AI

Lecture #4

Environment types

Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.

Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)

Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.

Environment types

- ✓ Static (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is semidynamic if the environment itself does not change with the passage of time but the agent's performance score does)
- ✓
- ✓ Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.
- ✓
- ✓ Single agent (vs. multiagent): An agent operating by itself in an environment.
- ✓

Environment types

		Chess with a clock		Chess without a clock		Taxi driving
Fully observable	Yes		Yes		No	
Deterministic		Strategic	Strategic	No		
Episodic		No		No		No
Static		Semi		Yes		No
Discrete	Yes		Yes		No	
Single agent		No		No		No

- The environment type largely determines the agent design
-
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent
-

What are the types of AI?

Artificial intelligence can be divided into different types on the basis of capabilities and functionalities.

Based on Capabilities:

- **Weak AI or Narrow AI:** Weak AI is capable of performing some dedicated tasks with intelligence. **Siri is an example of Weak AI**.
- **General AI:** The intelligent machines that can perform any intellectual task with efficiency as a human.
- **Strong AI:** It is the hypothetical concept that involves the machine that will be better than humans and will surpass human intelligence.

What are the types of AI?

Based on Functionalities:

•**Reactive Machines:** Purely reactive machines are the basic types of AI. These focus on the present actions and cannot store the previous actions**.**

•**Limited Memory:** As its name suggests, it can store the past data or experience for the limited duration. The self-driving car is an example of such AI types.

•**Theory of Mind:** It is the advanced AI that is capable of understanding human emotions, people, etc., in the real world.

•**Self-Awareness:** Self Awareness AI is the future of Artificial Intelligence that will have their own consciousness, emotions, similar to humans.

What are the different areas where AI has a great impact?

- Autonomous Transportation
- Education-system powered by AI.
- Healthcare
- Predictive Policing
- Space Exploration
- Entertainment, etc.

Explain rational agents and rationality?

A rational agent is an agent that has clear preferences, model uncertainty, and that performs the right actions always. A rational agent is able to take the best possible action in any situation.

Review: Search problem formulation

- Initial state
- Actions
- Transition model
- Goal state
- Path cost

- What is the optimal solution?
- What is the state space?

Review: Tree search

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

Search strategies

- A **search strategy** is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - **Completeness:** does it always find a solution if one exists?
  - **Optimality:** does it always find a least-cost solution?
  - **Time complexity:** number of nodes generated
  - **Space complexity:** maximum number of nodes in memory
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the optimal solution
  - *m*: maximum length of any path in the state space (may be infinite)

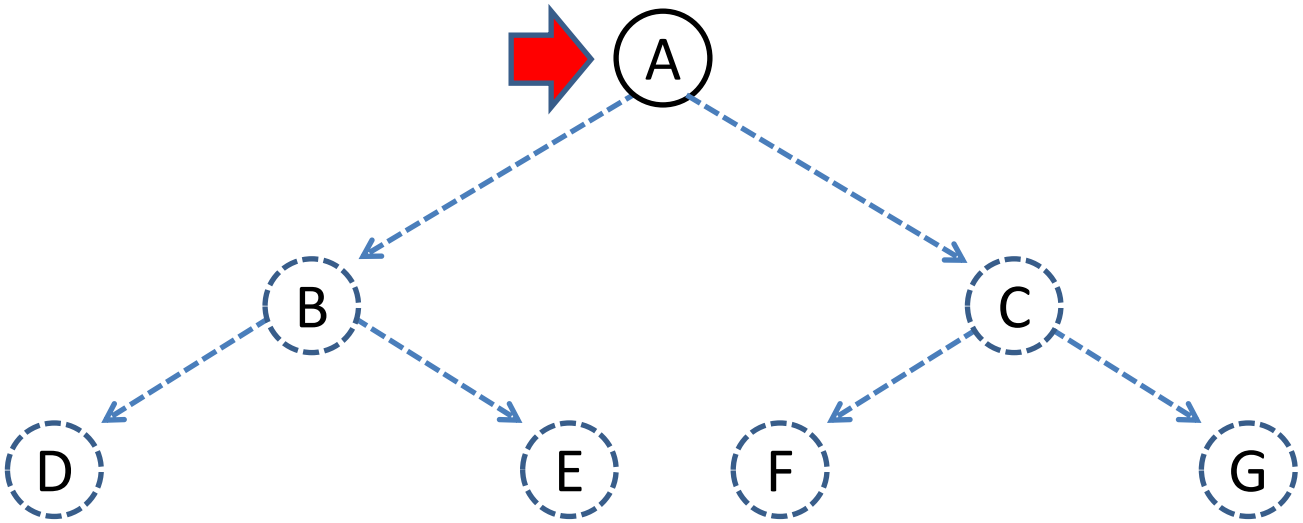
Uninformed search strategies

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

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

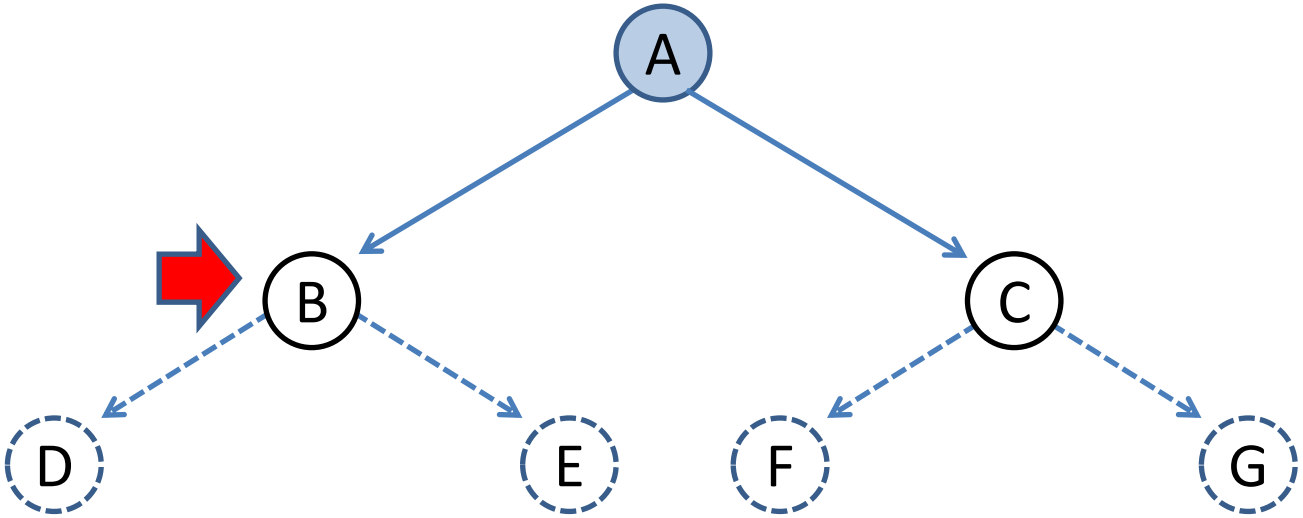
Breadth-first search

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



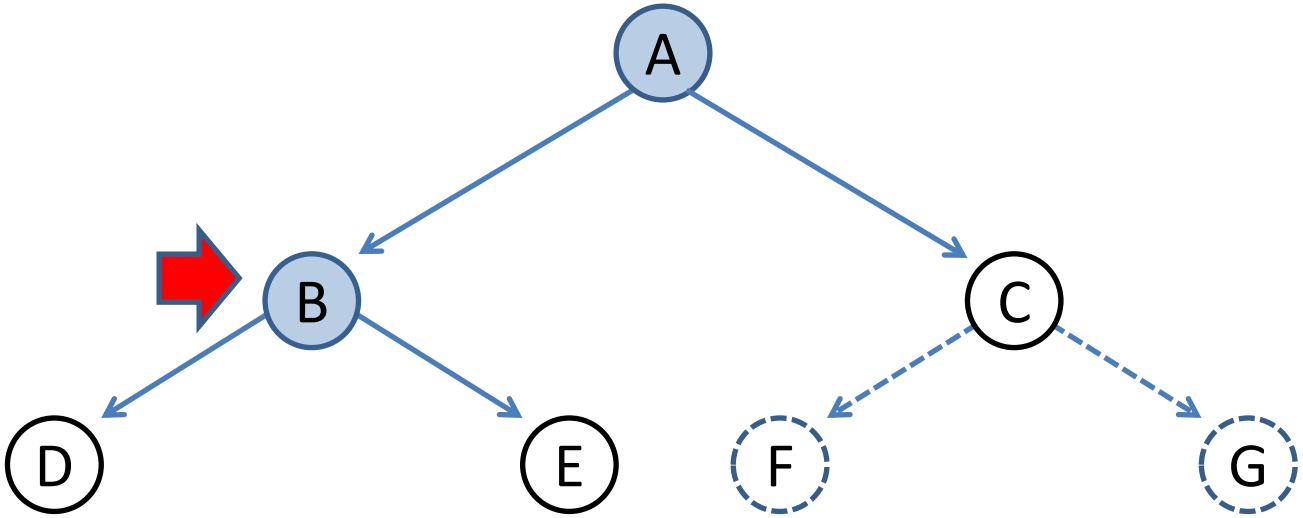
Breadth-first search

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



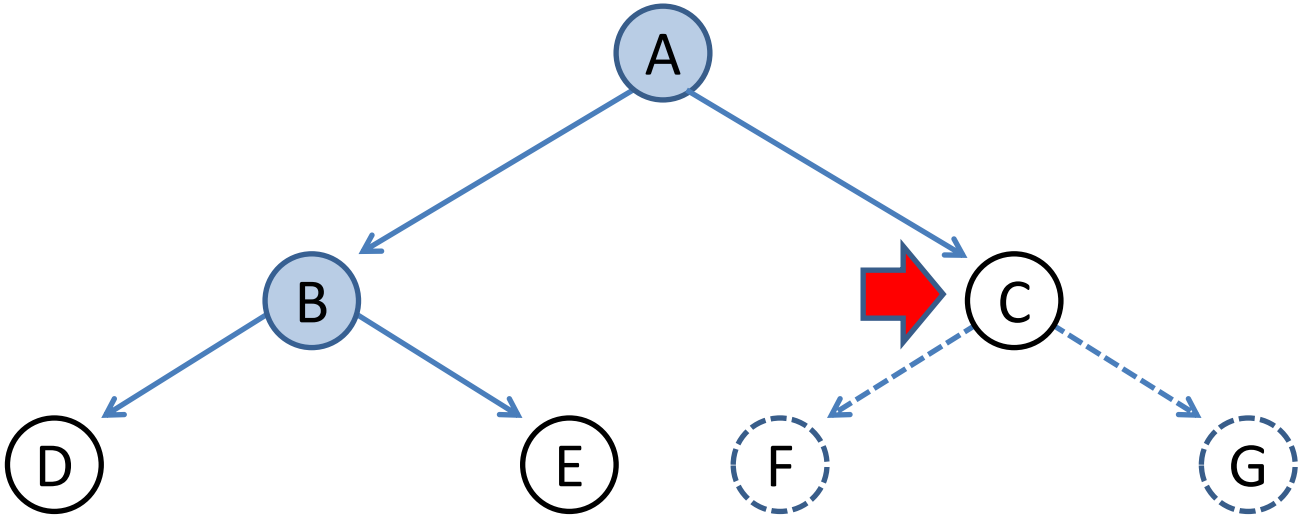
Breadth-first search

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



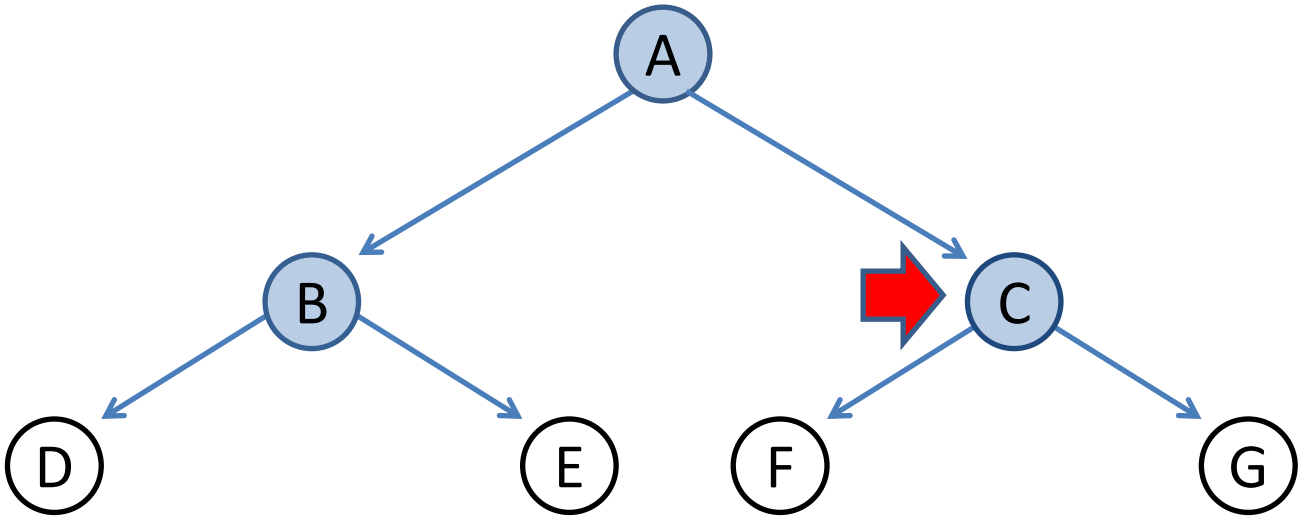
Breadth-first search

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



Breadth-first search

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



Properties of breadth-first search

- **Complete?**

  Yes (if branching factor b is finite)
- **Optimal?**

  Yes – if cost = 1 per step
- **Time?**

  Number of nodes in a b-ary tree of depth d: $O(b^d)$

  (d is the depth of the optimal solution)
- **Space?**

  $O(b^d)$

- Space is the bigger problem (more than time)

Uniform-cost search

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

- **Complete?**

  Yes, if step cost is greater than some positive constant ε
- **Optimal?**

  Yes – nodes expanded in increasing order of path cost
- **Time?**

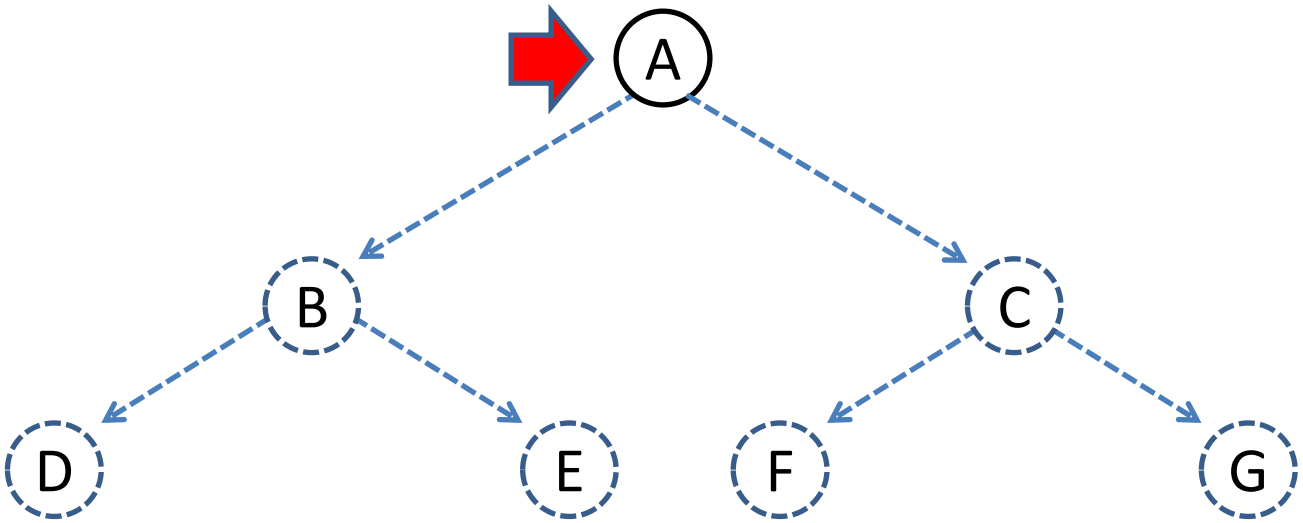
  Number of nodes with path cost ≤ cost of optimal solution (C^*), $O(b^{C^*/\varepsilon})$

  This can be greater than $O(b^d)$: the search can explore long paths consisting of small steps before exploring shorter paths consisting of larger steps
- **Space?**

  $O(b^{C^*/\varepsilon})$

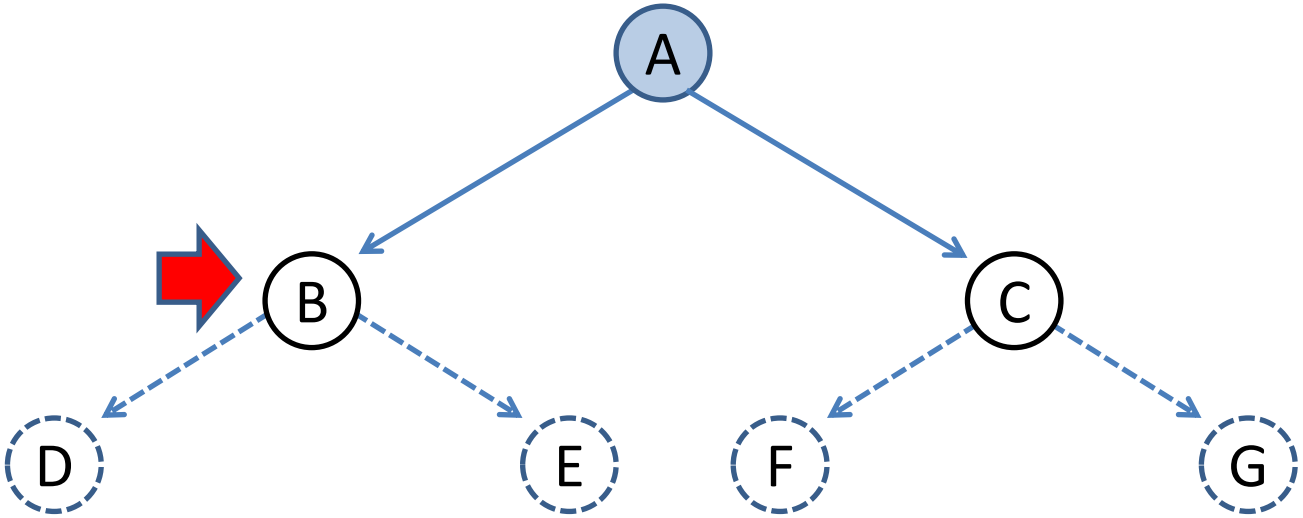
Depth-first search

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



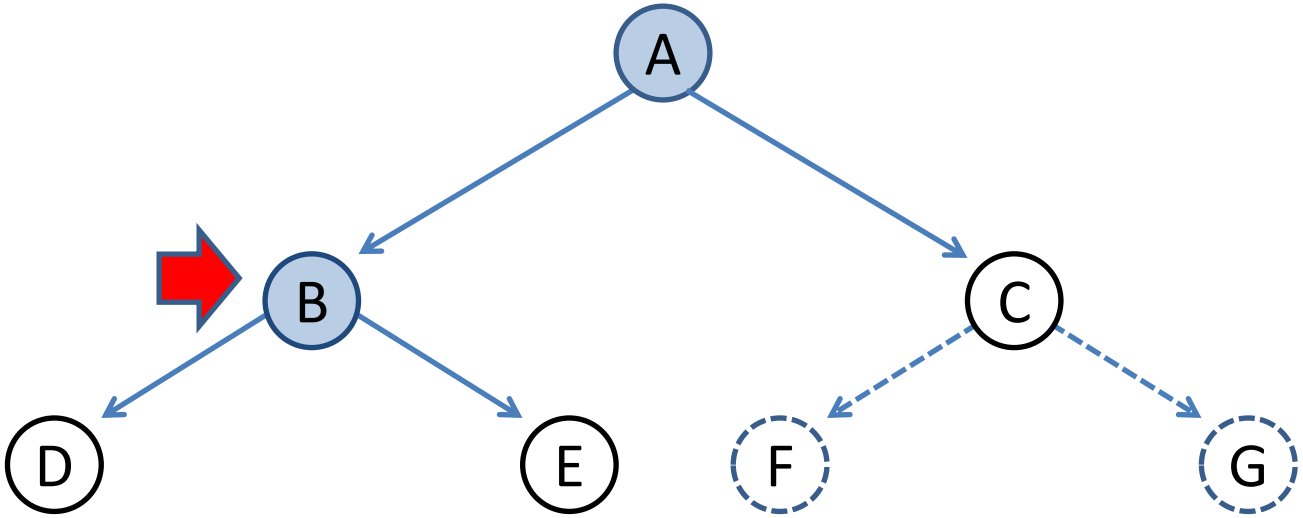
Depth-first search

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



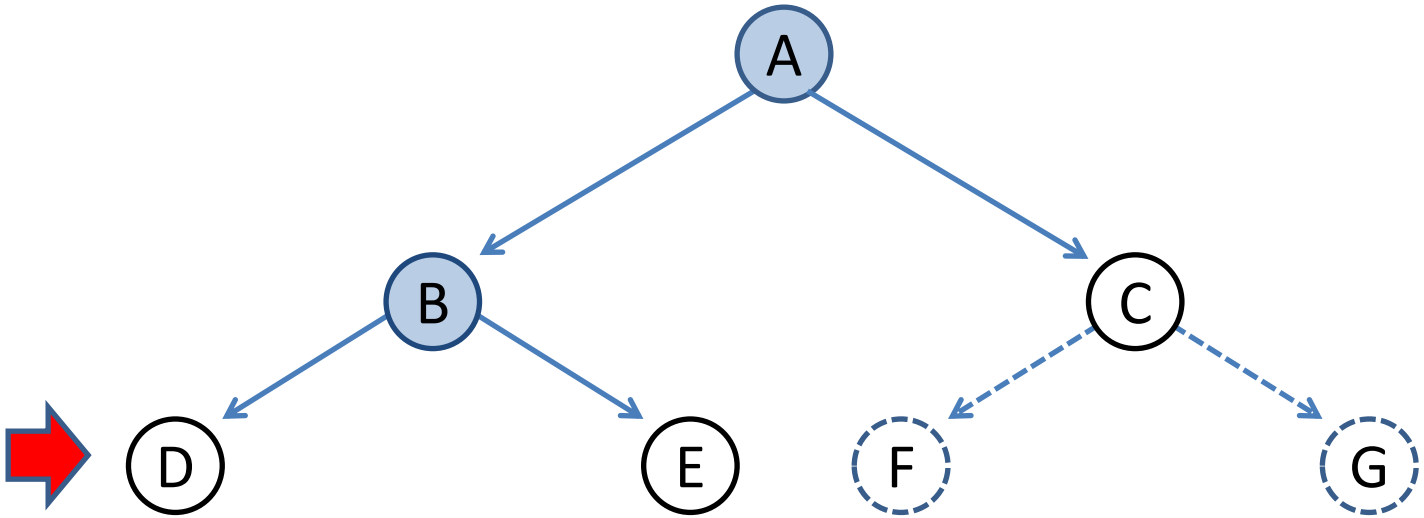
Depth-first search

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



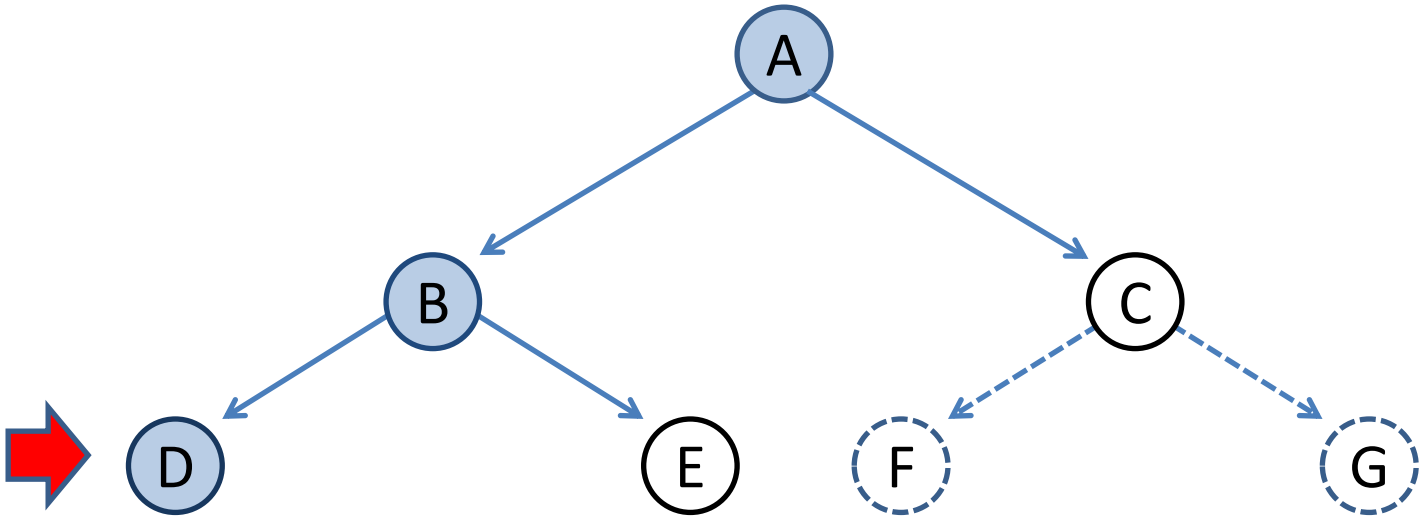
Depth-first search

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



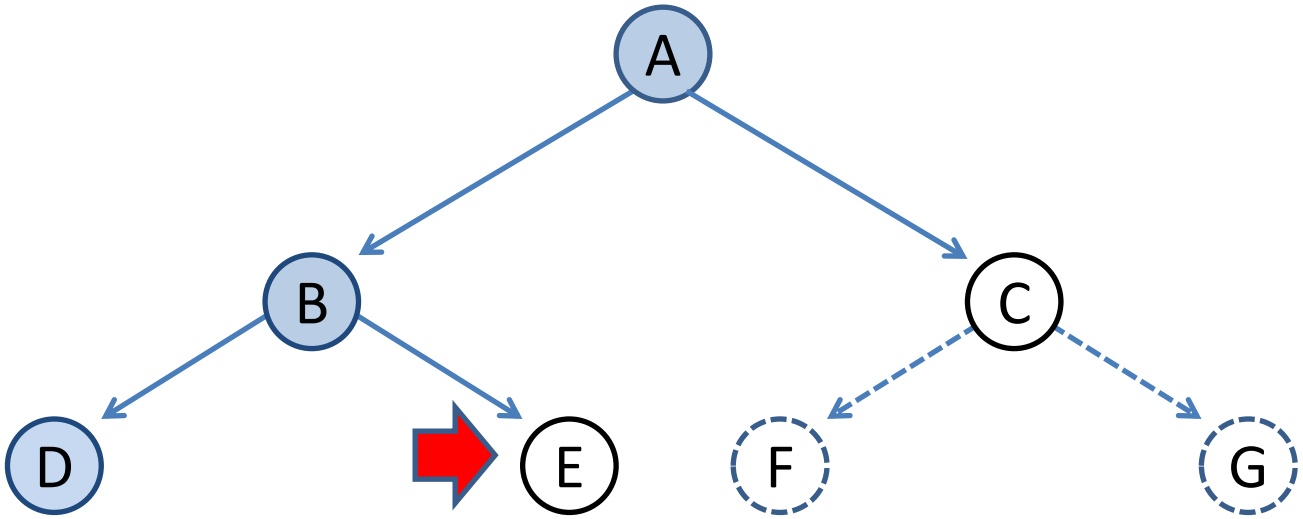
Depth-first search

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



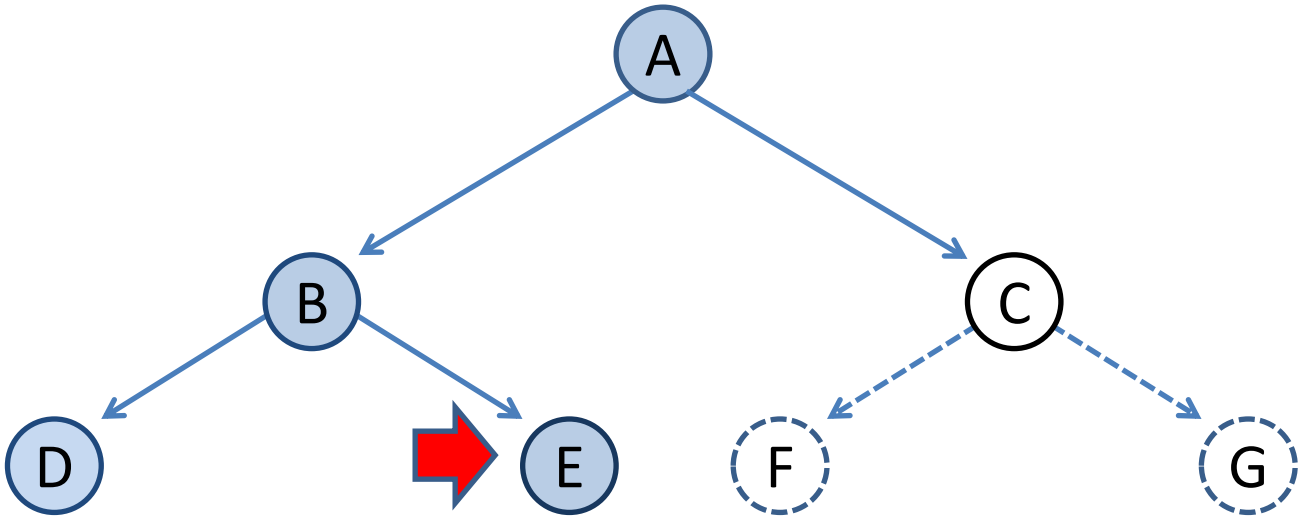
Depth-first search

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



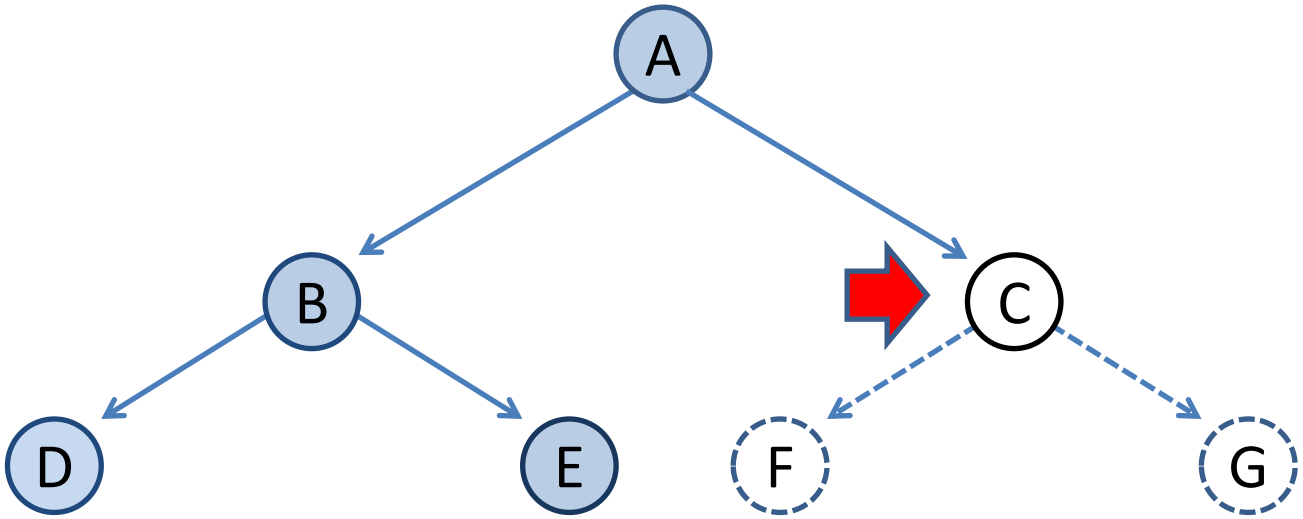
Depth-first search

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



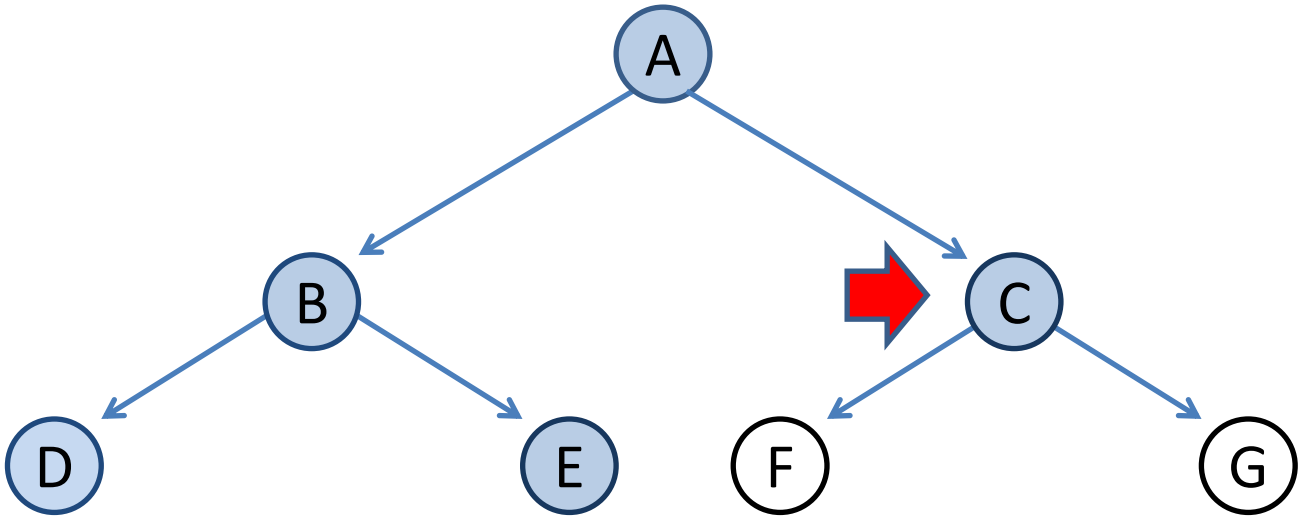
Depth-first search

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



Depth-first search

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



Properties of depth-first search

- **Complete?**

  Fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path

  → complete in finite spaces

- **Optimal?**

  No – returns the first solution it finds

- **Time?**

  Could be the time to reach a solution at maximum depth m: $O(b^m)$

  Terrible if m is much larger than d

  But if there are lots of solutions, may be much faster than BFS

- **Space?**

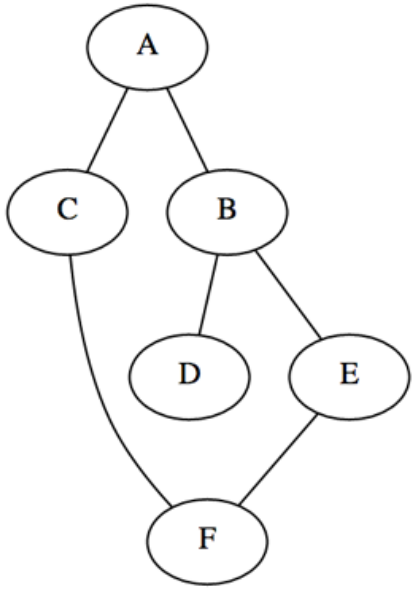
  $O(bm)$, i.e., linear space!

Depth-First and Breath-First search to achieve the goals highlighted below:

- Find all vertices in a subject vertices connected component.
- Return all available paths between two vertices.
- And in the case of BFS, return the shortest path (length measured by number of path edges).

There are two popular options for representing a graph, the first being an adjacency matrix (effective with dense graphs) and second an adjacency list

```
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E'])}
```



•Mark the current vertex as being visited.

•Explore each adjacent vertex that is not included in the visited set.

Connected Component

The implementation below uses the stack data-structure to build-up and return a set of vertices that are accessible within the subjects connected component. Using Python's overloading of the subtraction operator to remove items from a set, we are able to add only the unvisited adjacent vertices.

```
def dfs(graph, start):
    visited, stack = set(), [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            stack.extend(graph[vertex] - visited)
    return visited

dfs(graph, 'A') # {'E', 'D', 'F', 'A', 'C', 'B'}
```

Another Python language detail is that function variables are passed by reference, resulting in the visited mutable set not having to reassigned upon each recursive call.

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

dfs(graph, 'C') # {'E', 'D', 'F', 'A', 'C', 'B'}
```

Paths

We are able to tweak both of the previous implementations to return all possible paths between a start and goal vertex. The implementation below uses the stack data-structure again to iteratively solve the problem, yielding each possible path when we locate the goal. Using a generator allows the user to only compute the desired amount of alternative paths.

```
def dfs_paths(graph, start, goal):
    stack = [(start, [start])]
    while stack:
        (vertex, path) = stack.pop()
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                stack.append((next, path + [next]))

list(dfs_paths(graph, 'A', 'F')) # [['A', 'C', 'F'], ['A', 'B', 'E',
```

Iterative deepening search

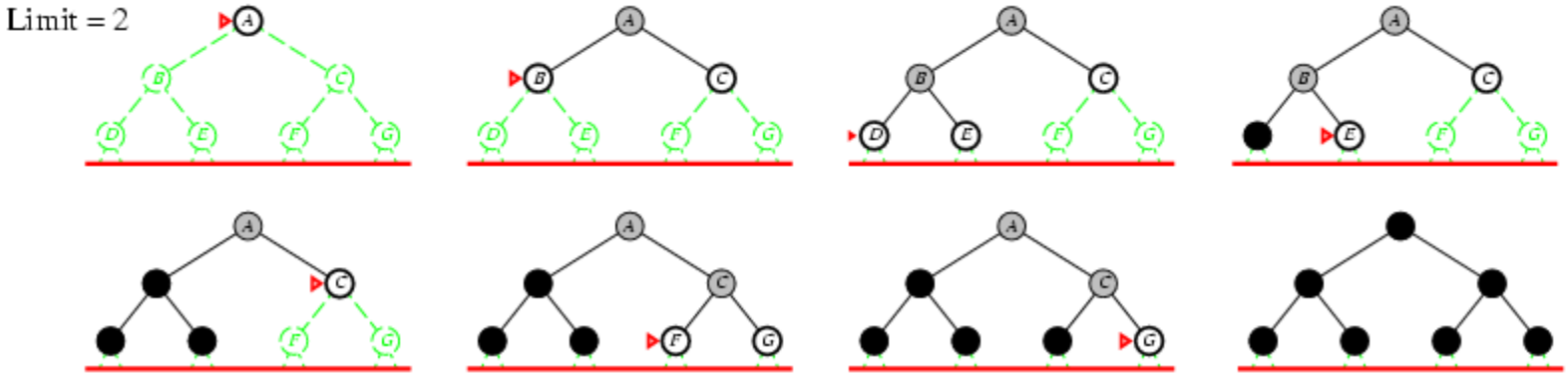
- Use DFS as a subroutine
  1. Check the root
  2. Do a DFS searching for a path of length 1
  3. If there is no path of length 1, do a DFS searching for a path of length 2
  4. If there is no path of length 2, do a DFS searching for a path of length 3…

Iterative deepening search

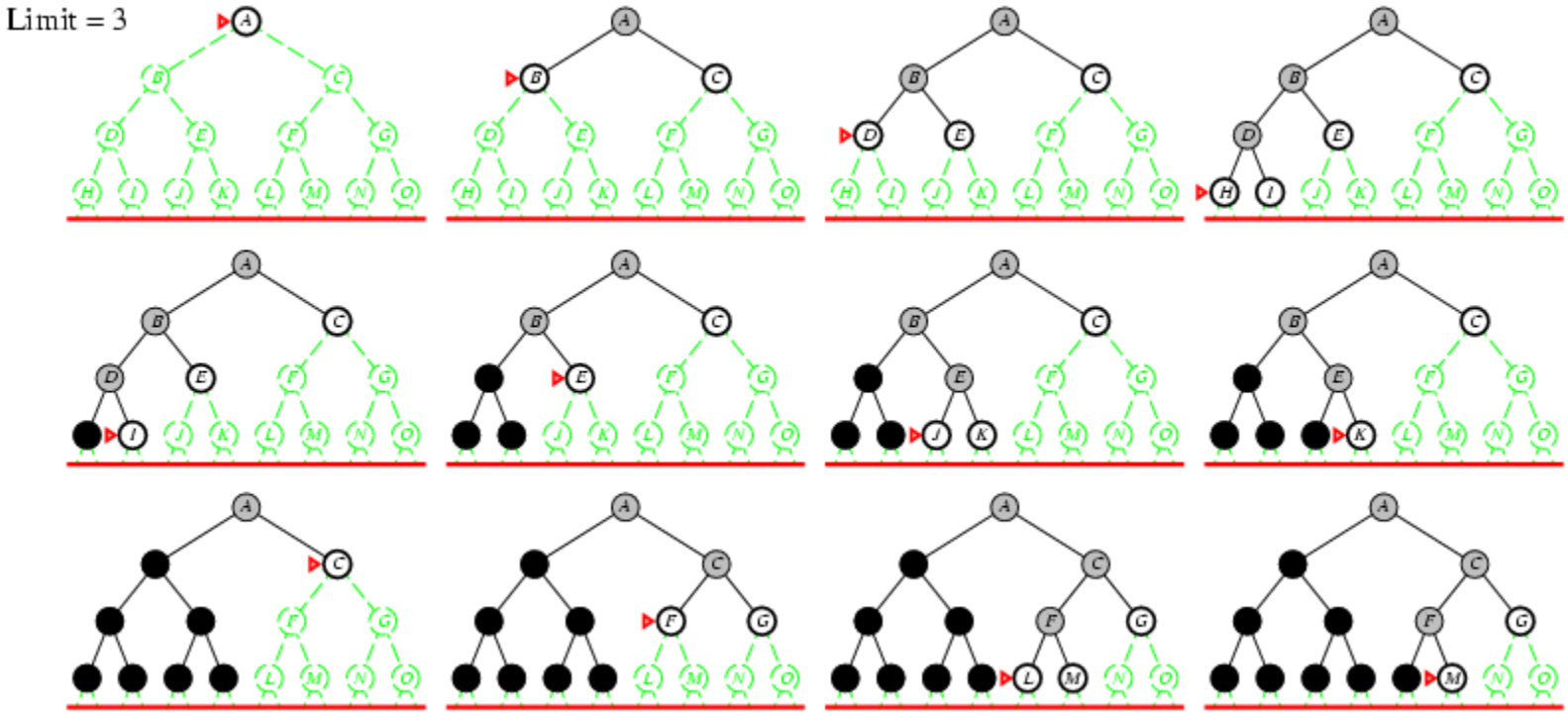
Limit = 0

Iterative deepening search

Iterative deepening search



Iterative deepening search



Properties of iterative deepening search

- **Complete?**

  Yes

- **Optimal?**

  Yes, if step cost = 1

- **Time?**

  $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

- **Space?**

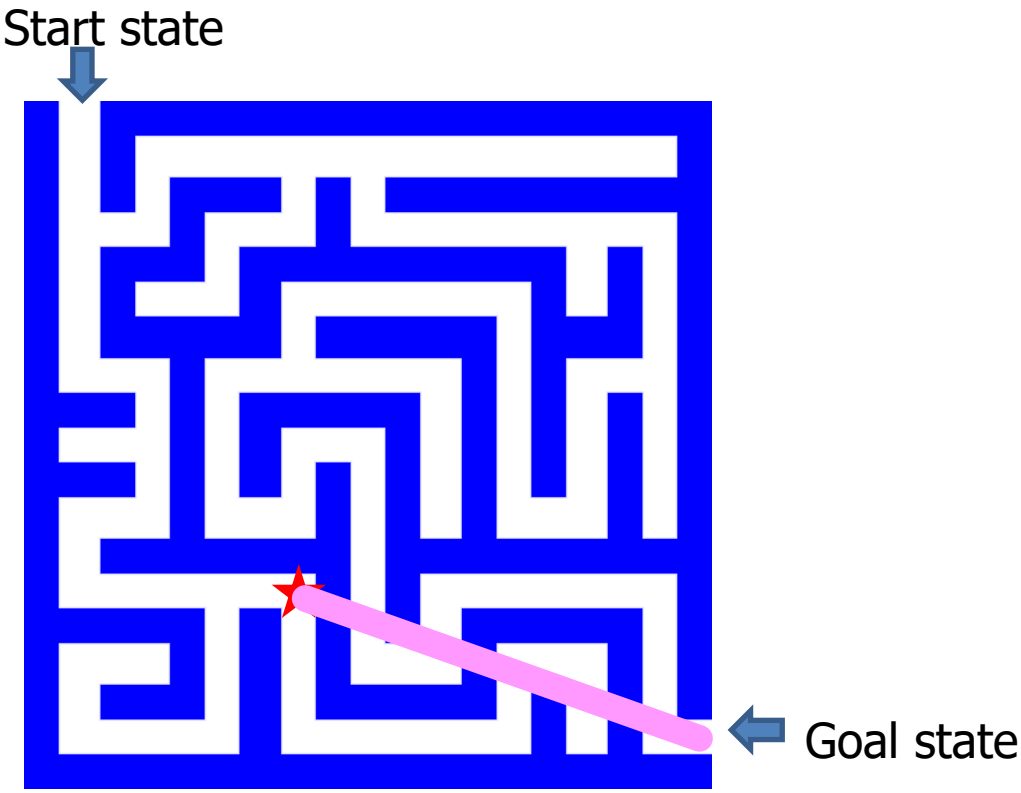
  $O(bd)$

Informed search

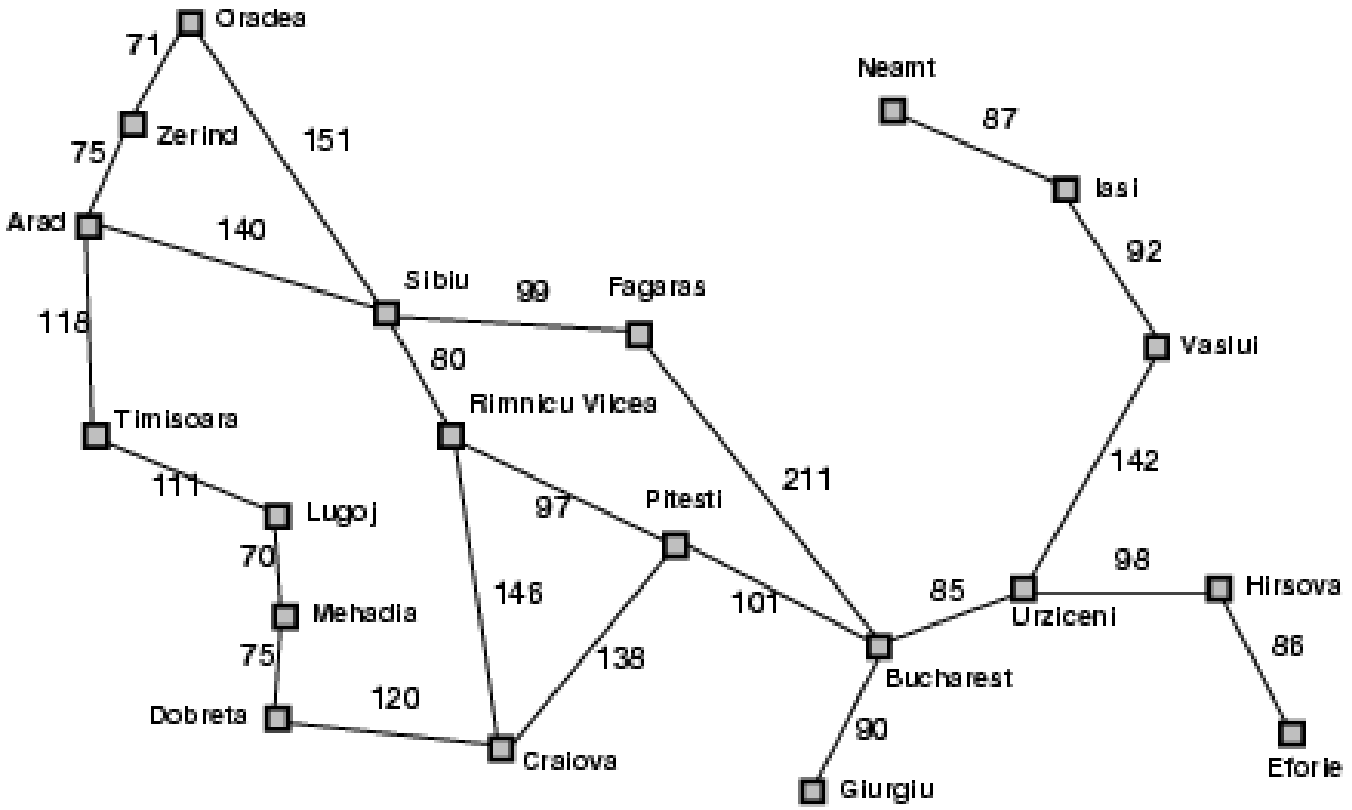
- **Idea**: give the algorithm "hints" about the desirability of different states
  - Use an *evaluation function* to rank nodes and select the most promising one for expansion
- Greedy best-first search
- A* search

Heuristic function

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



Heuristic for the Romania problem

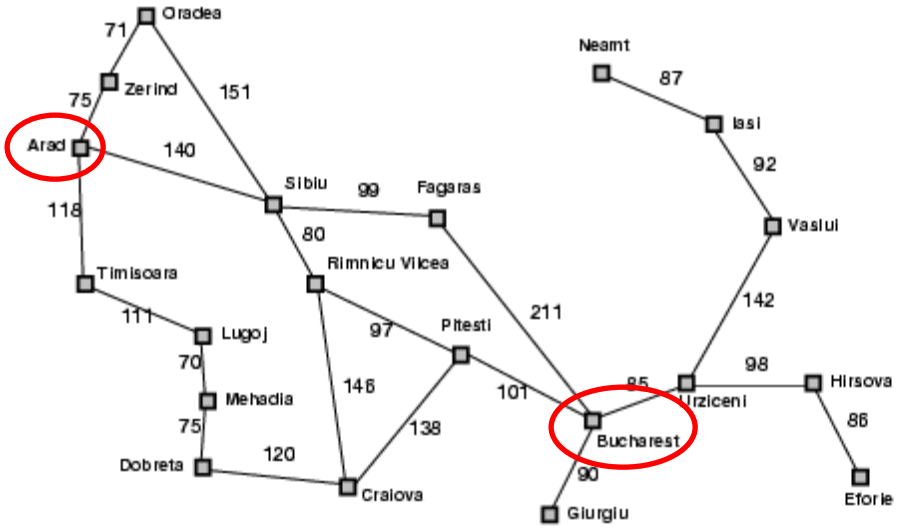


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

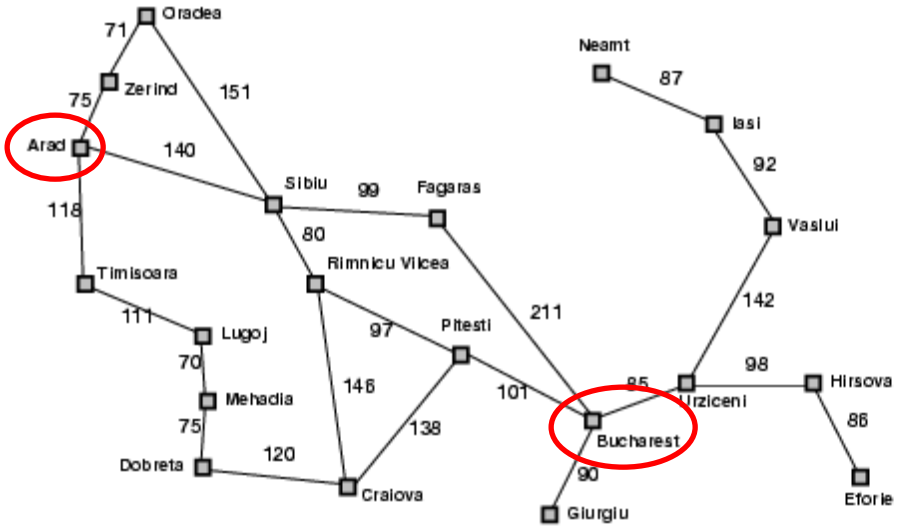
- Expand the node that has the lowest value of the heuristic function $h(n)$

Greedy best-first search example



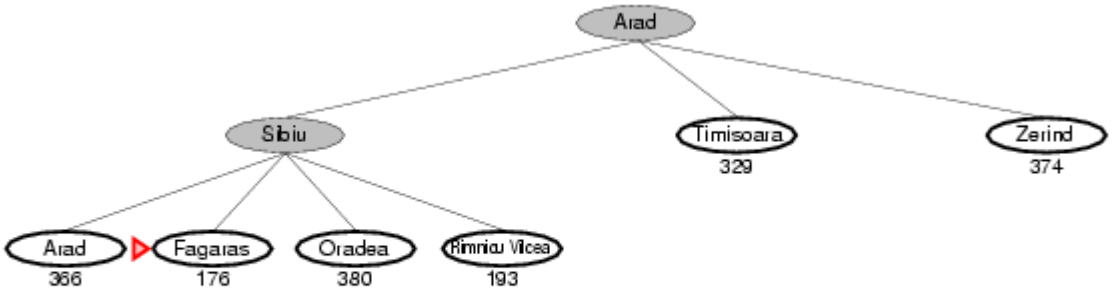
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

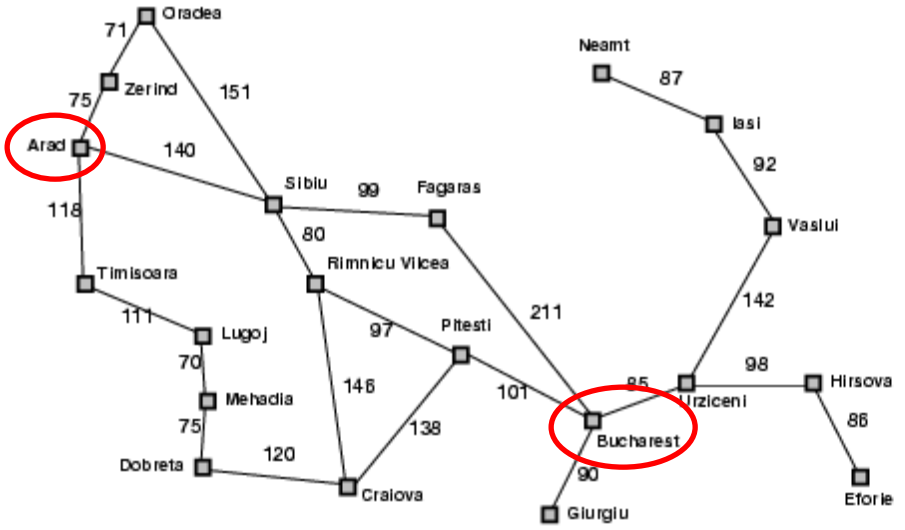
Greedy best-first search example



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

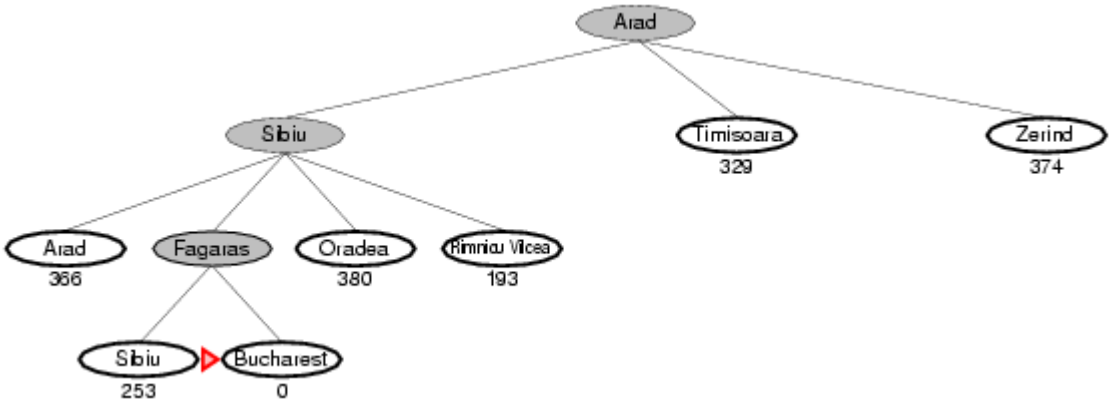
Greedy best-first search example

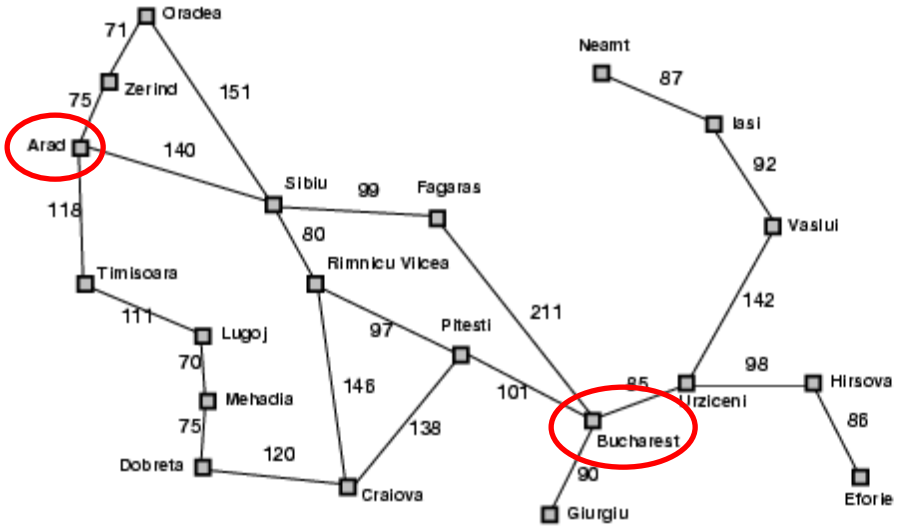




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example



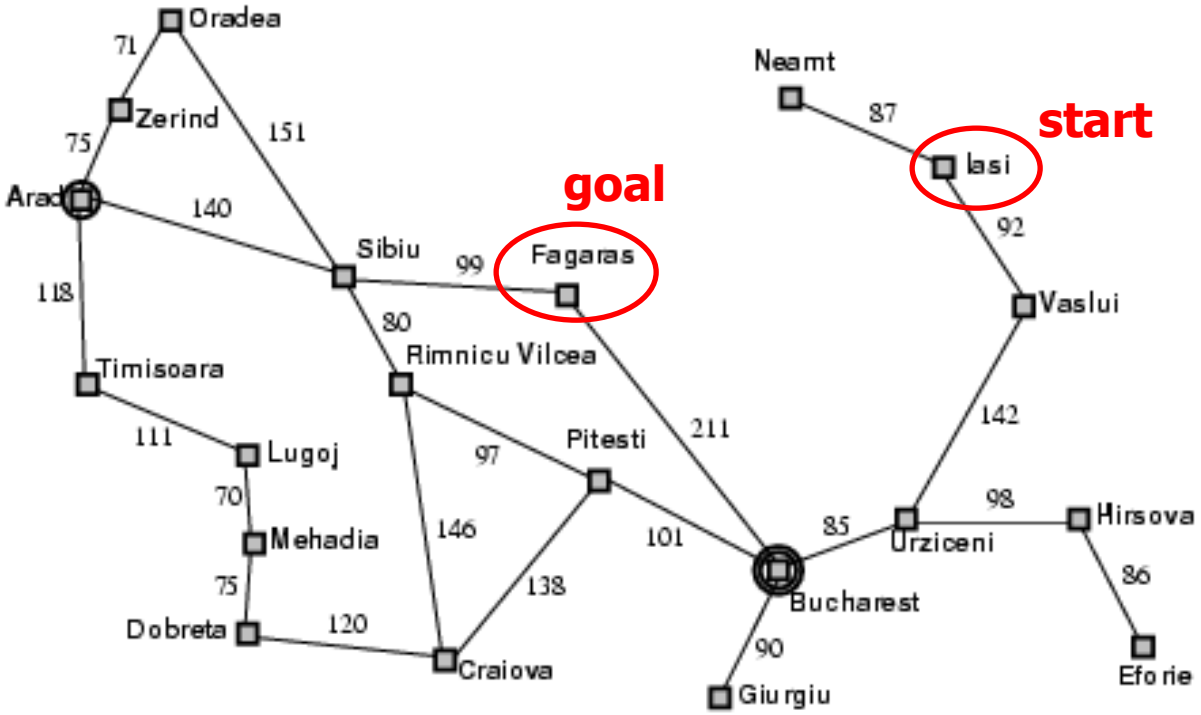


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops



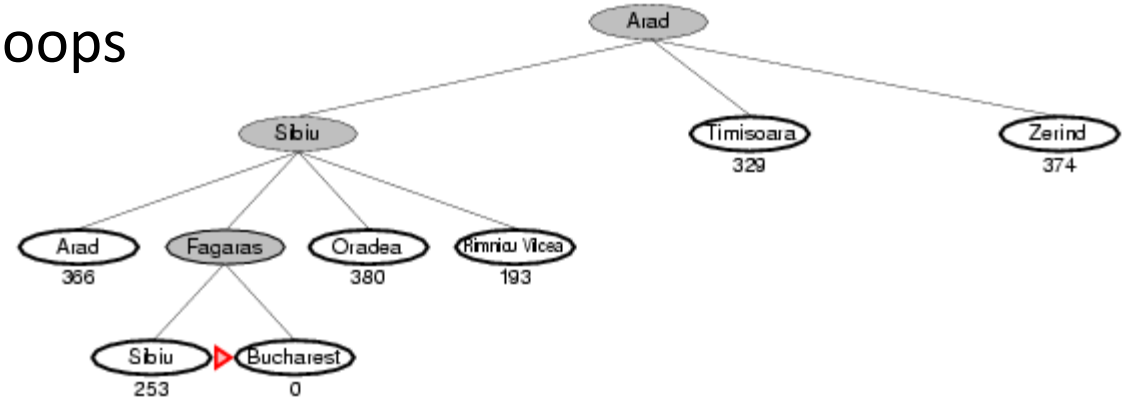
Properties of greedy best-first search

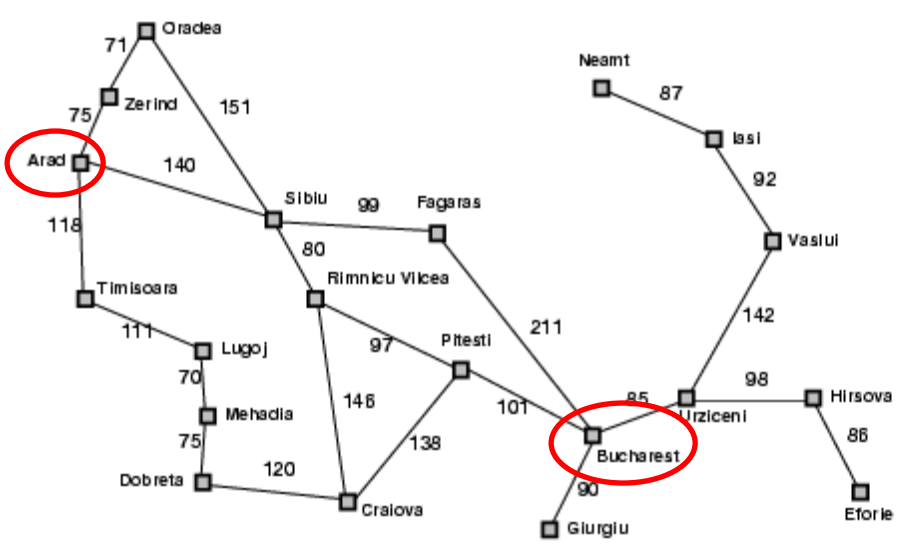
- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No





Straight–line distance to Bucharest

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

Properties of greedy best-first search

- **Complete?**

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

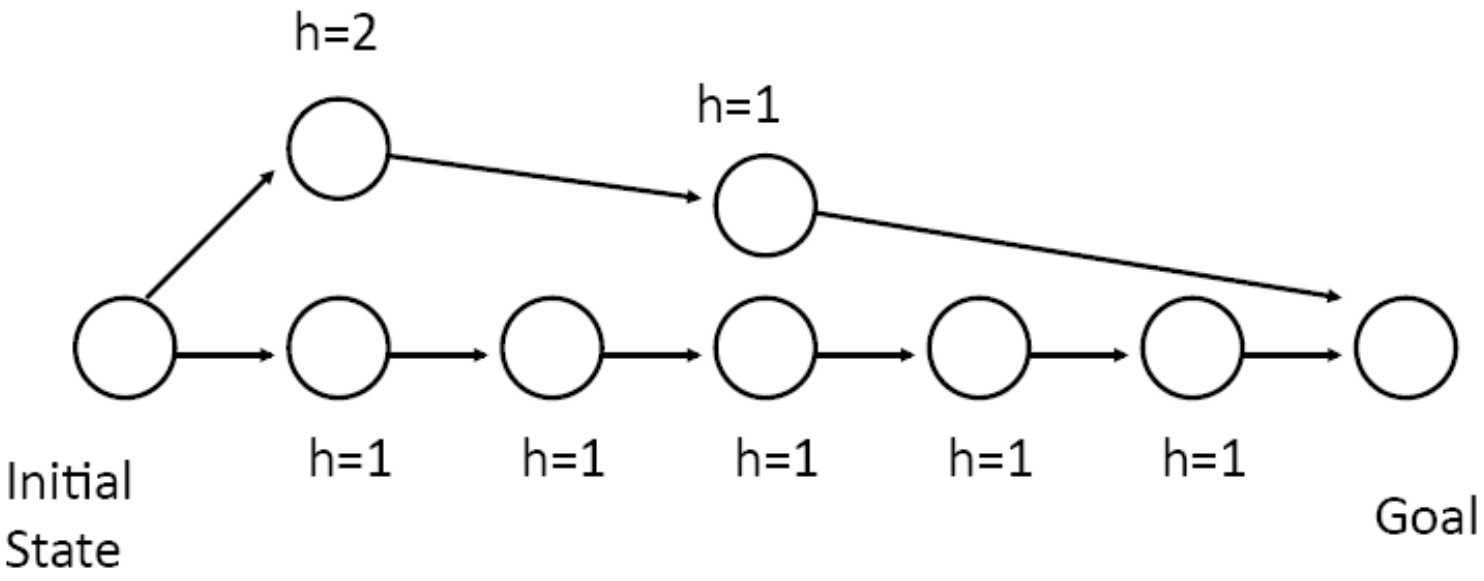
  No
- **Time?**

  Worst case: $O(b^m)$

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

  Worst case: $O(b^m)$

How can we fix the greedy problem?



A* search

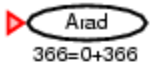
- Idea: avoid expanding paths that are already expensive
- The evaluation function $f(n)$ is the estimated total cost of the path through node n to the goal:

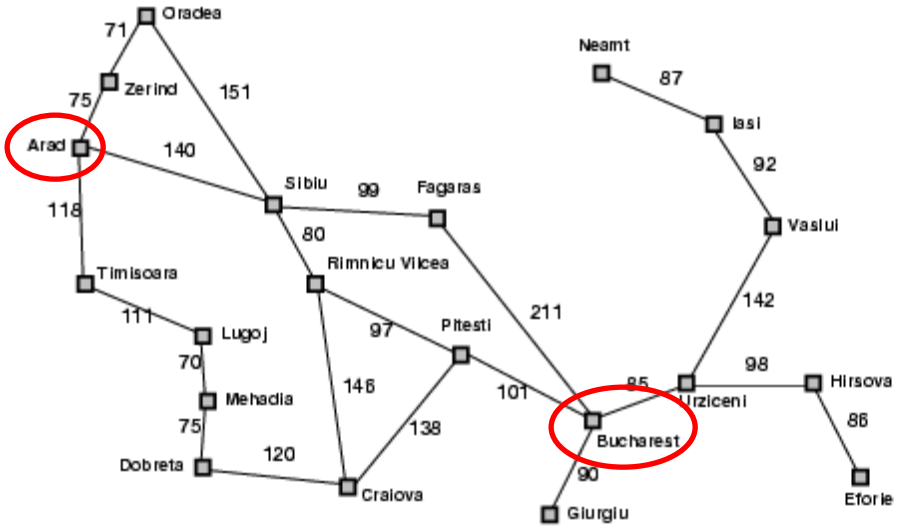
$$f(n) = g(n) + h(n)$$

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

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

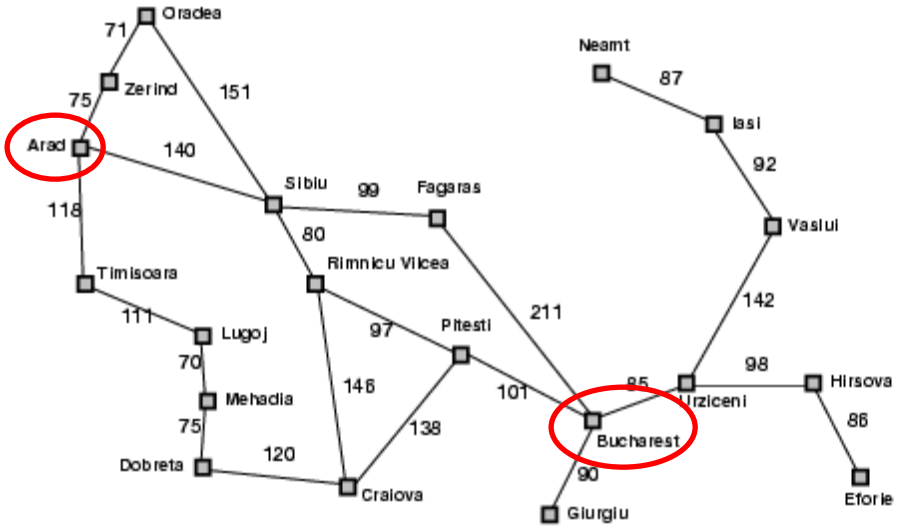
A* search example





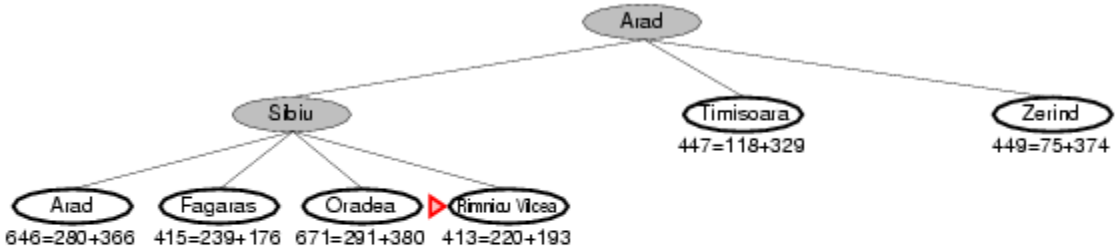
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

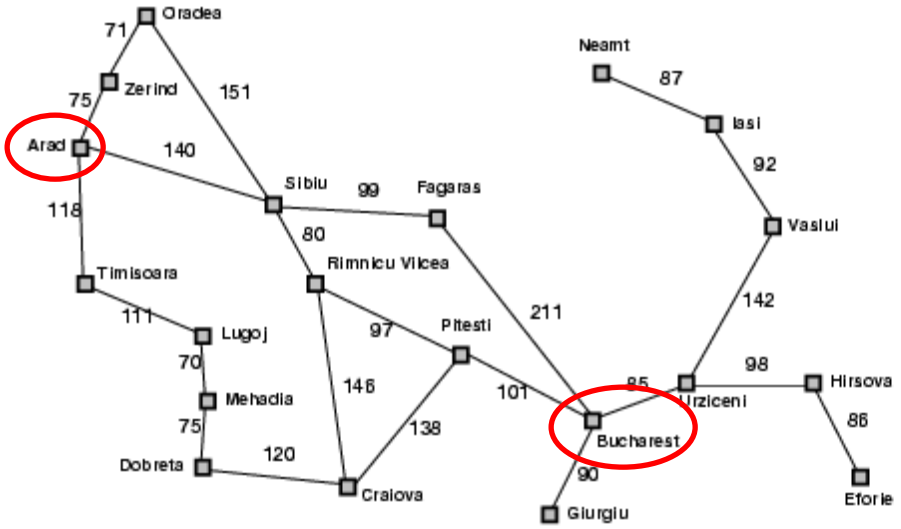
A* search example



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

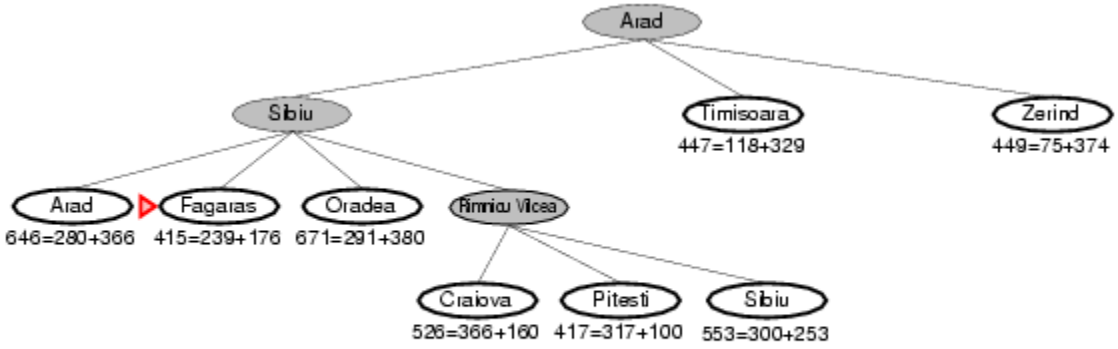
A* search example

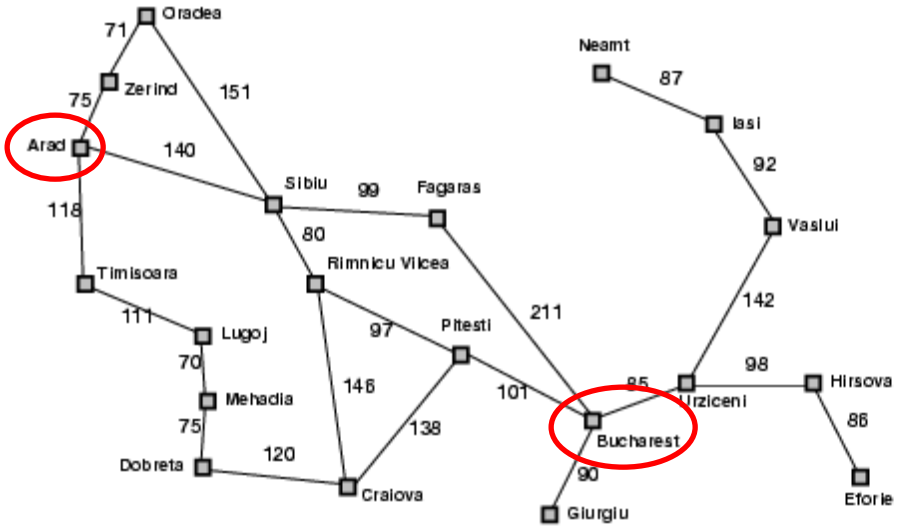




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

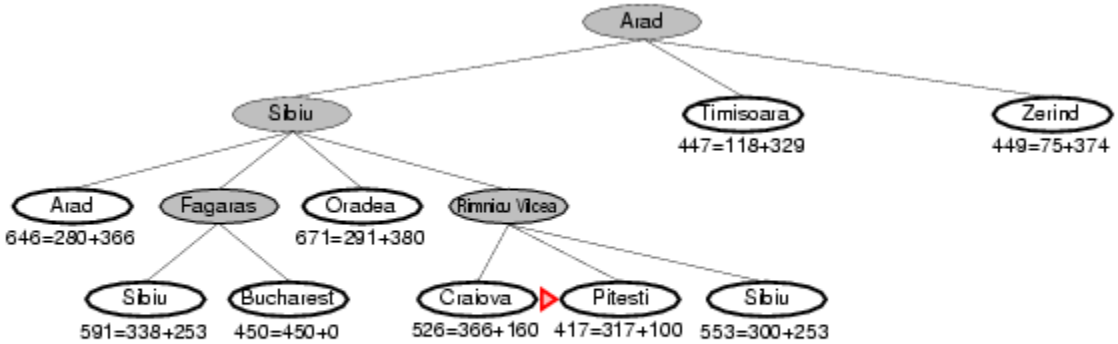
A* search example

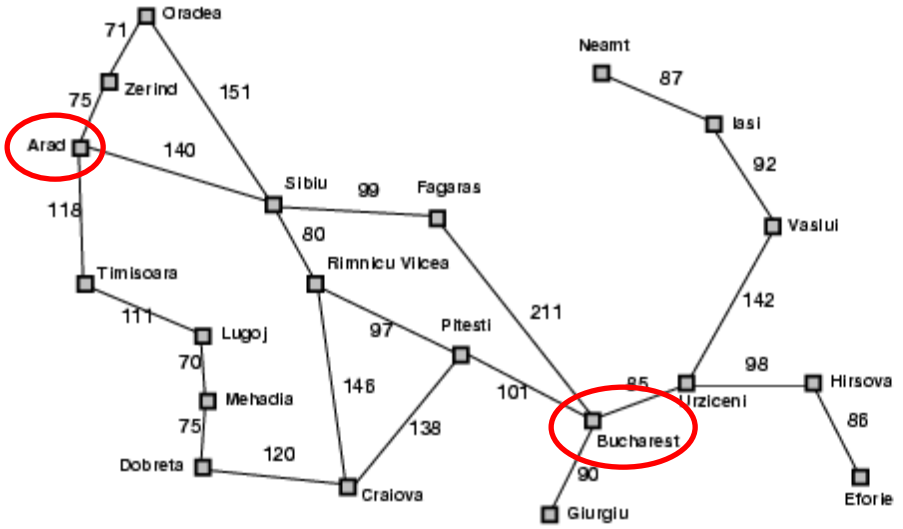




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

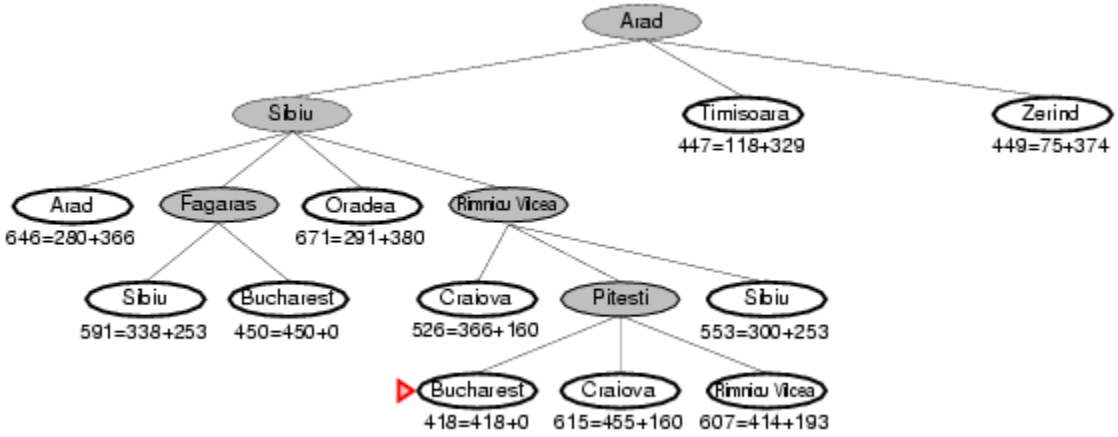
A* search example

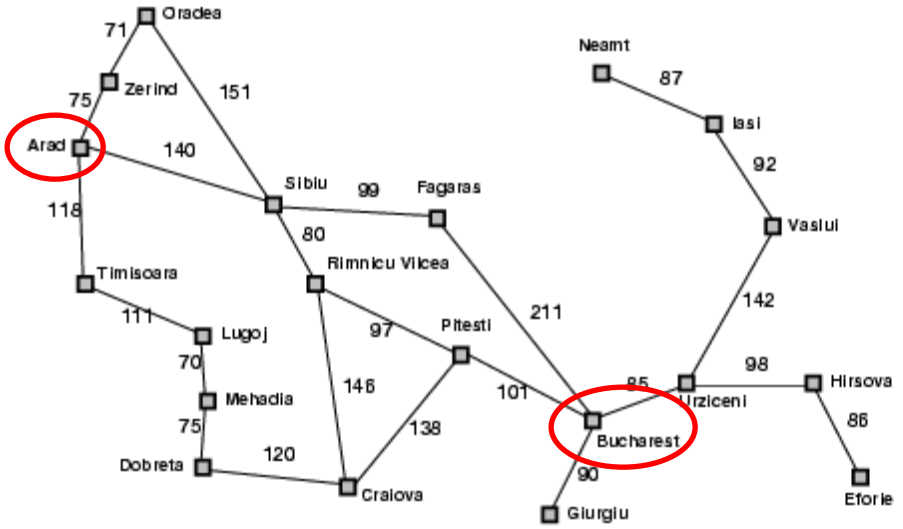




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example



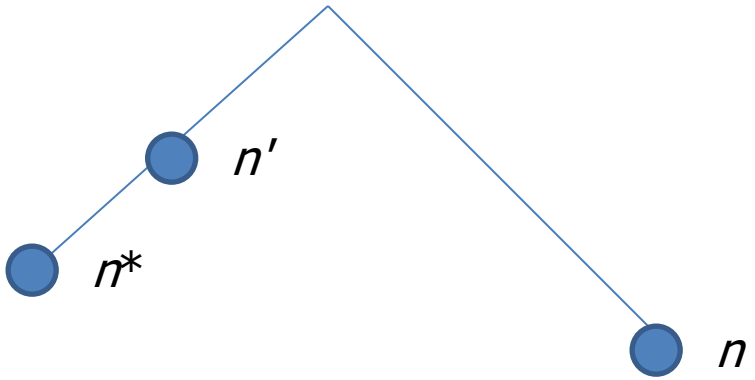


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Admissible heuristics

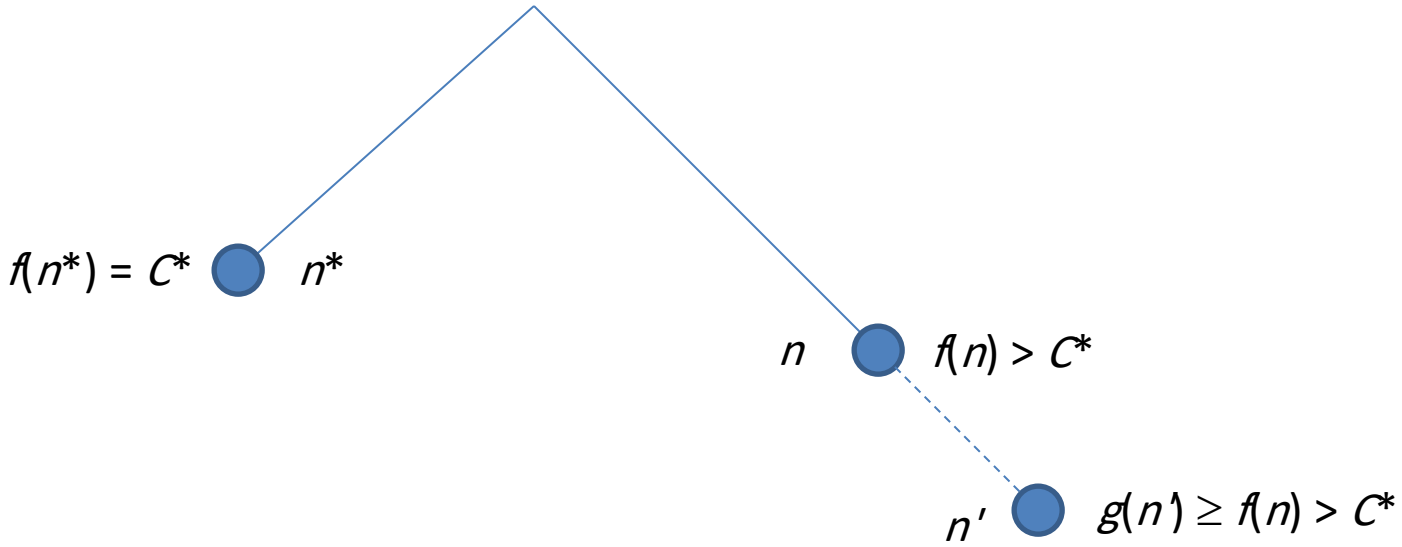
- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: straight line distance never overestimates the actual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

Optimality of A*



- Proof by contradiction
  - Let n^* be an optimal goal state, i.e., $f(n^*) = C^*$
  - Suppose a solution node n with $f(n) > C^*$ is about to be expanded
  - Let n' be a node in the fringe that is on the path to n^*
  - We have $f(n') = g(n') + h(n') \leq C^*$
  - But then, n' should be expanded before n – a contradiction

Optimality of A*



- In other words:
  - Suppose A* terminates its search at *n**
  - It has found a path whose *actual cost* $f(n^*) = g(n^*)$ is lower than the *estimated cost* $f(n)$ of any path going through any fringe node
  - Since $f(n)$ is an *optimistic* estimate, there is no way *n* can have a successor goal state *n'* with $g(n') < C^*$

Optimality of A*

- A* is optimally efficient – no other tree-based algorithm that uses the same heuristic can expand fewer nodes and still be guaranteed to find the optimal solution
  - Any algorithm that does not expand all nodes with $f(n) < C^*$ risks missing the optimal solution

Properties of A*

- **Complete?**

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

  Yes
- **Time?**

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

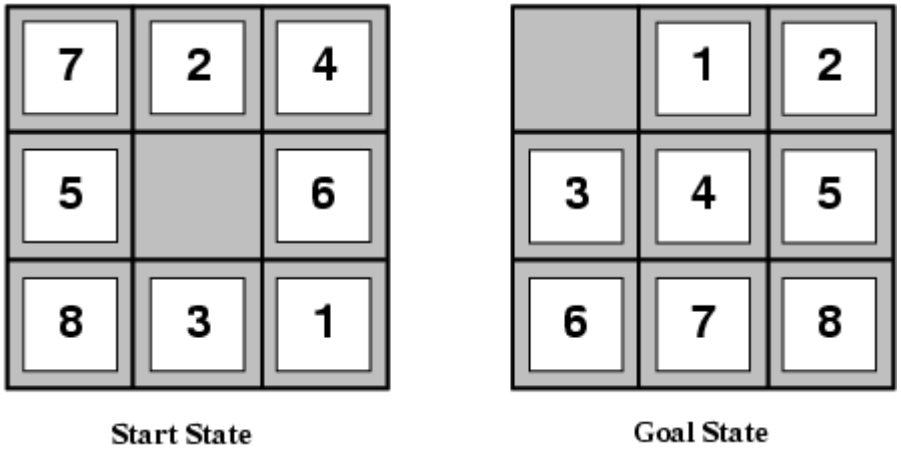
  Exponential

Designing heuristic functions

- Heuristics for the 8-puzzle

  $h_1(n)$ = number of misplaced tiles

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



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

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

- Are h_1 and h_2 admissible?

Heuristics from relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Dominance

- If h_1 and h_2 are both admissible heuristics and $h_2(n) \geq h_1(n)$ for all n, (both admissible) then h_2 dominates h_1
- Which one is better for search?
  - A* search expands every node with $f(n) < C^*$ or $h(n) < C^* - g(n)$
  - Therefore, A* search with h_1 will expand more nodes

Dominance

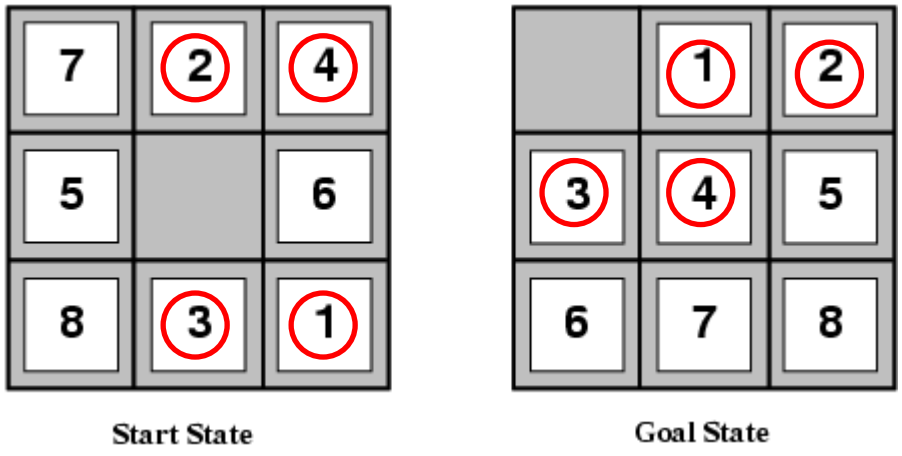
- Typical search costs for the 8-puzzle (average number of nodes expanded for different solution depths):

- d=12 IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes

- d=24 IDS ≈ 54,000,000,000 nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Heuristics from subproblems

- Let $h_3(n)$ be the cost of getting a subset of tiles (say, 1,2,3,4) into their correct positions
- Can precompute and save the exact solution cost for every possible subproblem instance – *pattern database*



Combining heuristics

- Suppose we have a collection of admissible heuristics $h_1(n)$, $h_2(n)$, …, $h_m(n)$, but none of them dominates the others
- How can we combine them?

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

Memory-bounded search

- The memory usage of A* can still be exorbitant
- How to make A* more memory-efficient while maintaining completeness and optimality?

- Iterative deepening A* search
- Recursive best-first search, SMA*
  - Forget some subtrees but remember the best f-value in these subtrees and regenerate them later if necessary

- Problems: memory-bounded strategies can be complicated to implement, suffer from “thrashing”

Comparison of search strategies

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