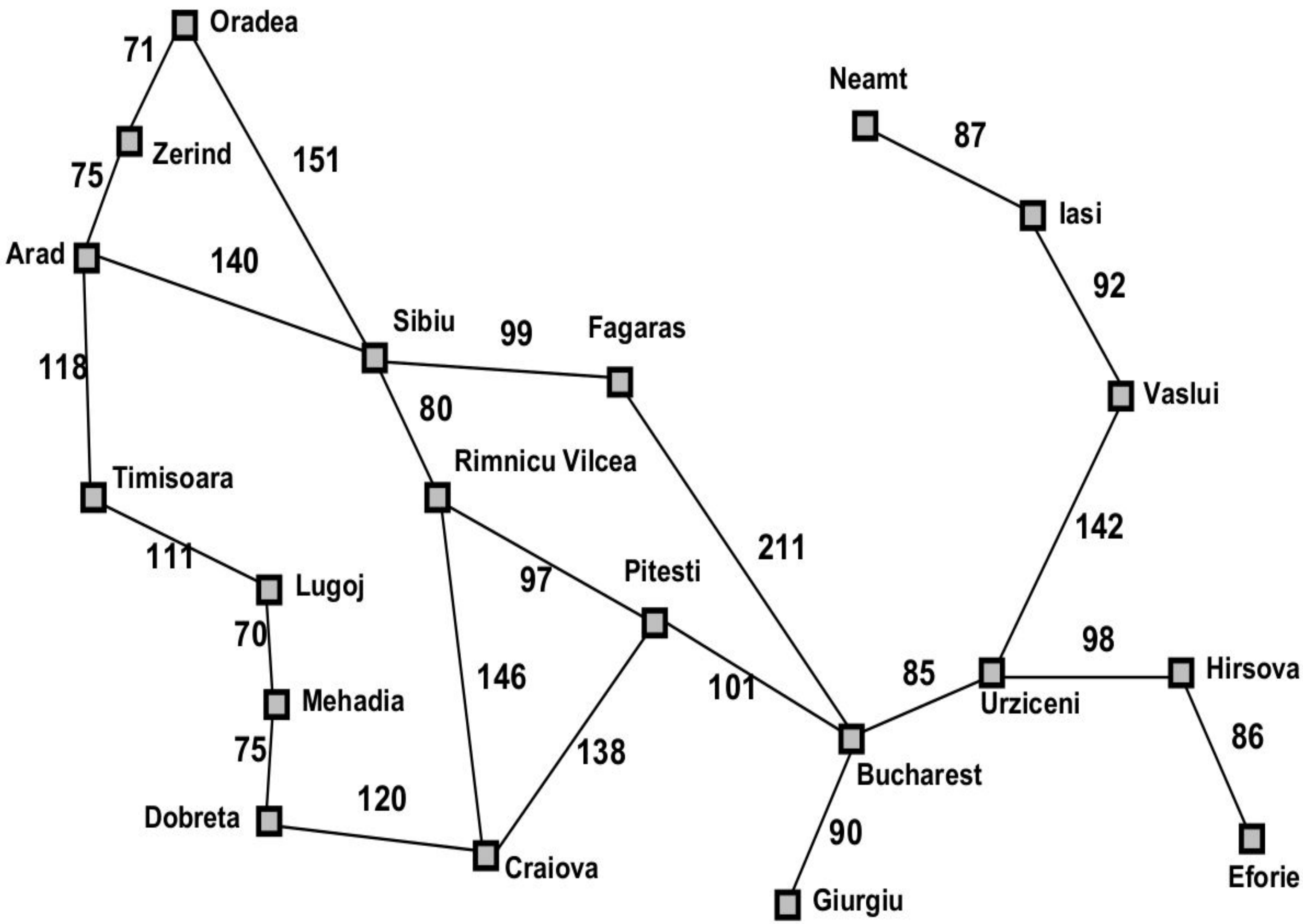
Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node

- Implementation:
  Order the nodes in frontier in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A^* search

Romania with step costs in km



Straight−line distance
to Bucharest

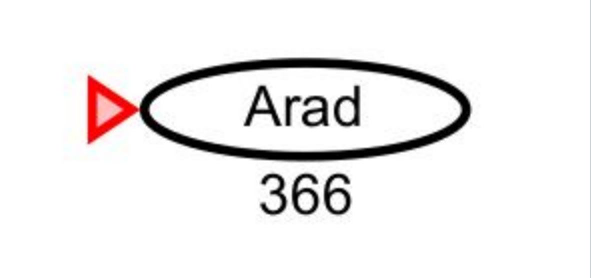
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

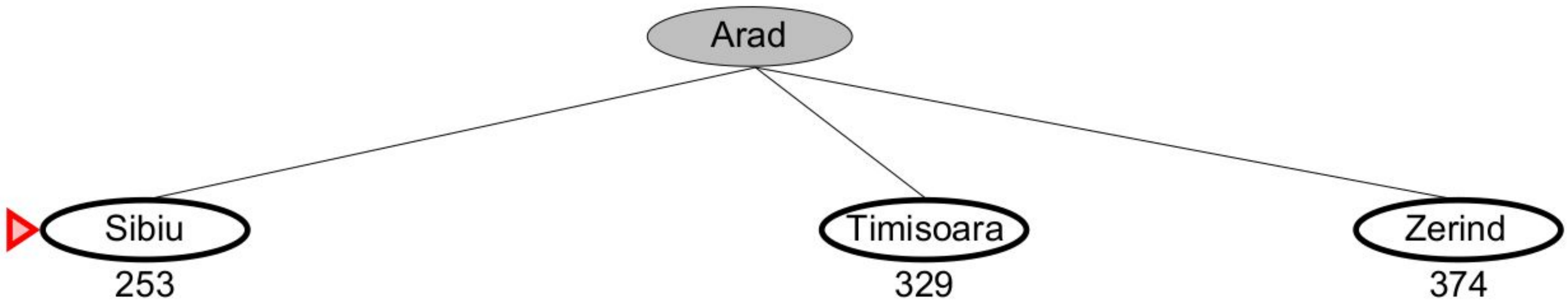
- Evaluation function
  - $f(n) = h(n)$ (heuristic)
  - = estimate of cost from n to *goal*

- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

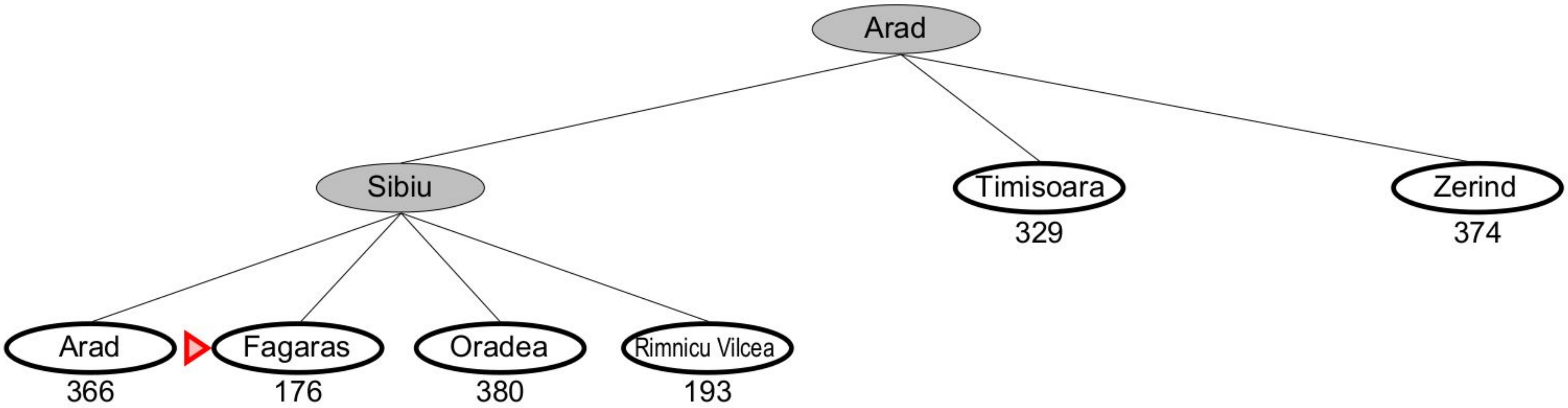
Greedy best-first search example



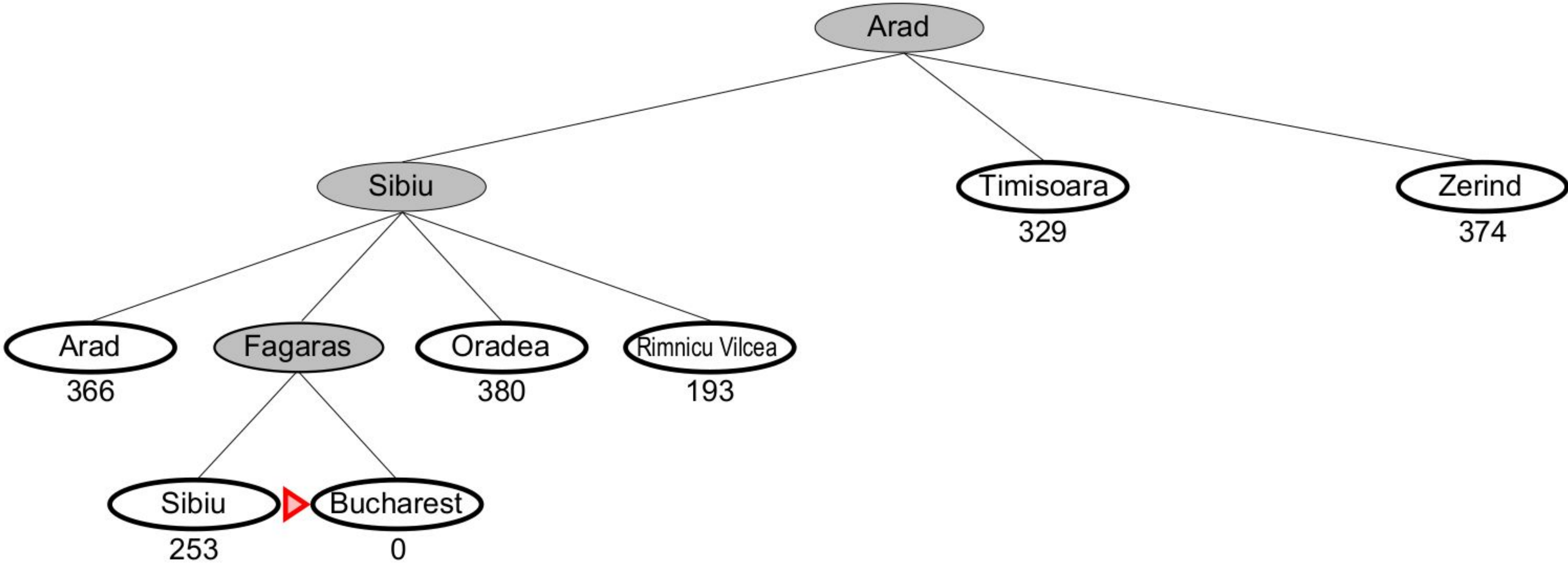
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



http://aispace.org/search/

Properties of greedy best-first search

- Complete? No – can get stuck in loops,
  - e.g. as Oradea as goal

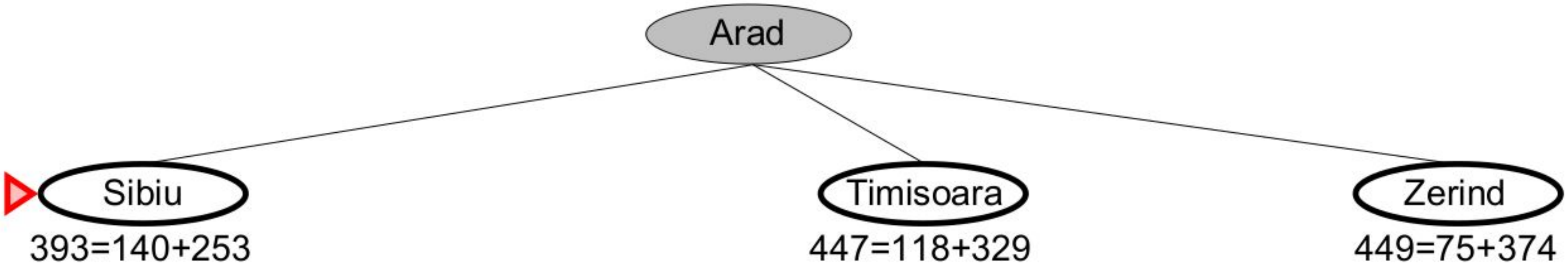
    Iasi -> Neamt -> Iasi -> Neamt …..
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

A^* search

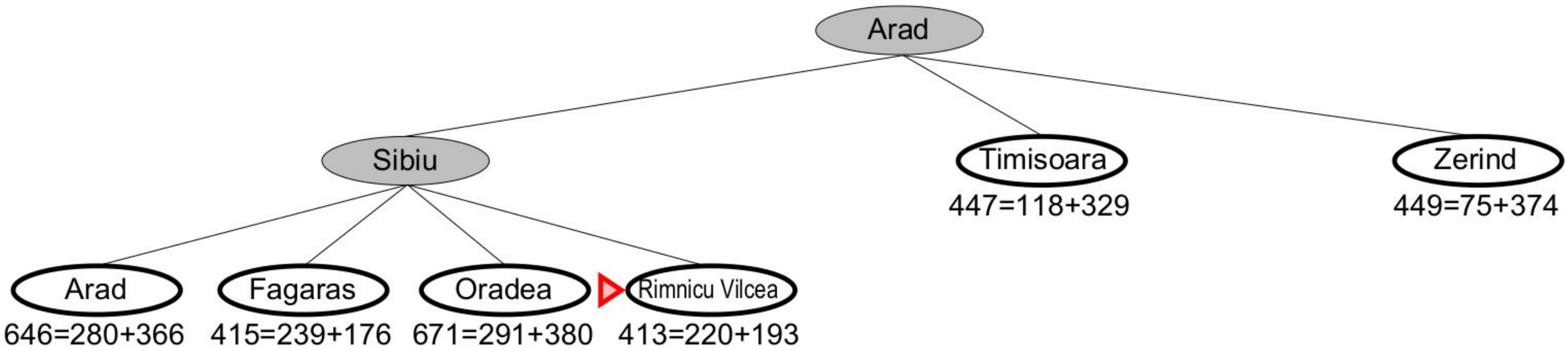
- Idea: avoid expanding paths that are already expensive.
- Very important!
- Evaluation function $f(n) = g(n) + h(n)$
  $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- **$f(n)$ = estimated total cost of path through n to goal**

A^* search example

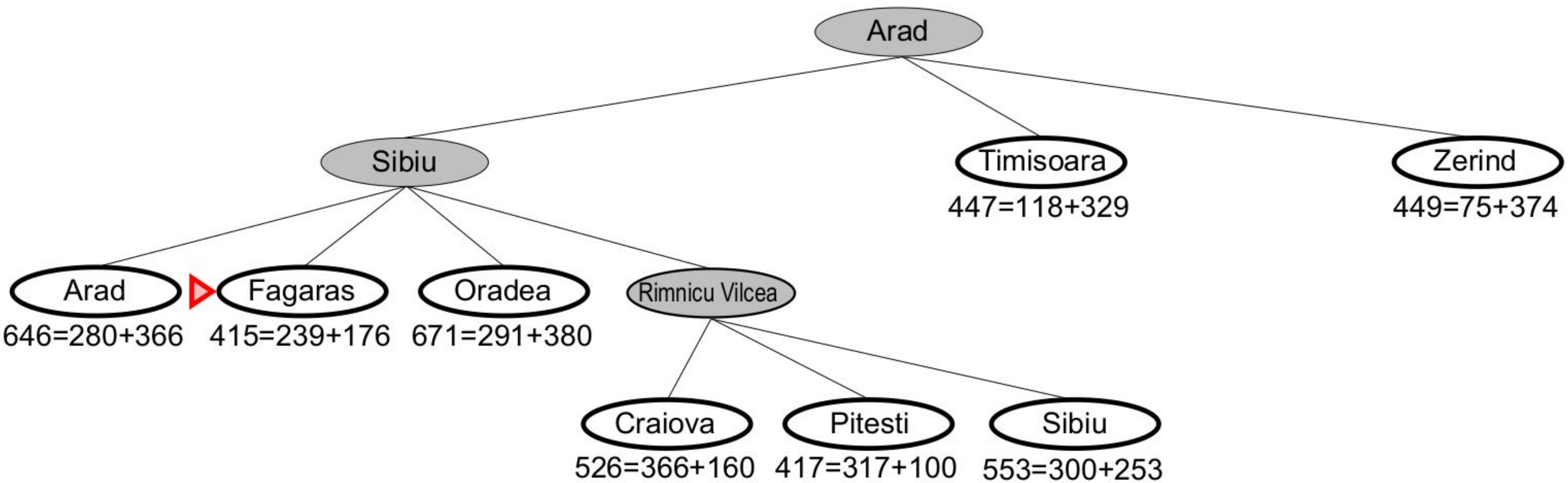
A* search example



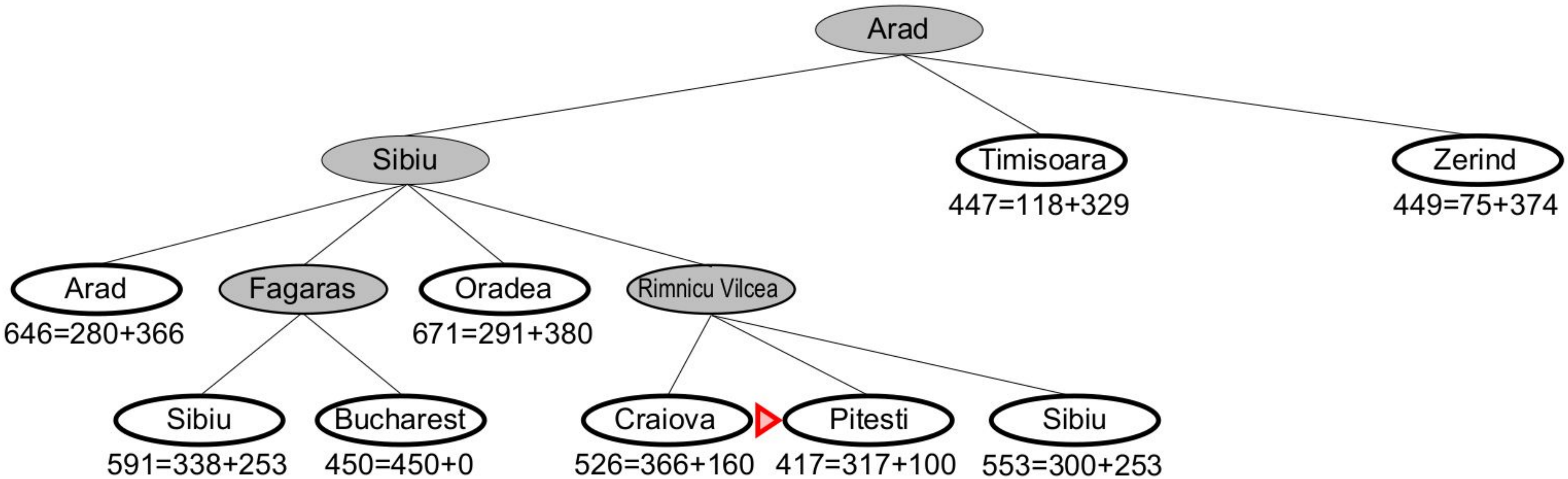
A^* search example



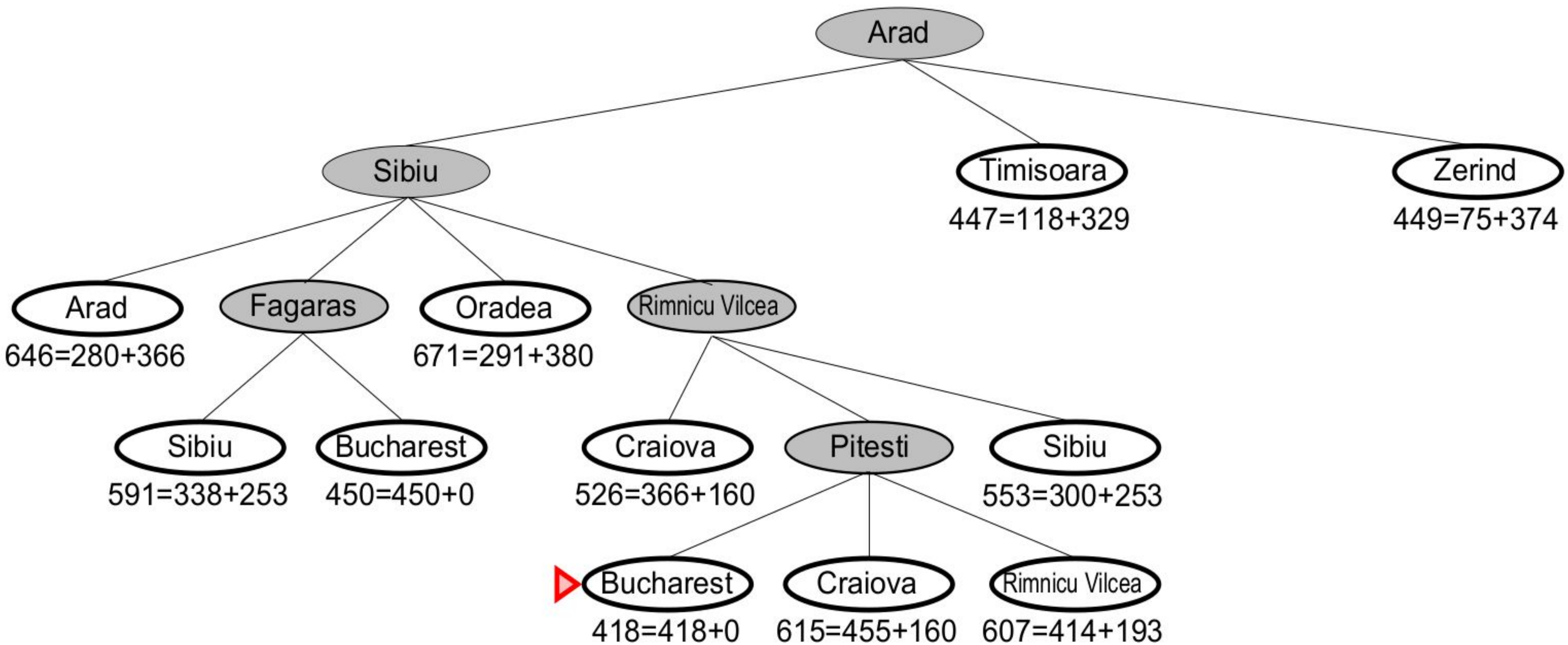
A* search example



A* search example



A* search example



http://aispace.org/search/

- We stop when the node with the lowest f-value is a goal state.
- Is this guaranteed to find the shortest path?

Properties of A*

- Complete? Yes (unless there are infinitely many nodes with f ≤ *f(G)*)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

Summary

- 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)