BLG435E
Artificial Intelligence

Lecture 3: Problem-Solving

Outline

- Problem-solving agents
- Problem formulation
- Uninformed search strategies
- Informed search strategies
  - Best-first search
  - A* search
  - Heuristics

Problem Solving

- Goal formulation
  - limiting objectives
  - Uses current situation and agent's performance measure
  - A set of world states
- Problem formulation
  - What actions and states to consider
- Search procedure
  - Problem → solution in the form of action sequence

Problem-solving Agents

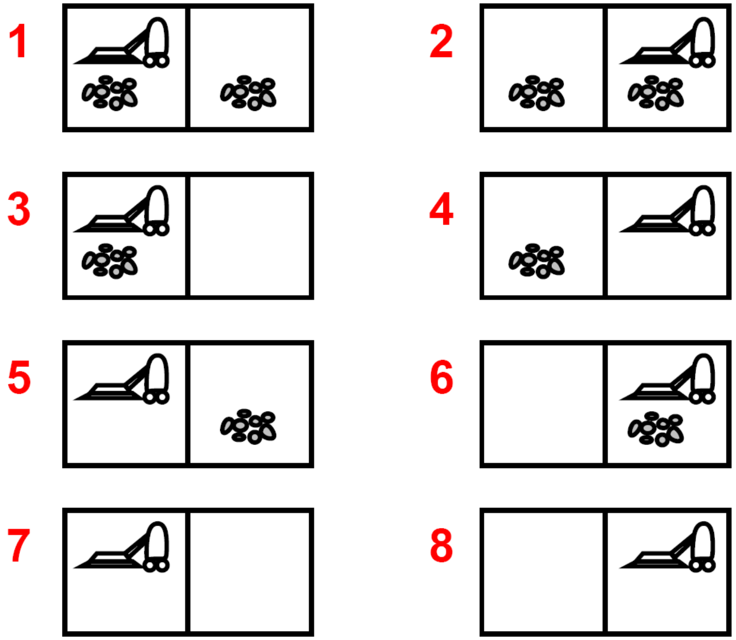
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

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

Well-defined Problems

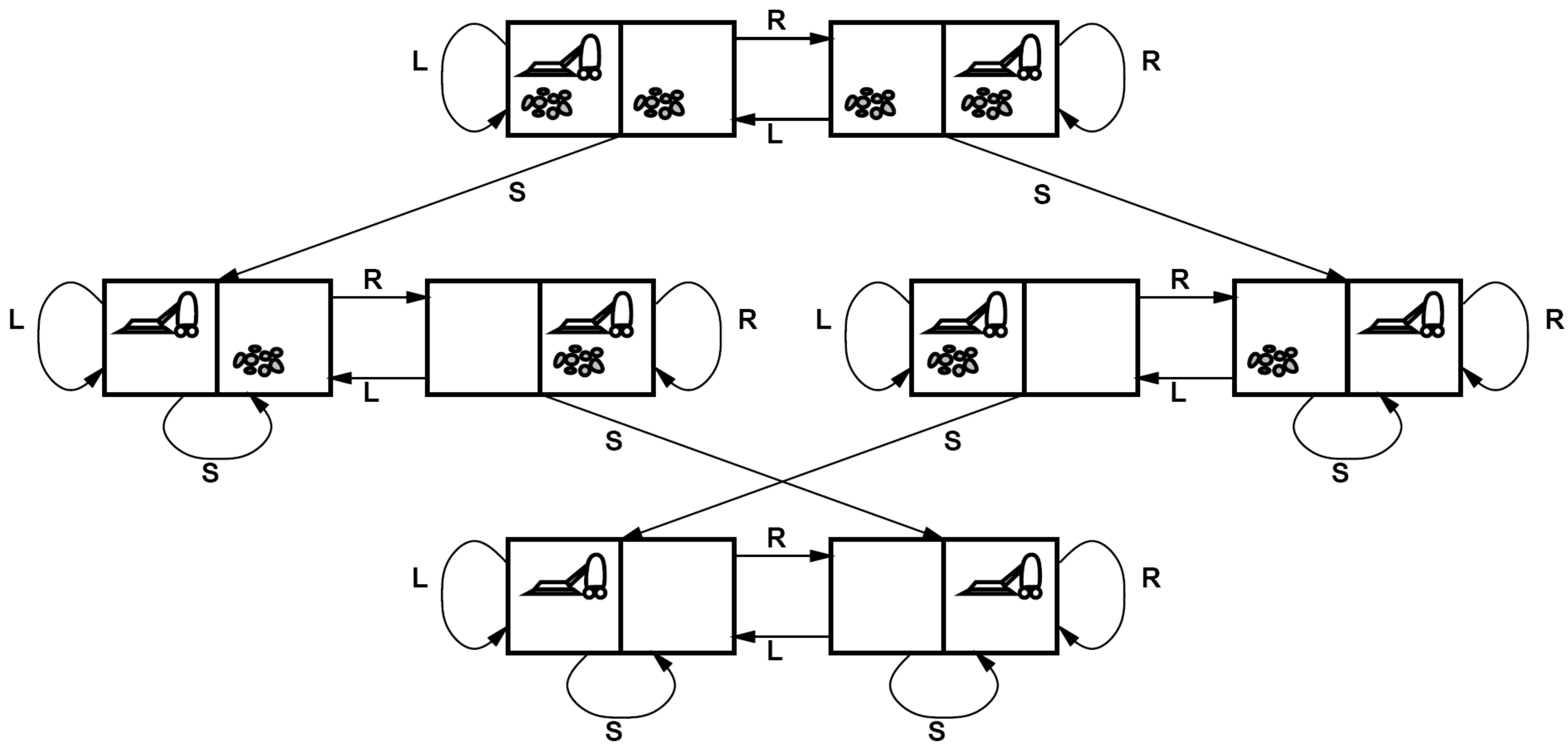
- Initial state
- Actions
- Transition model
- State space
  - Path in the state space
  - Size of the search space
- Goal test
  - Explicit vs. implicit
- Path cost
  - Step cost
- The solution to the problem
  - Optimal solution
- Abstraction

Vacuum-world Problem

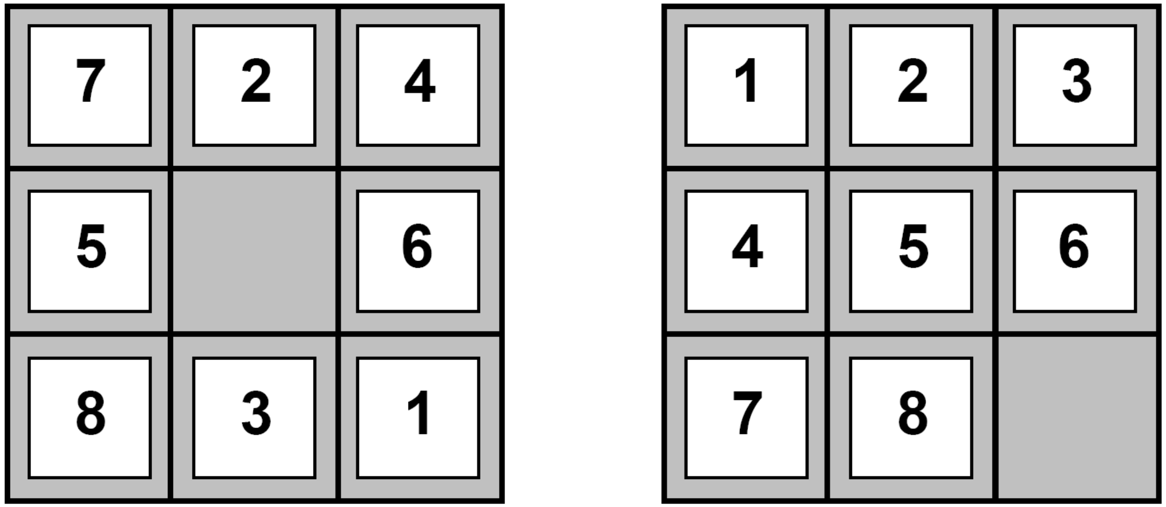
- Single state, start in #5
- Solution?



Vacuum-world State Space Graph



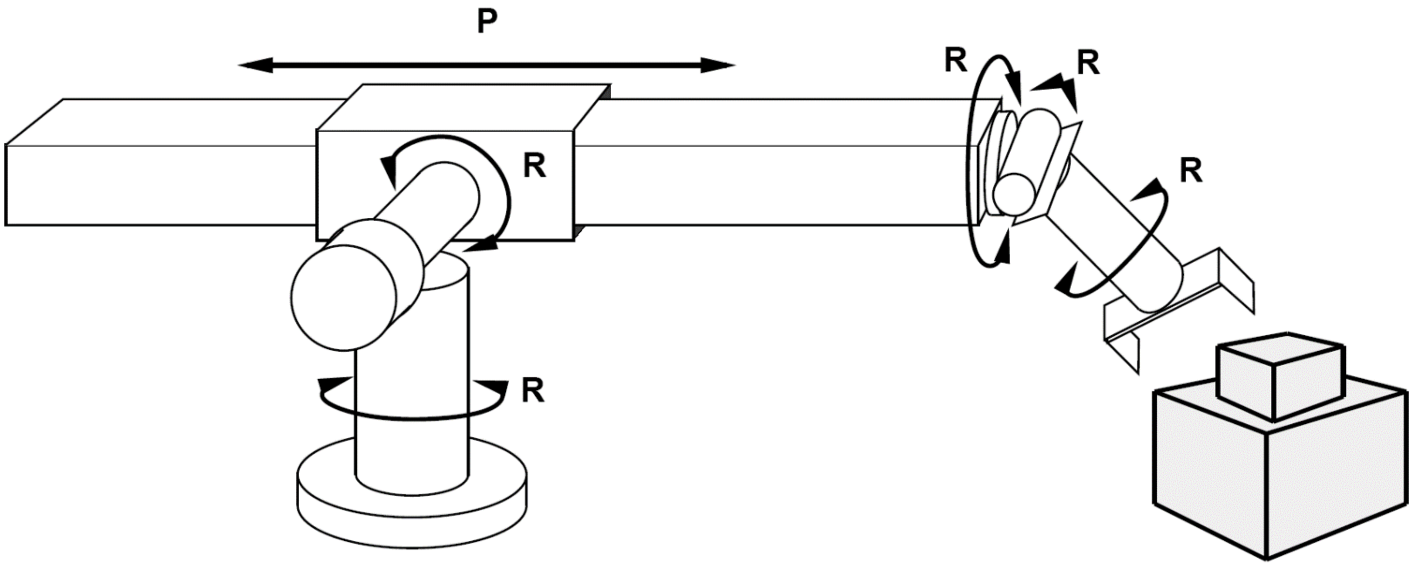
8-Puzzle



Start State

Goal State

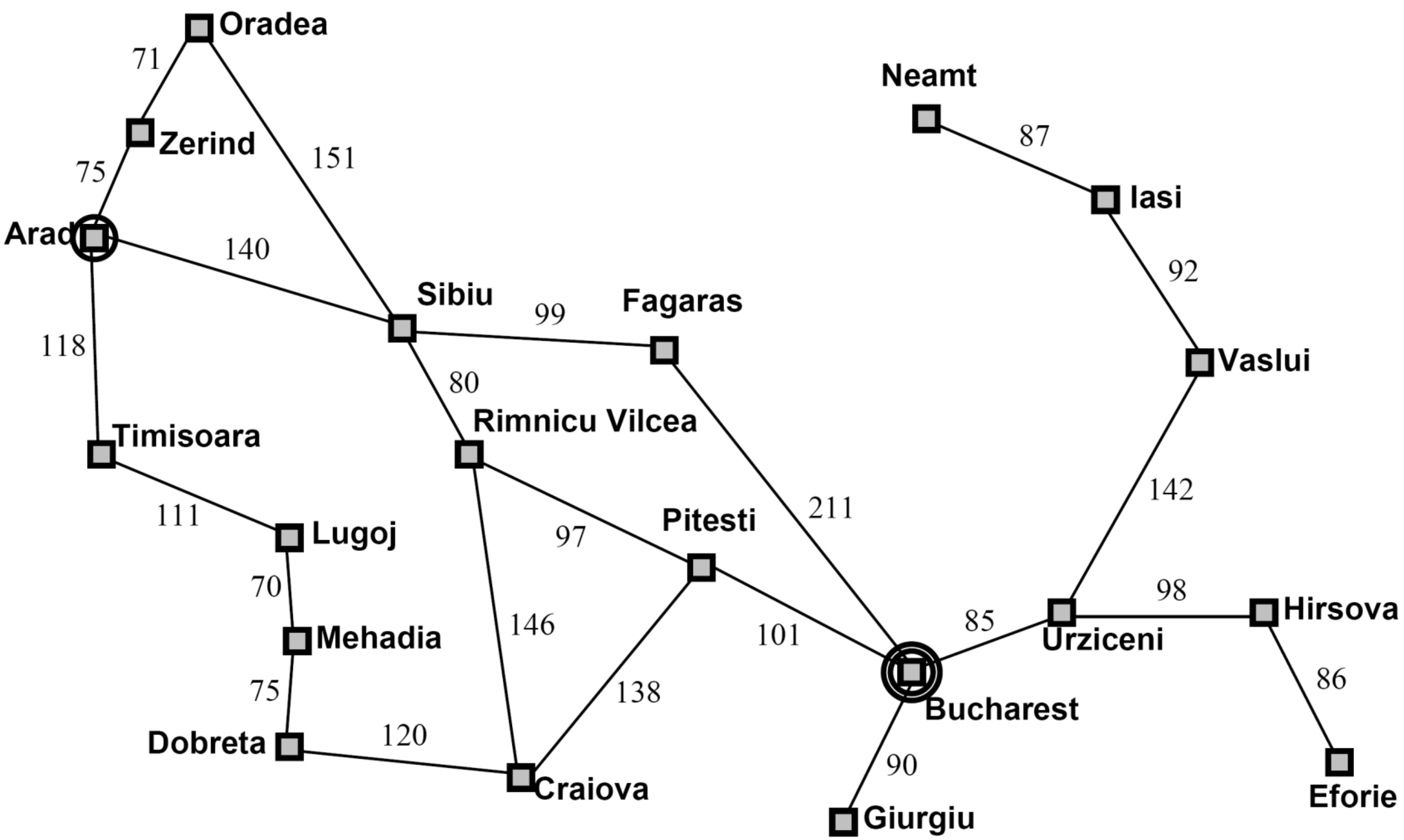
Robotic Assembly



Romania Problem

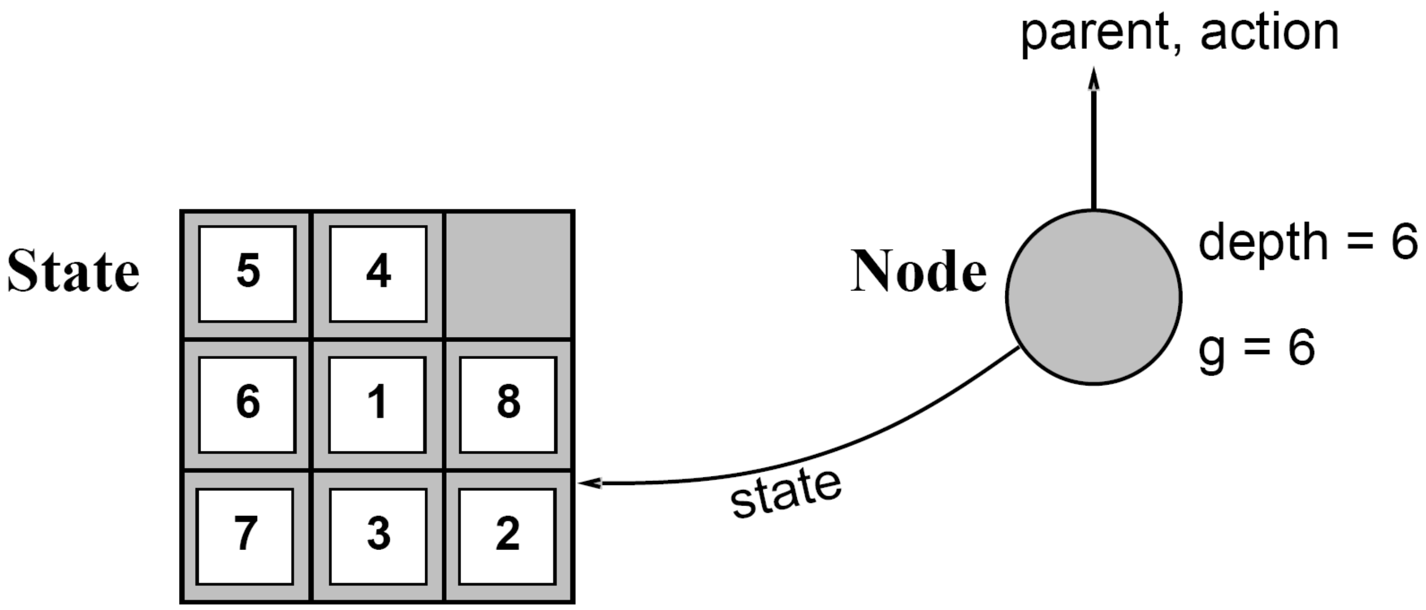
- On holiday in Romania, currently in Arad. Flight leaves tomorrow from Bucharest
- Formulate goal:
  - Be in Bucharest
- Formulate problem:
  - States: various cities
  - Actions: drive between cities
- Find solution: sequence of actions

Romania



Oradea
71
Zerind
151
75
Arad
140
118
Sibiu
99
Fagaras
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
Urziceni
98
Hirsova
85
Bucharest
86
90
Giurgiu
Eforie

Implementation: states vs. nodes



Node data structure

- **State**: the state to which the node corresponds
- **Parent-node**: the node in the search tree that generated this node
- **Action**: action that was applied
- **Path-cost**: the cost from the initial state to the node - $g(n)$
- **Depth**: the number of steps along the path from the initial state

Node data structure

- Nodes = states ?
- Frontier: the collection of nodes that have been generated but not yet expanded
- Implementation of frontier? Set?

Tree and Graph -Search Algorithms

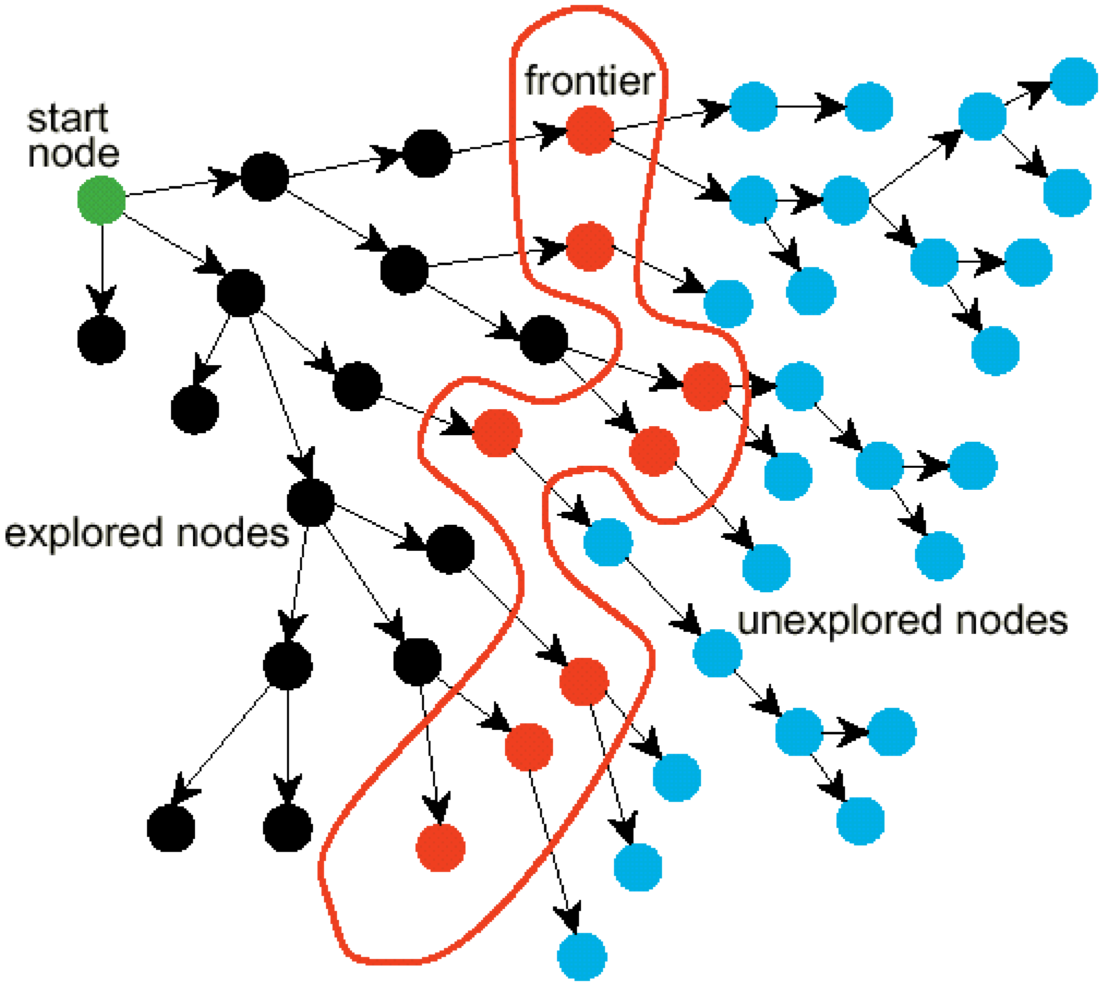
- Off-line, simulated exploration of the state space by expanding states

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     add the node to the explored set
     expand the chosen node, adding the resulting nodes to the frontier
        only if not in the frontier or explored set
```

- Choosing, testing, expanding

Nodes in The Search Tree

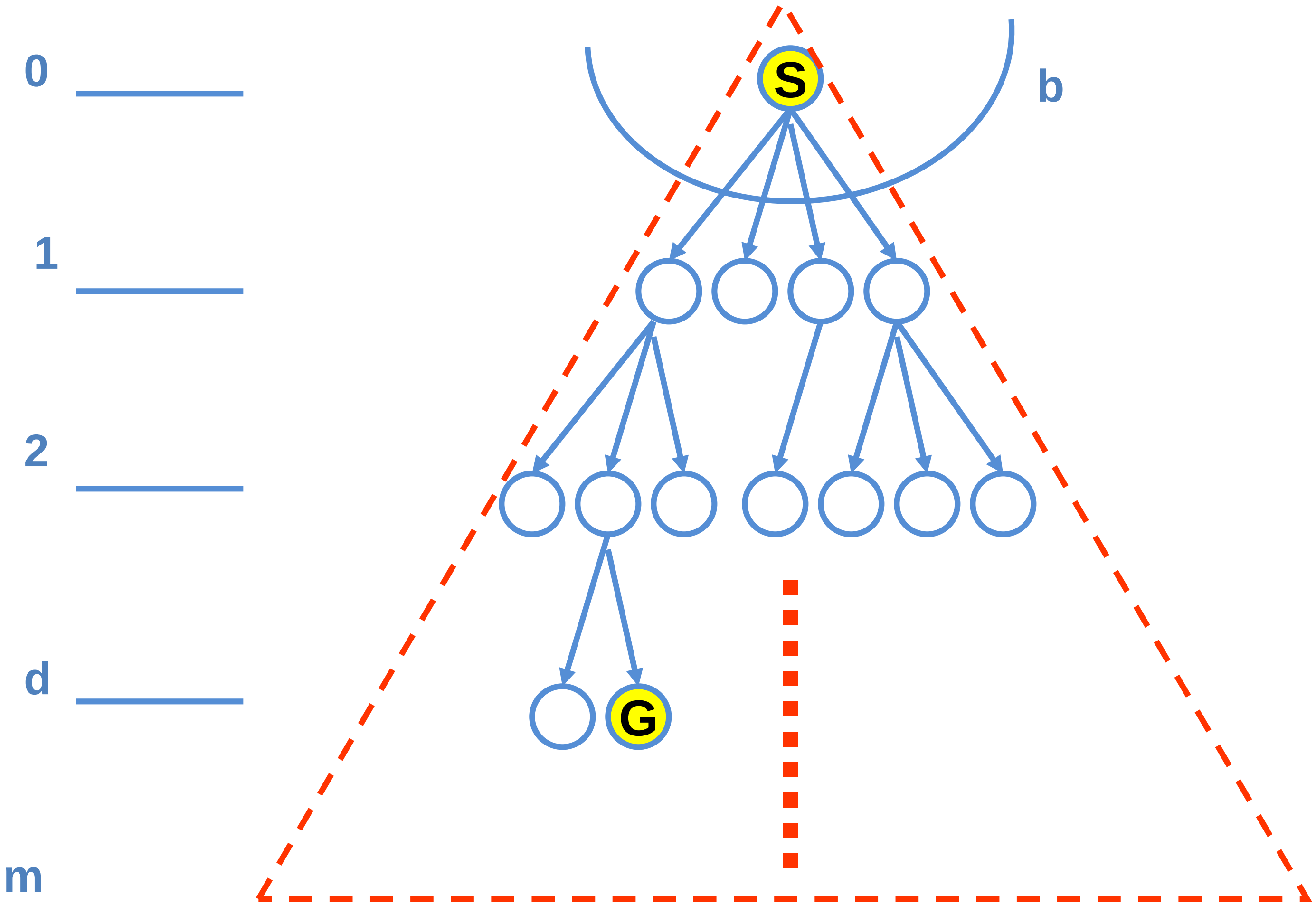


Measuring Problem-Solving Performance

- Strategy = order of node expansion

- Completeness
- Optimality
- Time complexity
- Space complexity
  - b: maximum branching factor
  - d: depth of the least-cost solution
  - m: maximum depth of the search tree

Measuring Problem-Solving Performance



BLG435E
Artificial Intelligence

Uninformed Search Strategies

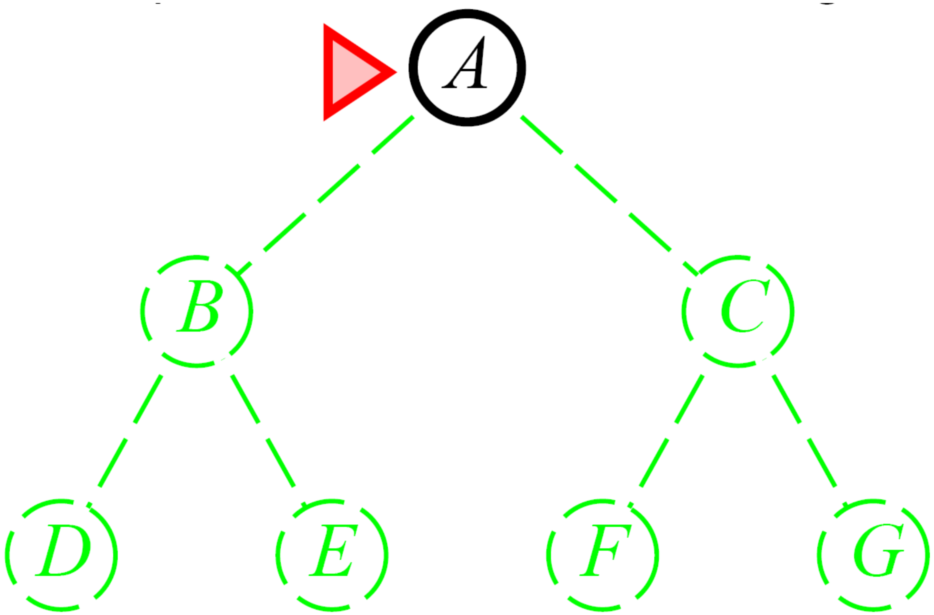
Uninformed Search Strategies

- Use information only available in the problem definition

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative Deepening search

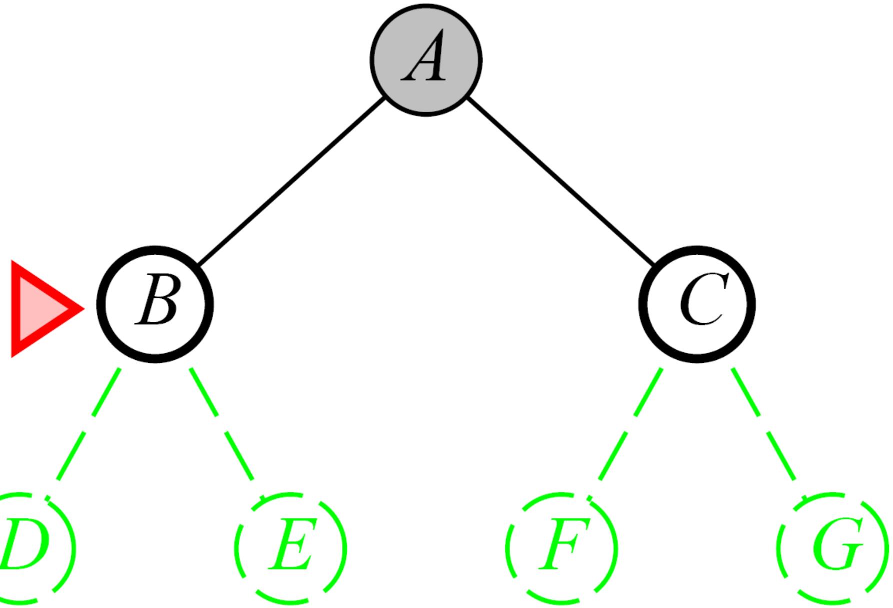
Breadth-first Search (BFS)

- Expand shallowest unexpanded node
- Frontier is a FIFO queue



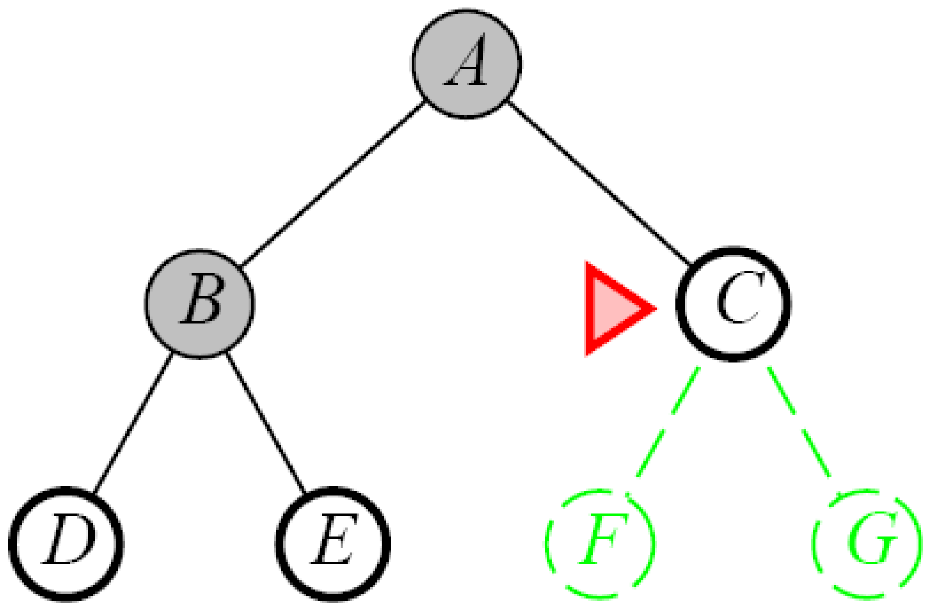
Breadth-first Search (BFS)

- Expand shallowest unexpanded node
- Frontier is a FIFO queue



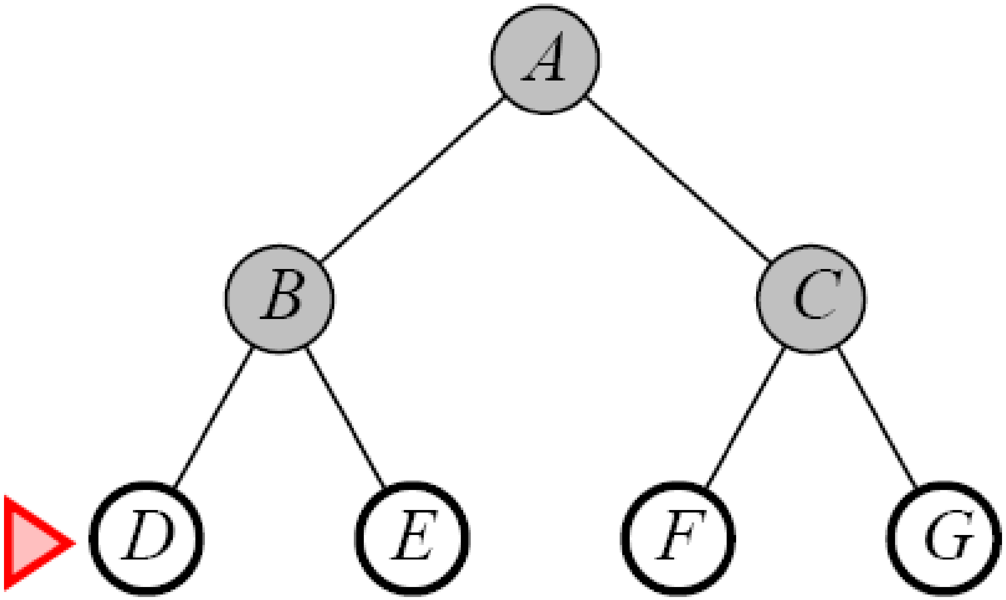
Breadth-first Search (BFS)

- Expand shallowest unexpanded node
- Frontier is a FIFO queue



Breadth-first Search (BFS)

- Expand shallowest unexpanded node
- Frontier is a FIFO queue



Properties of BFS

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

Properties of BFS

- Complete? Yes, if b is limited
- Optimal? Optimal, if path cost is nondecreasing
- Time complexity? $O(b^d)$
- Space complexity? $O(b^d)$ keeps every node in memory

BFS – Time/Space Requirements

- b = 10, 1.000.000 can be generated per sec

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

- A node requires 1000 bytes of storage

Uniform-cost Search (UCS)

- Expand least-cost (g(n)) unexpanded node
- Frontier = queue ordered by path cost
- What if all costs are the same?

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

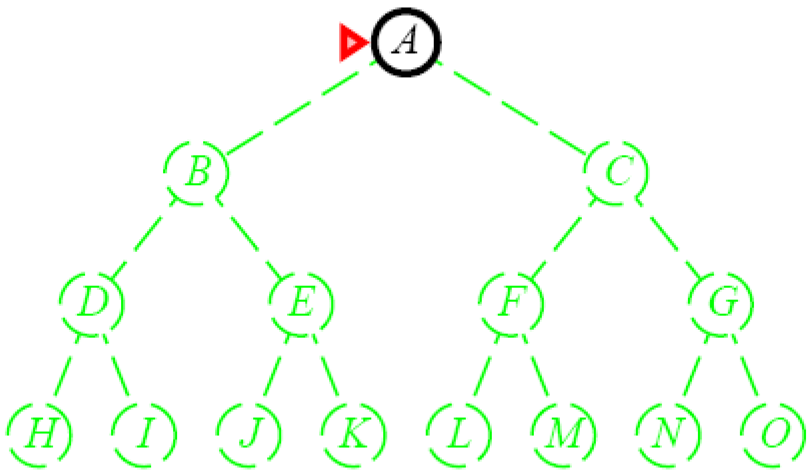
Uniform-cost Search (UCS)

- Expand least-cost unexpanded node
- Frontier = queue ordered by path cost
- What if all costs are the same?

- Complete? Yes, if step cost > ε
- Optimal? Yes,
- Time complexity? $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
- Space complexity? $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$

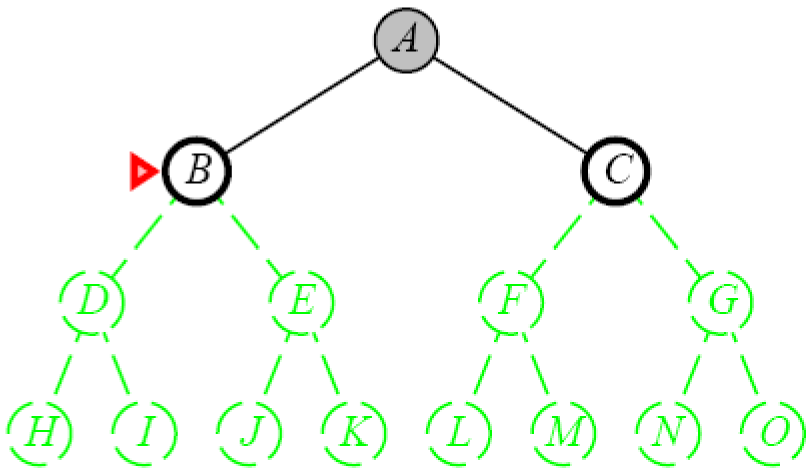
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



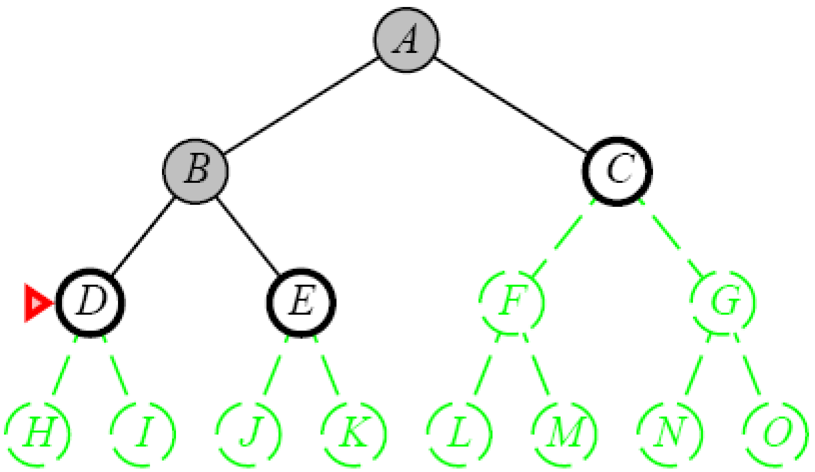
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



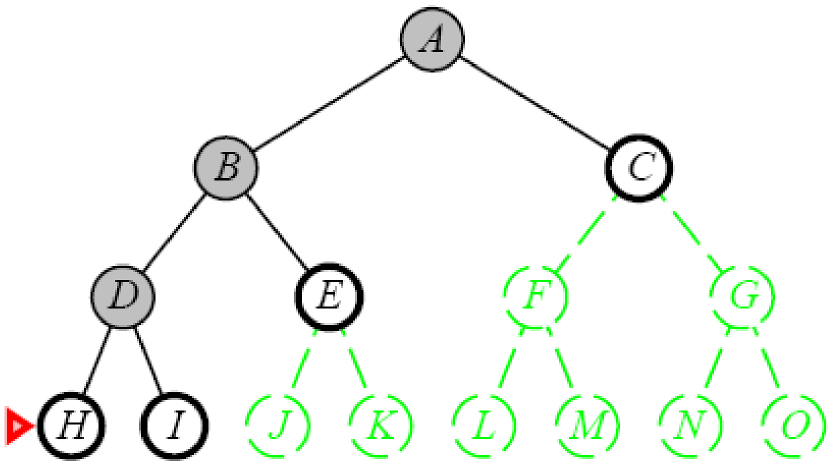
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



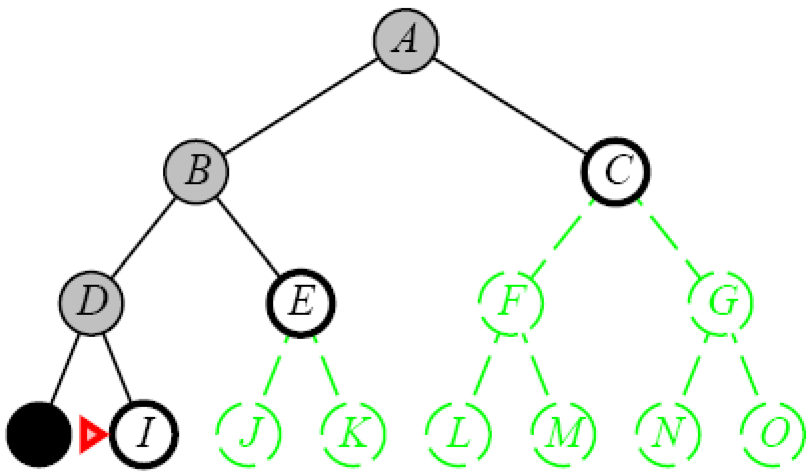
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



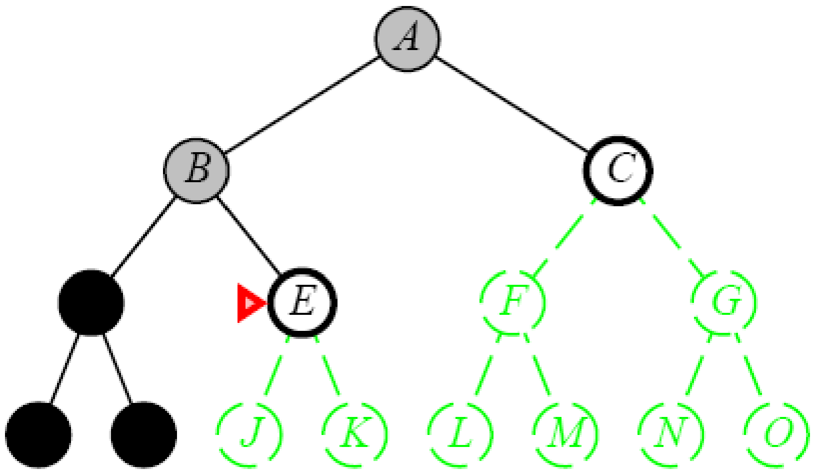
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



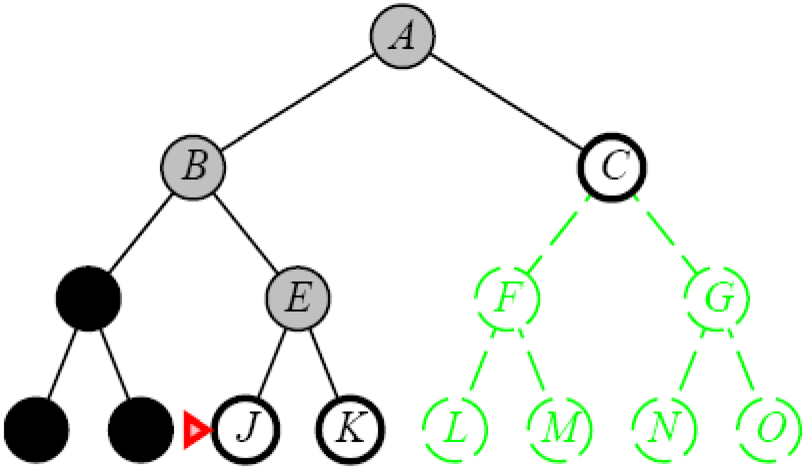
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



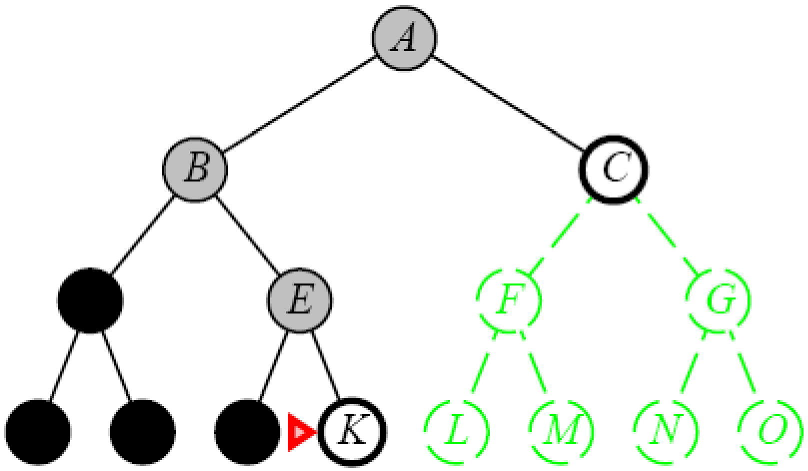
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



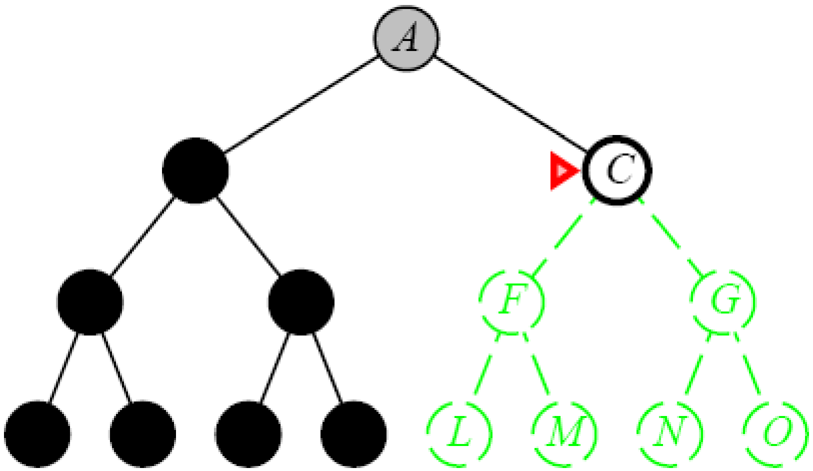
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



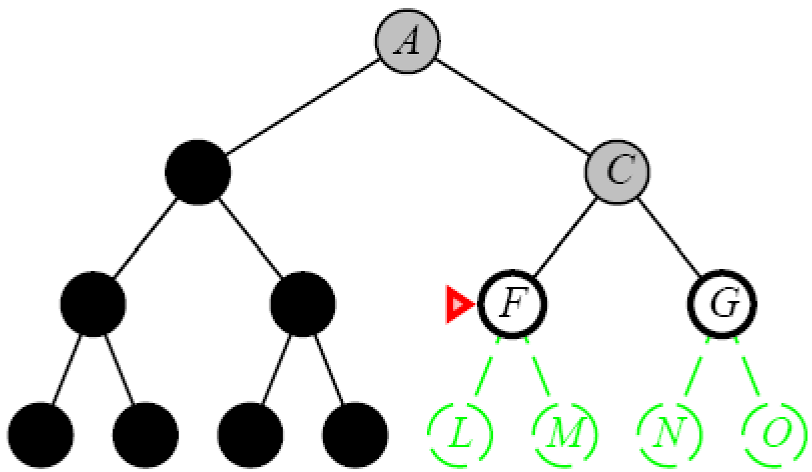
Depth-first Search (DFS)

- Expand deepest unexpanded node
- Frontier = LIFO queue



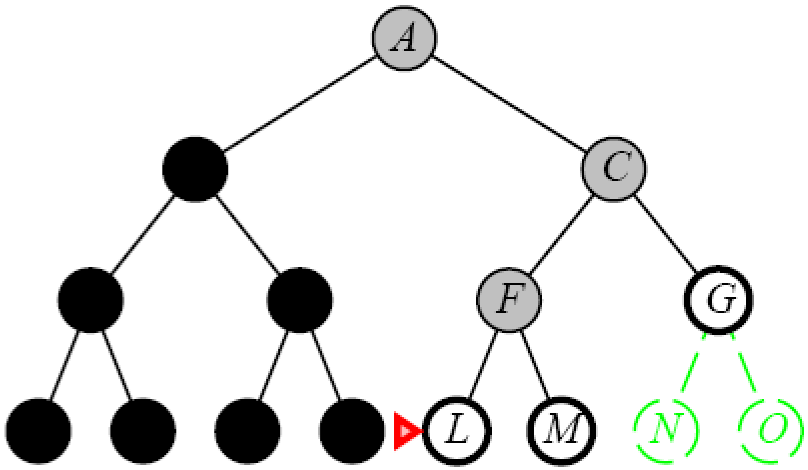
Depth-first Search

- Expand deepest unexpanded node
- Frontier = LIFO queue



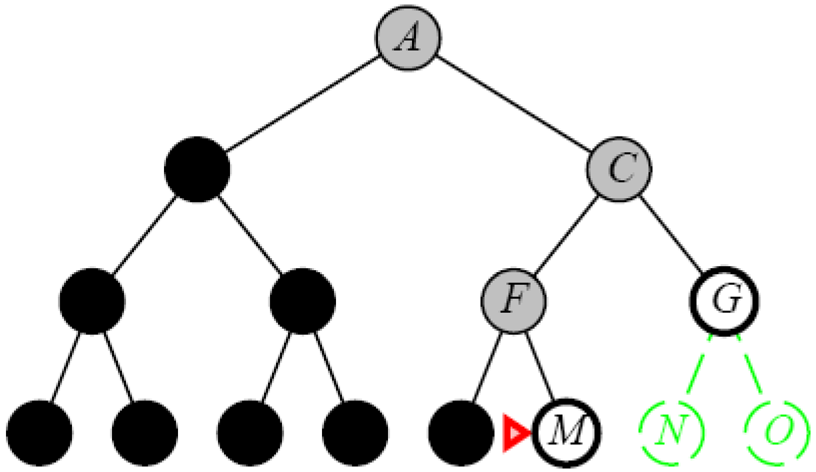
Depth-first Search

- Expand deepest unexpanded node
- Frontier = LIFO queue



Depth-first Search

- Expand deepest unexpanded node
- Frontier = LIFO queue



Properties of DFS

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

Properties of DFS

- Complete? No
- Optimal? No
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$

Depth-limited Search

- Depth-first search with depth limit: *l*

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST(problem, STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

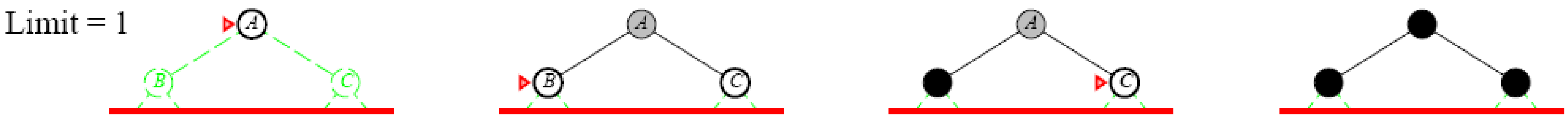
Iterative Deepening Search (IDS)

```
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
```

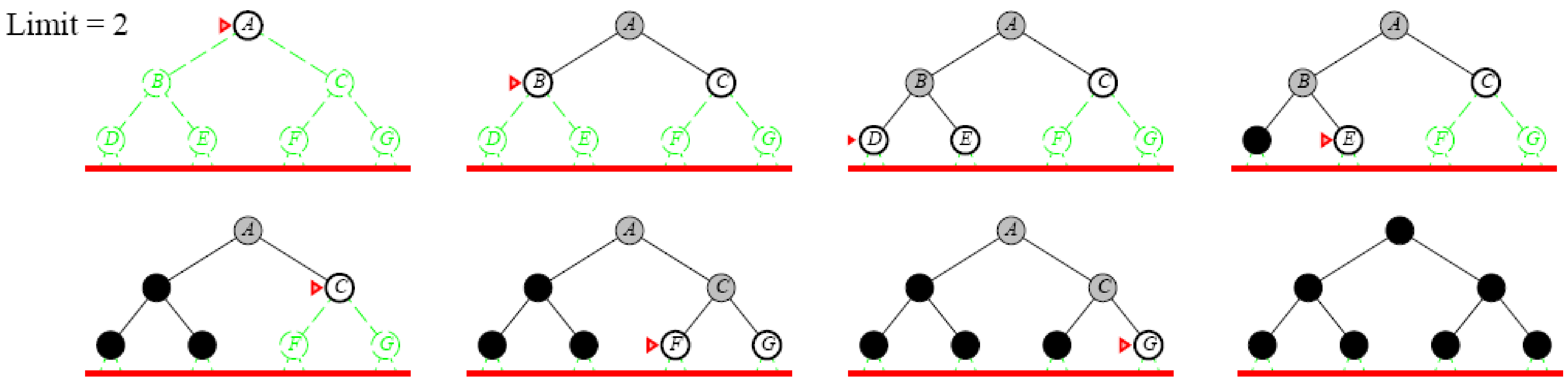
Iterative Deepening Search (IDS)

Limit = 0

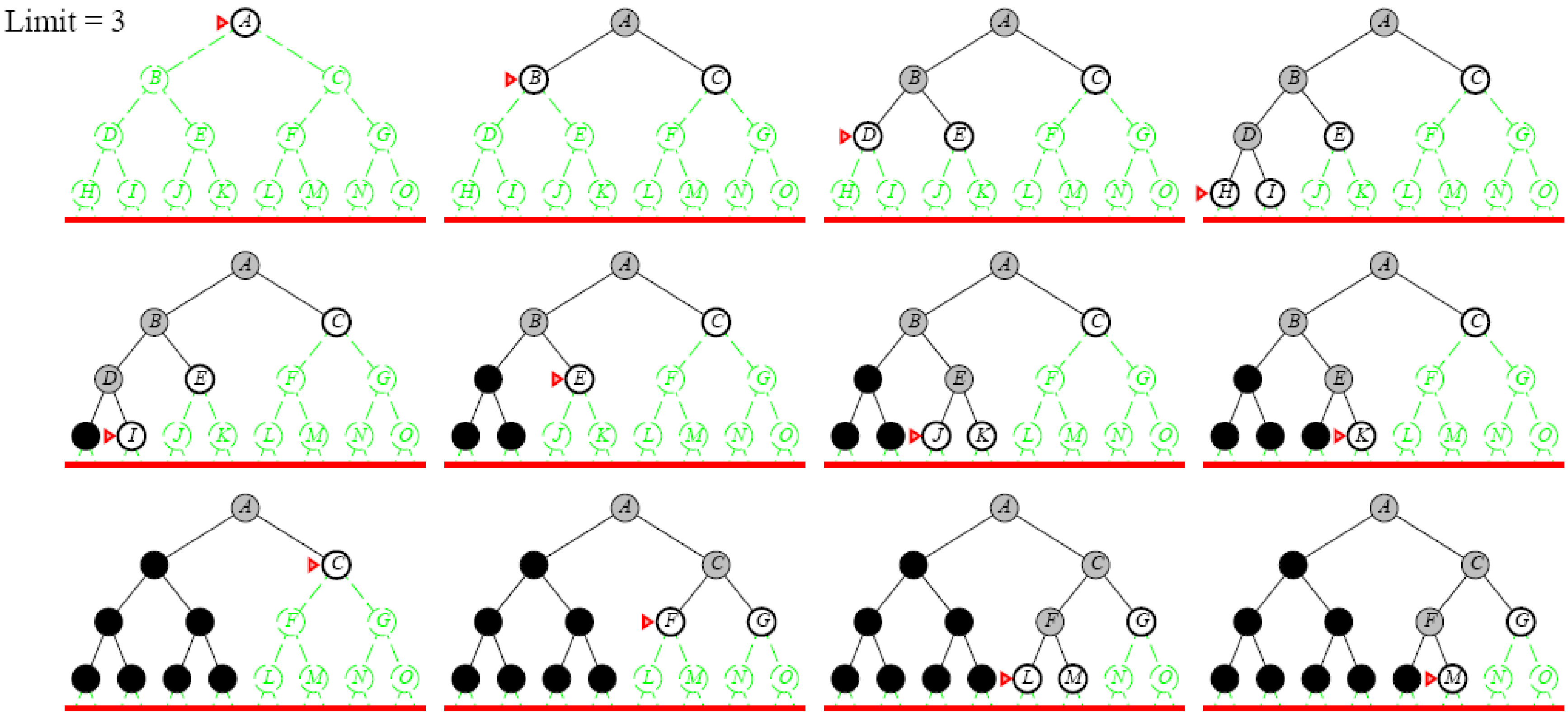
Iterative Deepening Search (IDS)



Iterative Deepening Search (IDS)



Iterative Deepening Search (IDS)



Properties of IDS

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

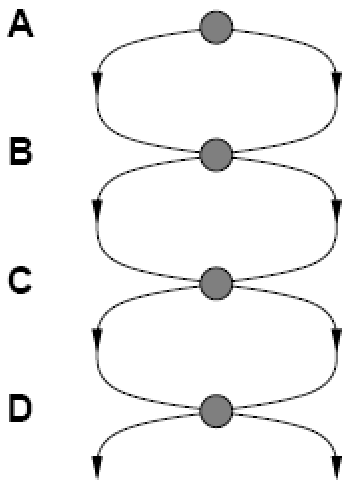
Properties of IDS

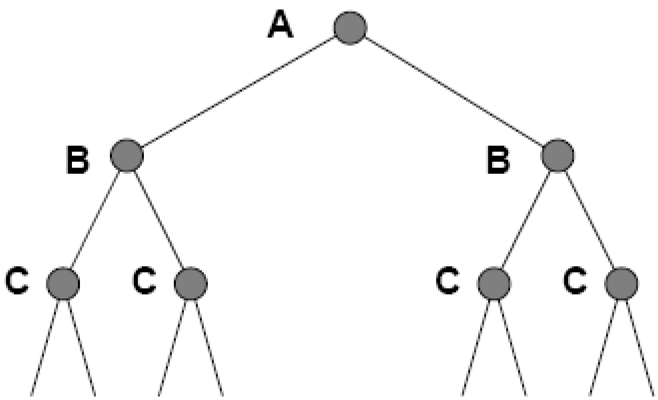
- Complete? Yes, if b is limited
- Optimal? Yes, if path cost is nondecreasing
- Time complexity? $O(b^d)$
- Space complexity? $O(bd)$

Summary of Algorithms

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

Avoiding Repeating States





General Graph-search Algorithm

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

BLG435E
Artificial Intelligence

Informed Search Strategies

Tree-Search Algorithms-Revisited

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     add the node to the explored set
     expand the chosen node, adding the resulting nodes to the frontier
        only if not in the frontier or explored set
```

Best-first Search

- Node is selected based on $f(n)$
  - Estimate of desirability
  - The node with lowest evaluation is selected

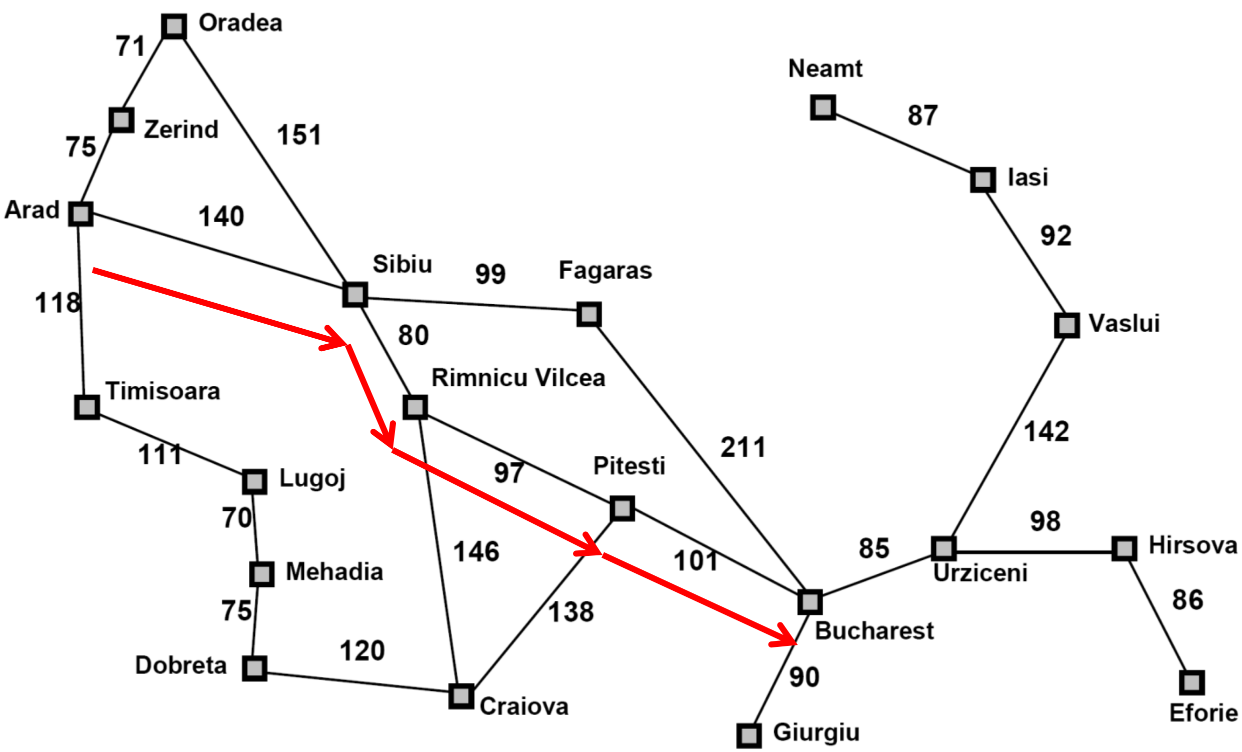
- Frontier is a queue sorted in decreasing order of desirability

- A key component: heuristic function
  - $h(n)$ = estimated cost of the cheapest path from node n to a goal node

Greedy Best-First Search

- Expands the node that is closest to the goal
- $f(n) = h(n)$
- $h_{SLD}(n) =$ Straight-line distance heuristic

Romania Problem Step Costs

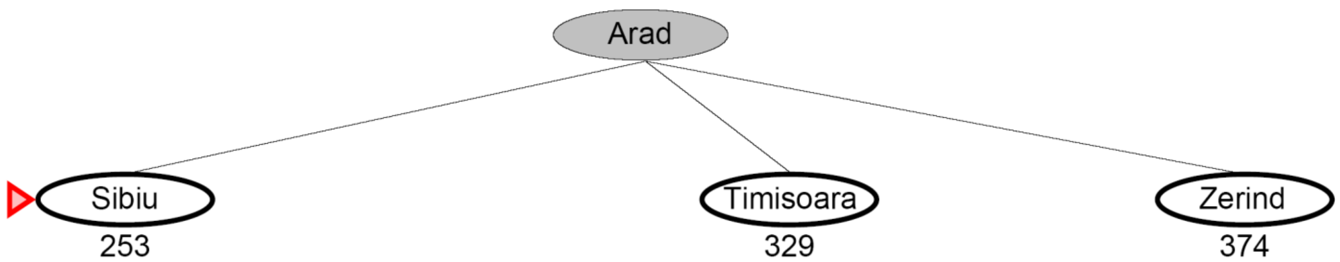


Straight−line distance to Bucharest

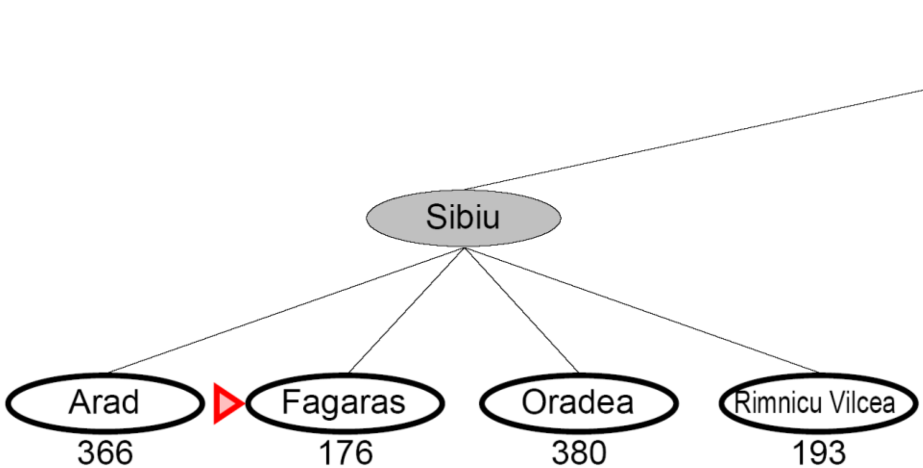
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romania Problem–Greedy Search

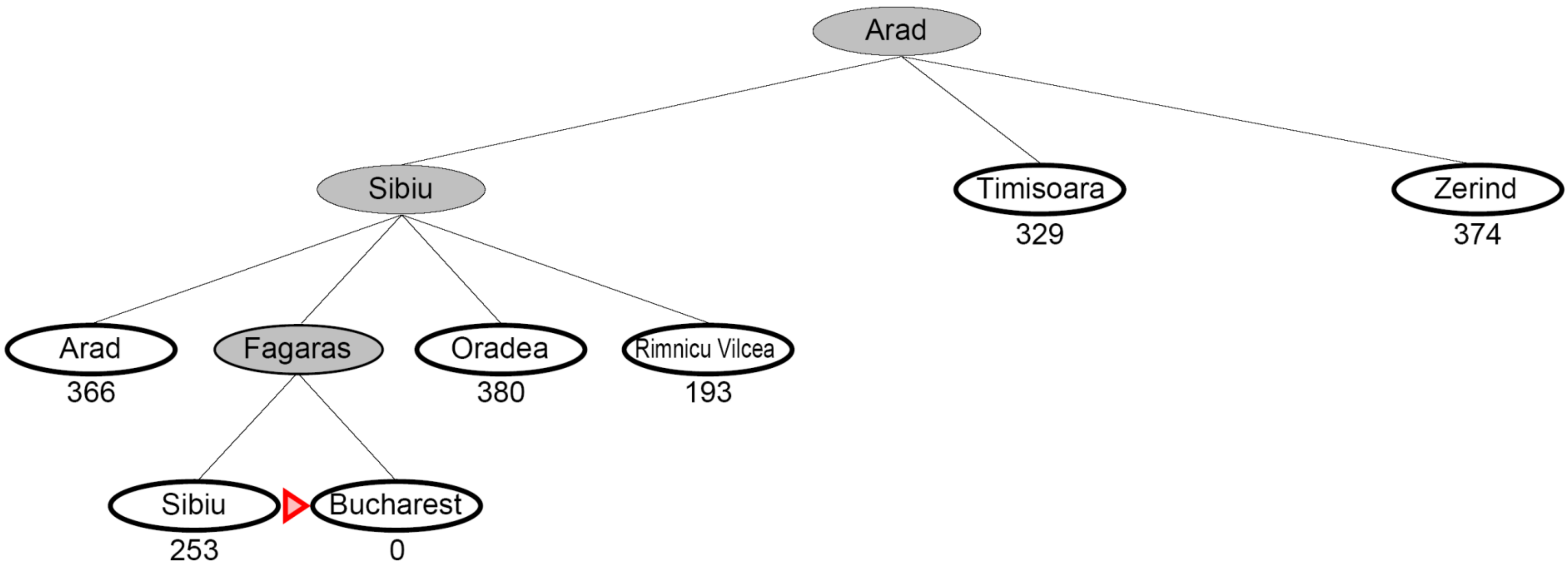
Romania Problem–Greedy Search



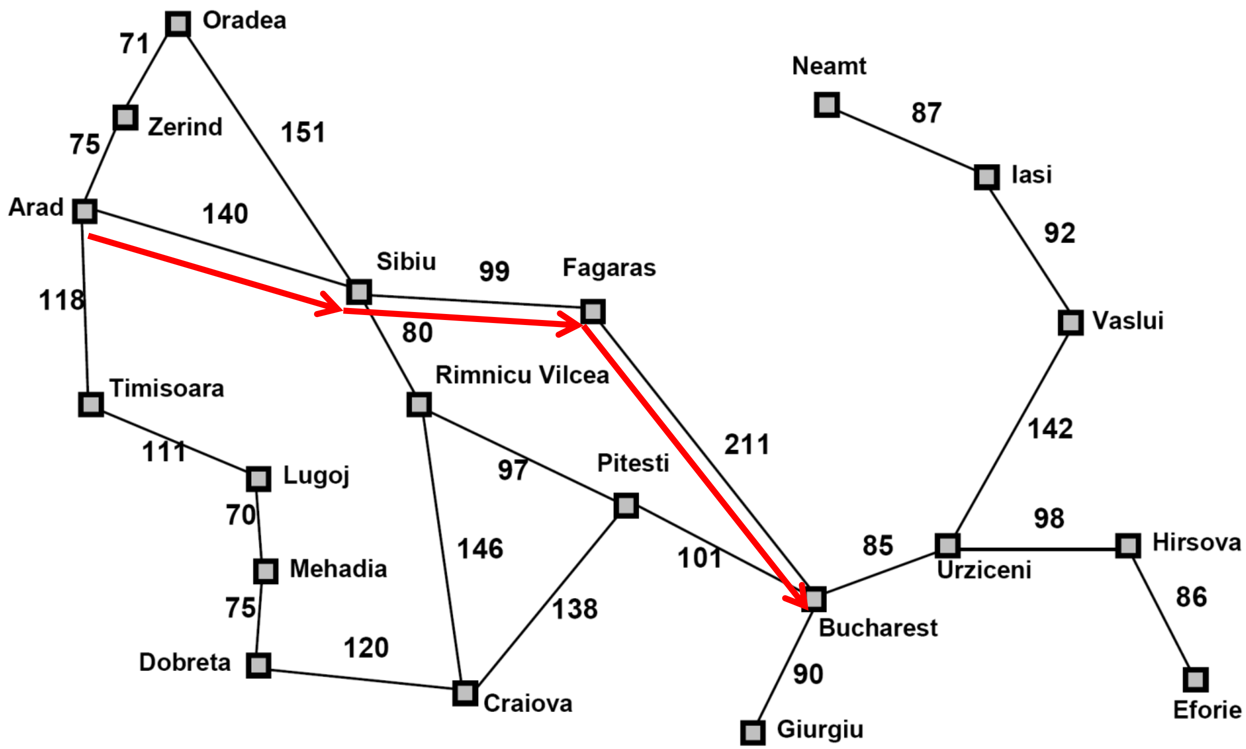
Romania Problem–Greedy Search



Romania Problem–Greedy Search



Romania Problem – Greedy Solution



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, can stuck in loops
- Time? $O(b^m)$, can be improved with a good heuristic
- Space? $O(b^m)$
- Optimality? No

A* Search

- The most well-known form of best-first search
- Avoiding paths that are already expensive

- $f(n) = g(n) + h(n)$
  - $g(n)$ = the path cost from the start node to node n

- $f(n)$ becomes the estimated cost of the cheapest solution through n

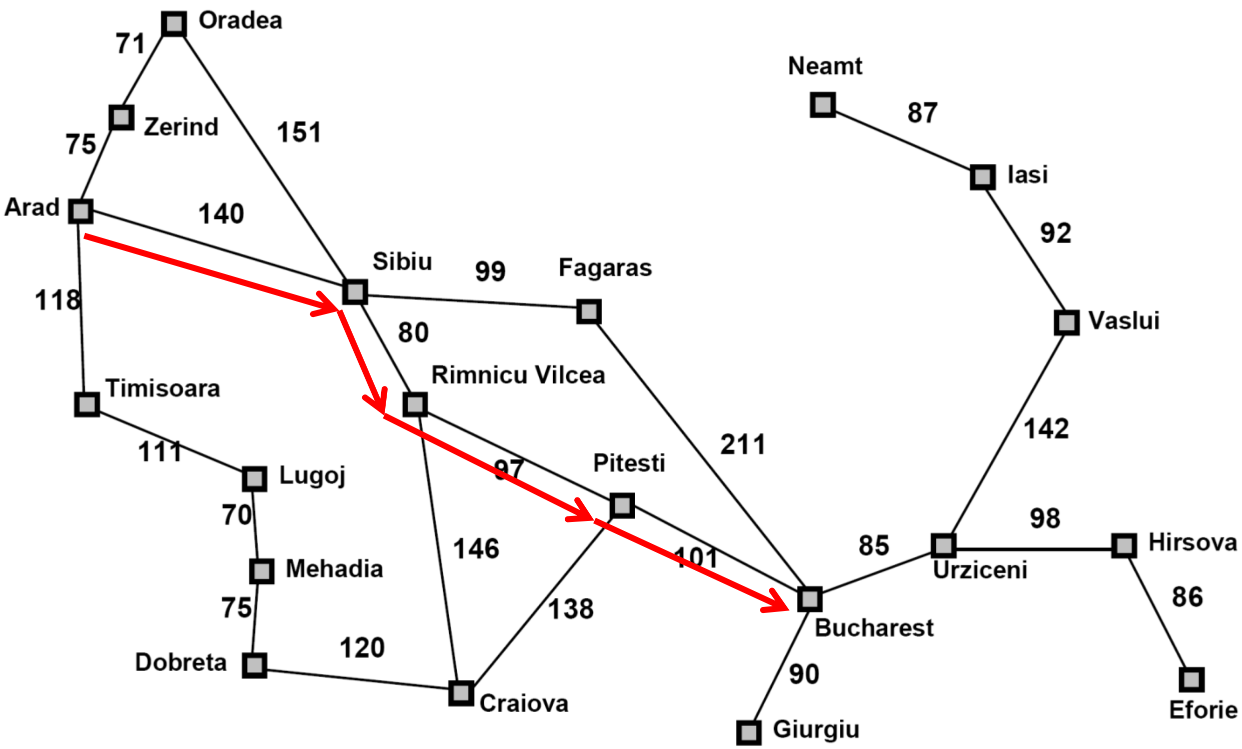
Admissible Heuristic

- Admissible heuristic:
  - $h(n) \leq h(n)^*$ (true cost from n)
  - $h(n) \geq 0$ and $h(G) = 0$

- A* (with tree-search) is optimal if $h(n)$ is admissible

- $h_{SLD}(n)$ admissible?

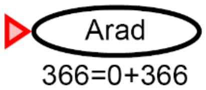
Romania Problem – A* Search



Straight−line distance to Bucharest

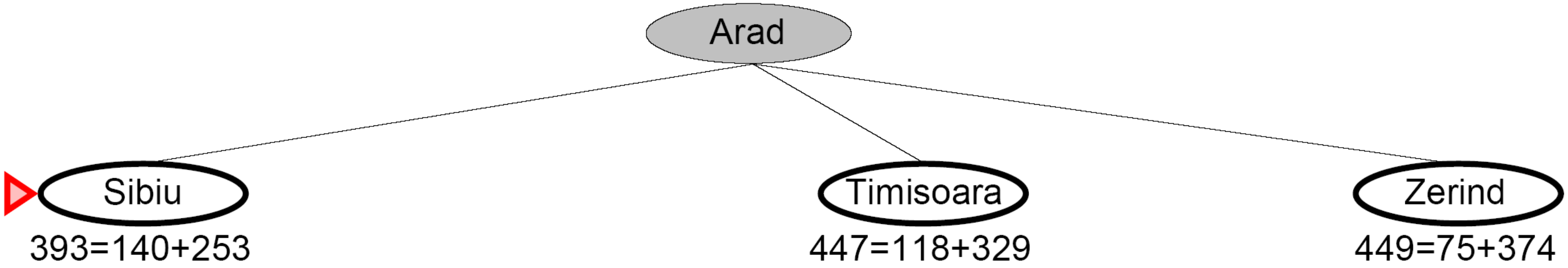
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romania Problem – A* Search

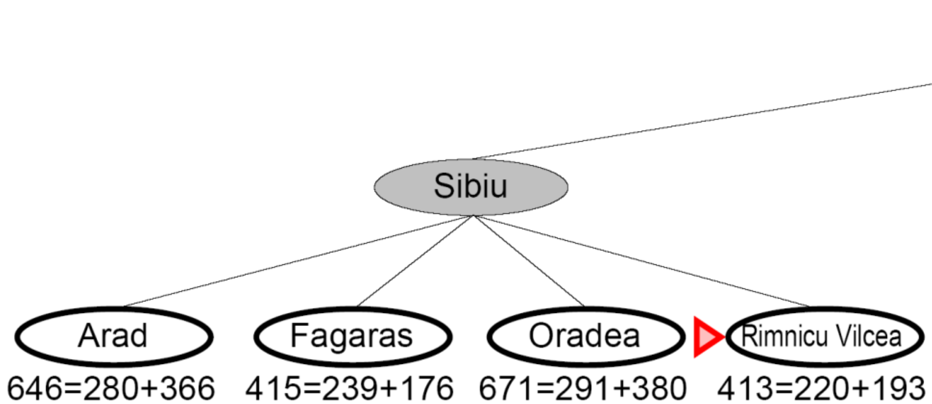


- $g(n)$ values-computed from the step costs
- $h_{SLD}(n)$ values-known initially

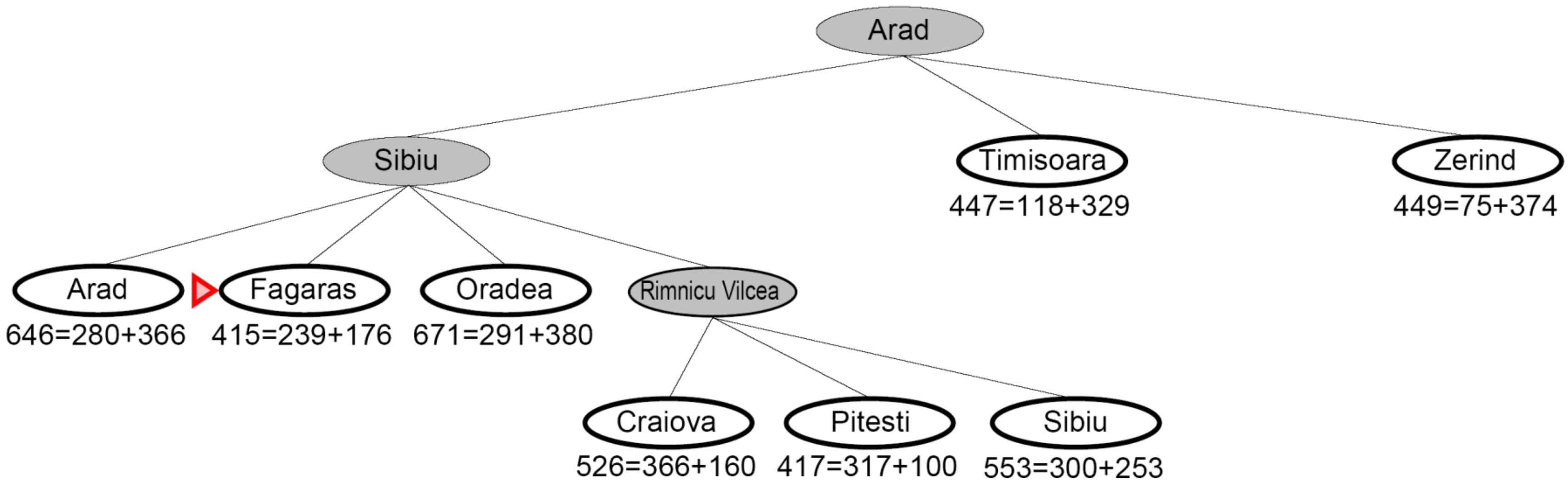
Romania Problem – A* Search



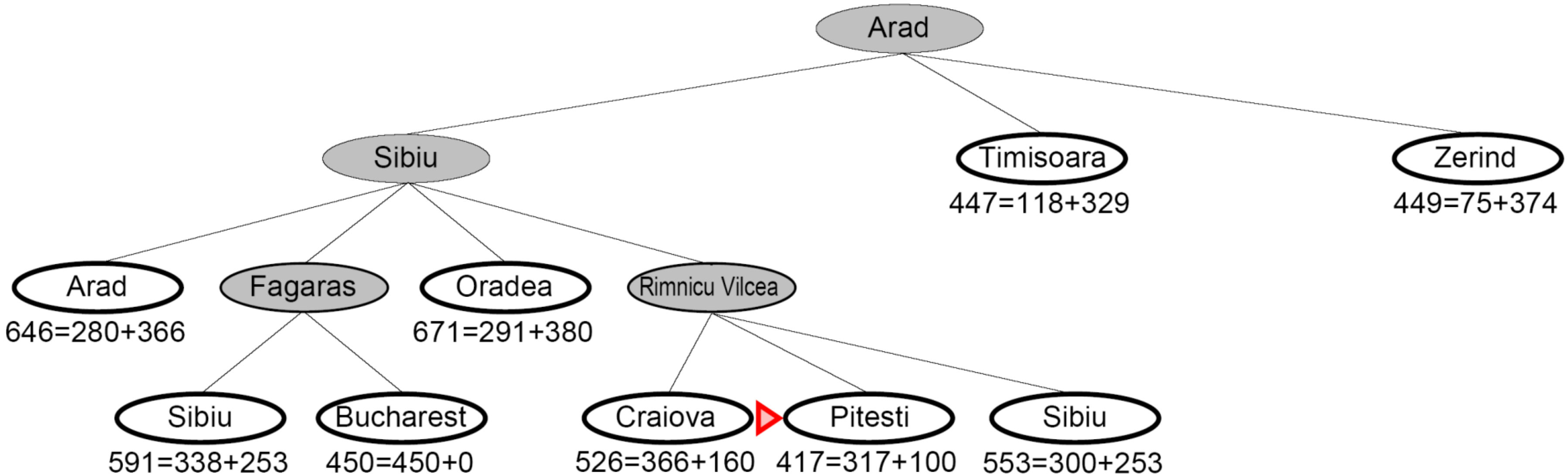
Romania Problem – A* Search



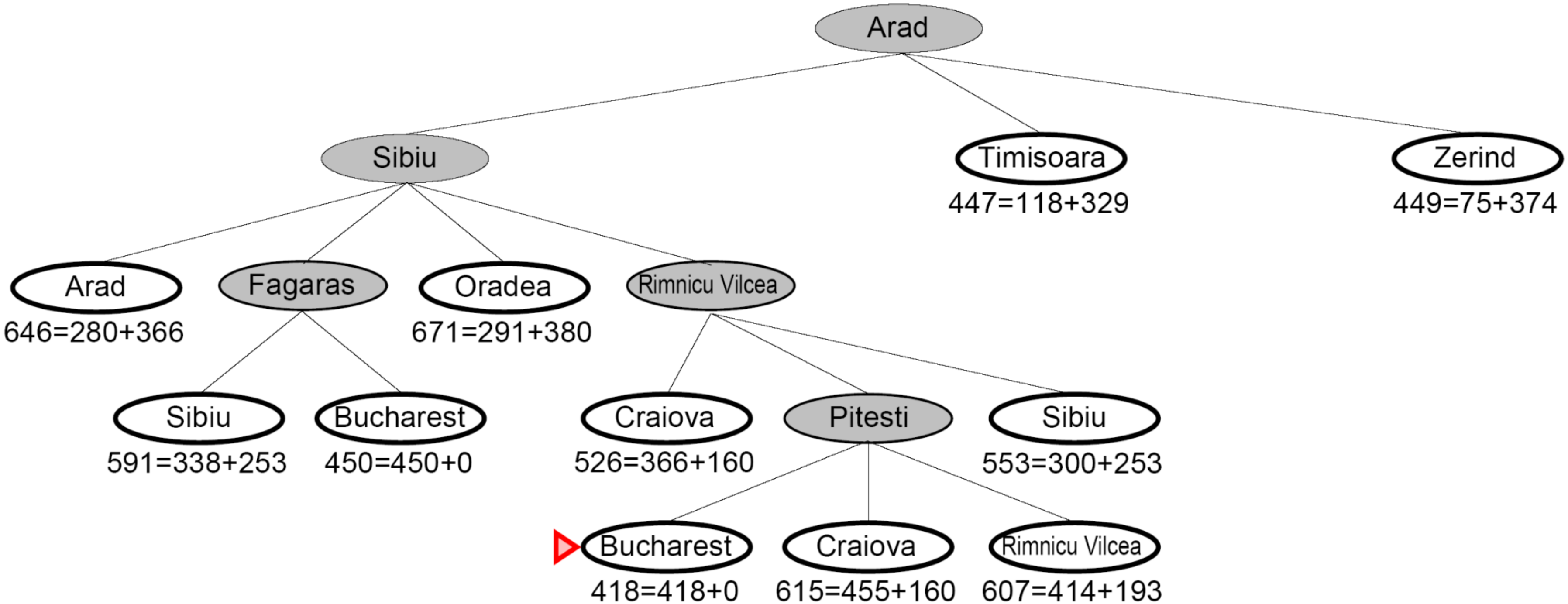
Romania Problem – A* Search



Romania Problem – A* Search

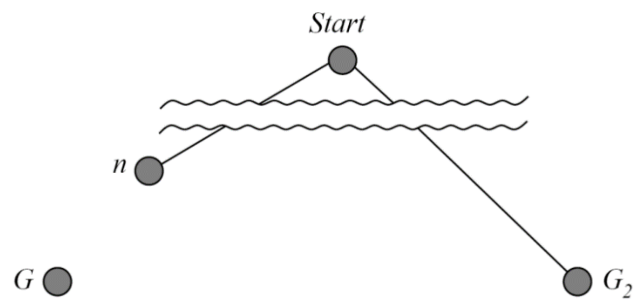


Romania Problem – A* Search



Optimality of A* (standard proof)

- Suppose a suboptimal goal G_2
- Let n be an unexpanded node on a shortest path to an optimal goal G_1



$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

- Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

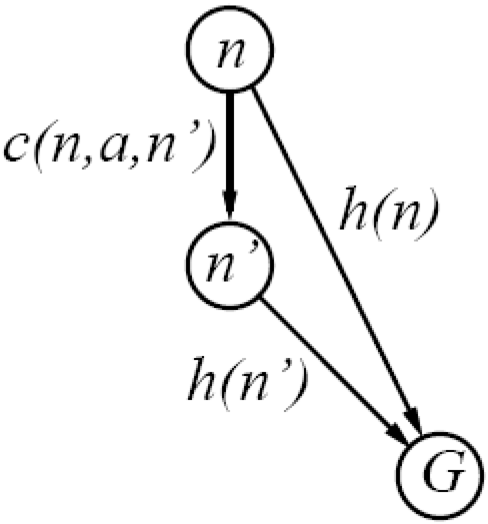
Consistency

A heuristic is consistent if

$$h(n) \leq c(n, a, n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$



I.e., $f(n)$ is nondecreasing along any path.

When do we need a consistent heuristic?

- We need an admissable and consistent heuristic for the graph search version of A*

- Why?

Properties of A*

- Complete? Yes, unless there are infinitely many nodes with f < C*
- Time? Exponential in
  - [relative error in h * length of solution]
- Space? Keeps all nodes in memory
- Optimality? Yes

To reduce the space complexity

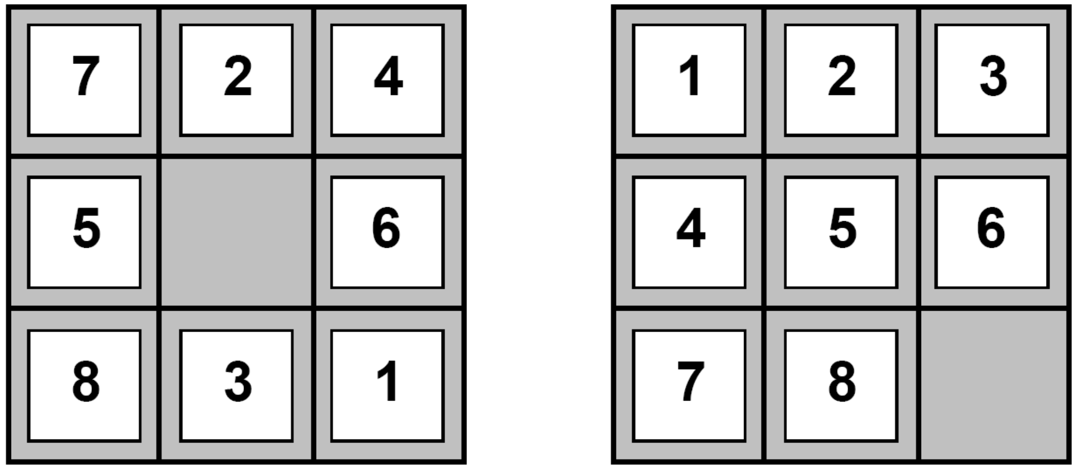
- Iterative deepening A* (IDA*)
  - Cut-off used is the f-cost

- Recursive best-first search (RBFS)
  - f_limit of best alternative path

- Memory-Bounded A* (MA*)
- Simplified MA* (SMA*)
  - drops the worst leaf node when memory is full

Admissible Heuristics

- 8-puzzle



Start State   Goal State

- $h_1(n)$ = the number of misplaced tiles
- $h_2(n)$ = the total Manhattan distance

Dominance

- if $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1, and is better for search

- Given any admissible heuristics h_a, h_b

  $$h(n) = \max(h_a(n) , h_b(n))$$

is also admissible and dominates h_a, h_b