Artificial Intelligence (AI)

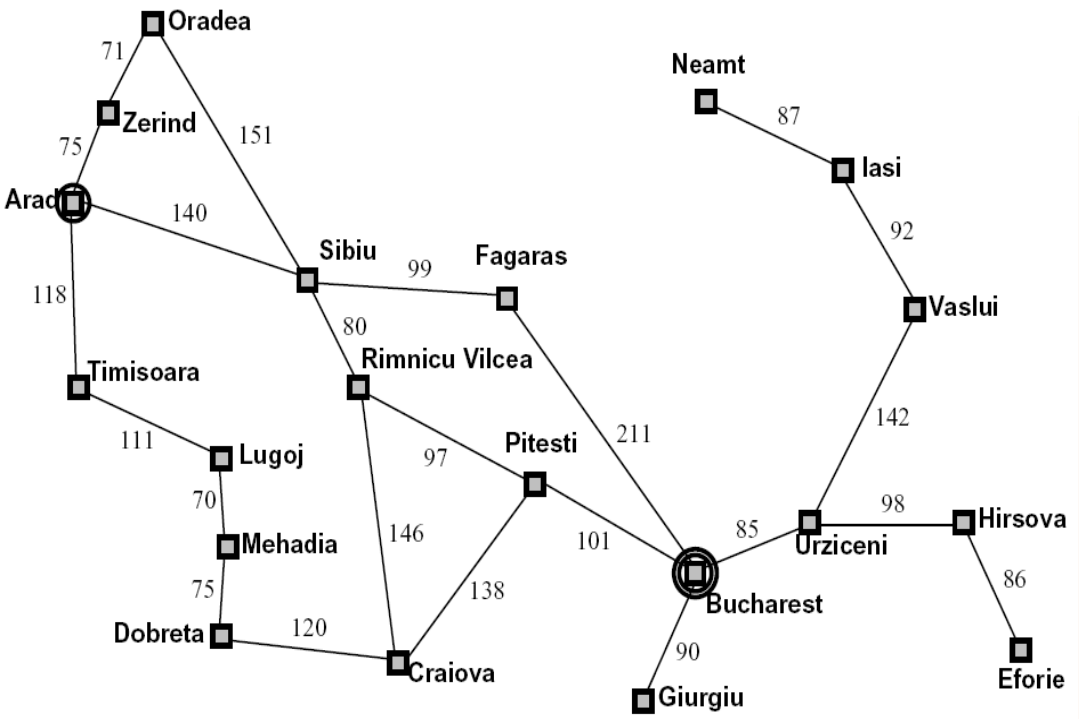
Lecture 2: Introduction

CPS 480/ CPS 580 Artificial Intelligence
Ju Shen, Spring 2018

Search Definition

State space search is a process used in the field of computer science, including artificial intelligence (AI), in which successive configurations or states of an instance are considered, with the intention of finding a goal state with a desired property

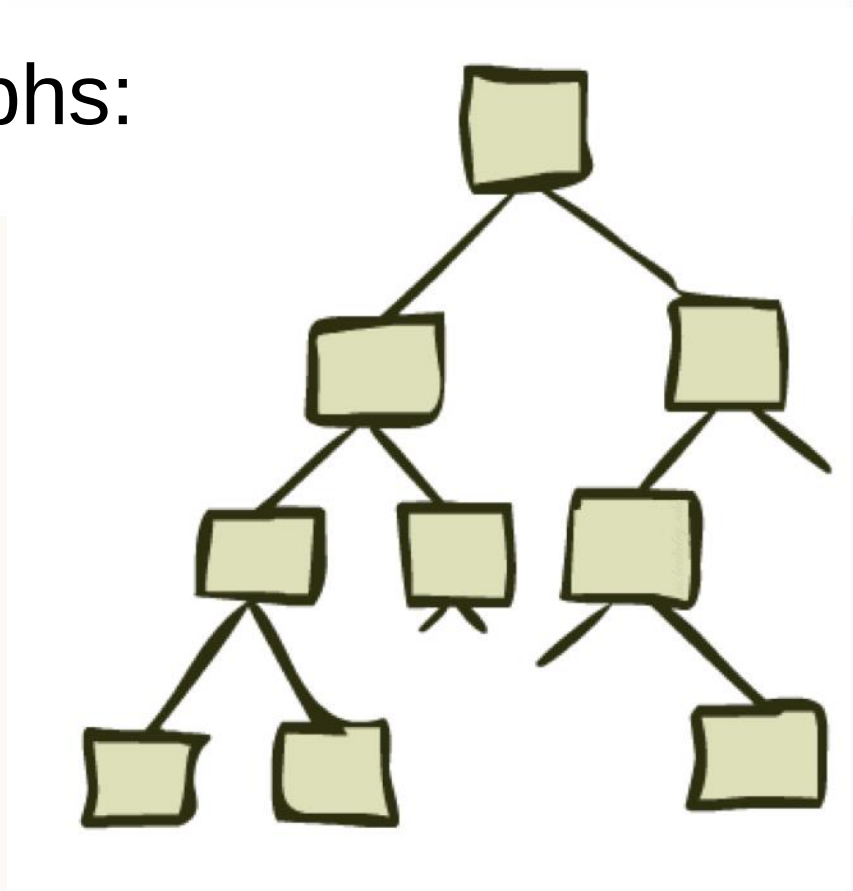
Search



- State space:
  - **Cities**
- Successor function:
  - **Roads: Go to adjacent city with cost = distance**
- Start state:
  - **Arad**
- Goal test:
  - **Is state == Bucharest?**
- Solution?

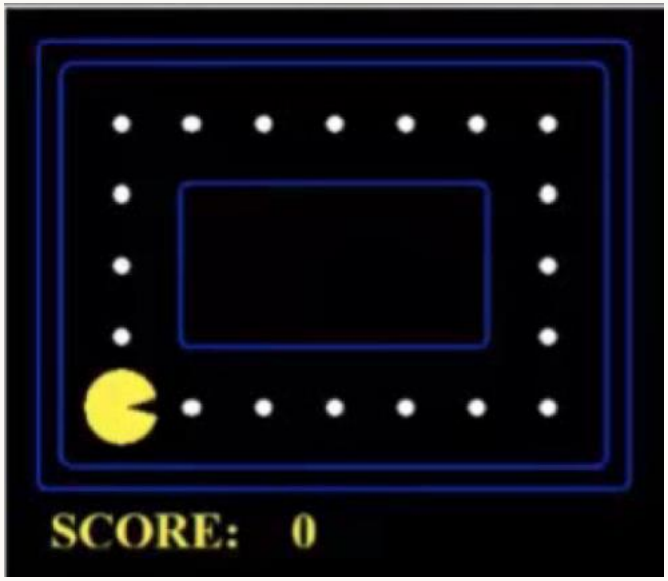
Search

- State Space Graphs:
  - Search Trees



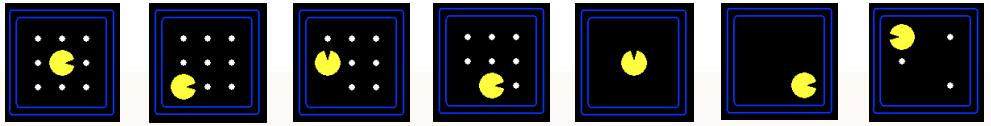
Search

Problem: eat all dots

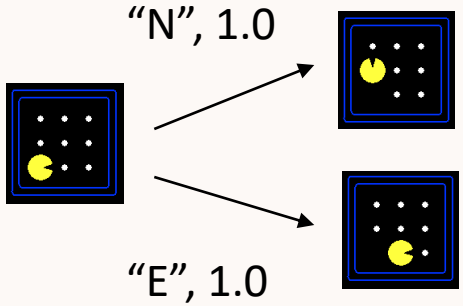


Search

- A search problem consists of:
  - A state space



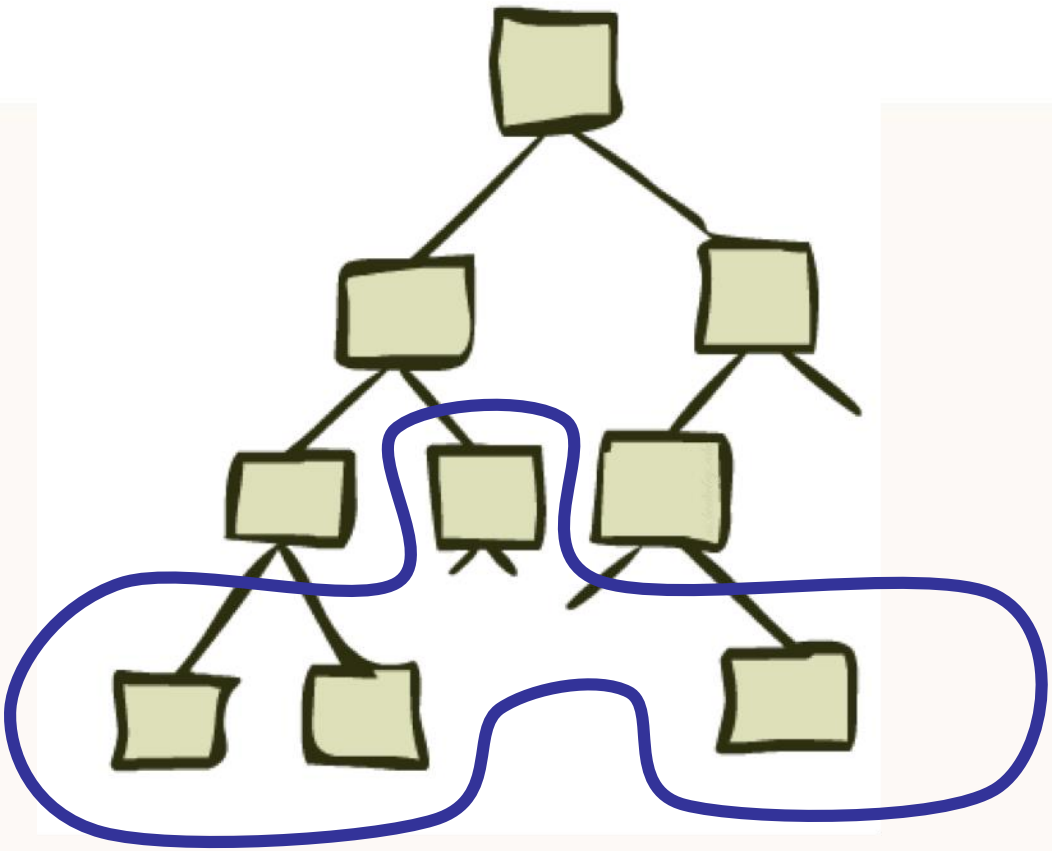
  - A successor function (with actions, costs)



  - A start state and a goal test

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

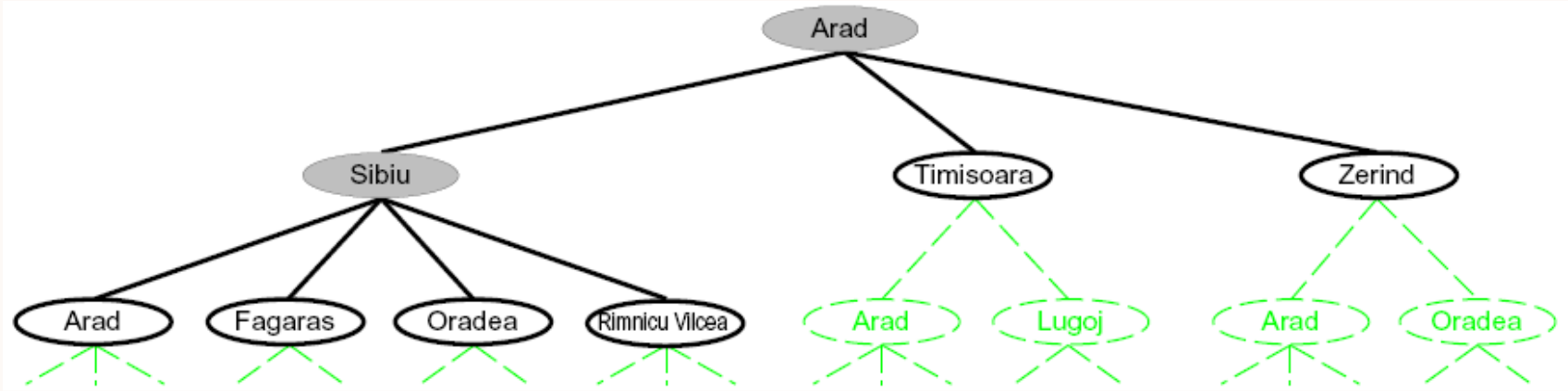
Tree Search



fringe

Tree Search

- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible



Tree Search

- ***Search: tree search and graph search***

- *Uninformed* search: very briefly (covered before in other pre-requisite courses – recommendation: review these techniques at home)

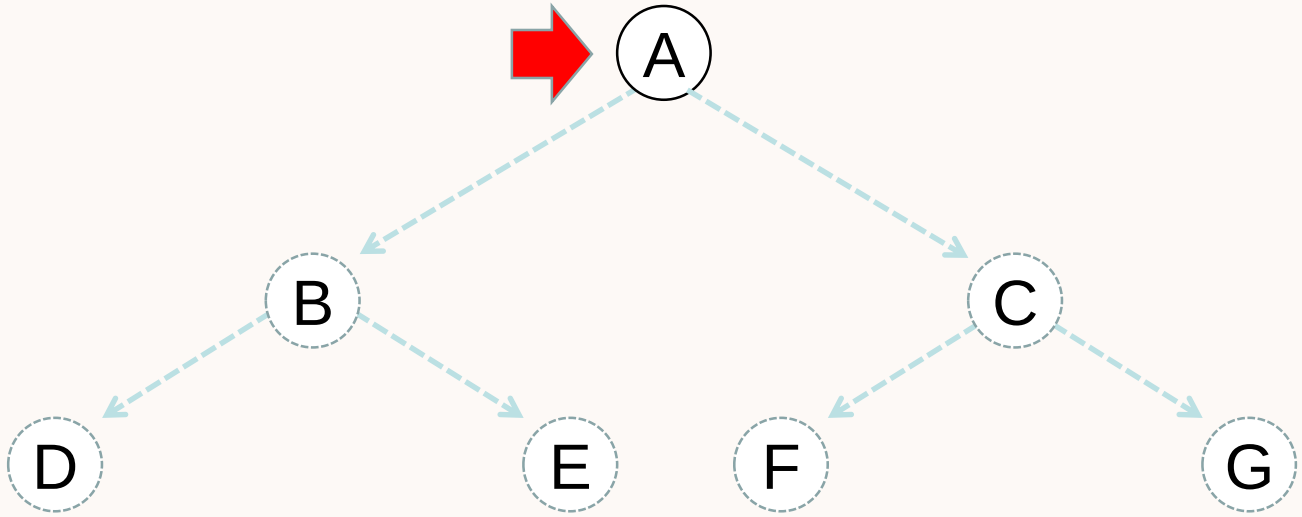
- *Informed* search ← focus of the lecture

Uninformed Search

- **Uninformed (blind) search 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**
  - **Bidirectional search**

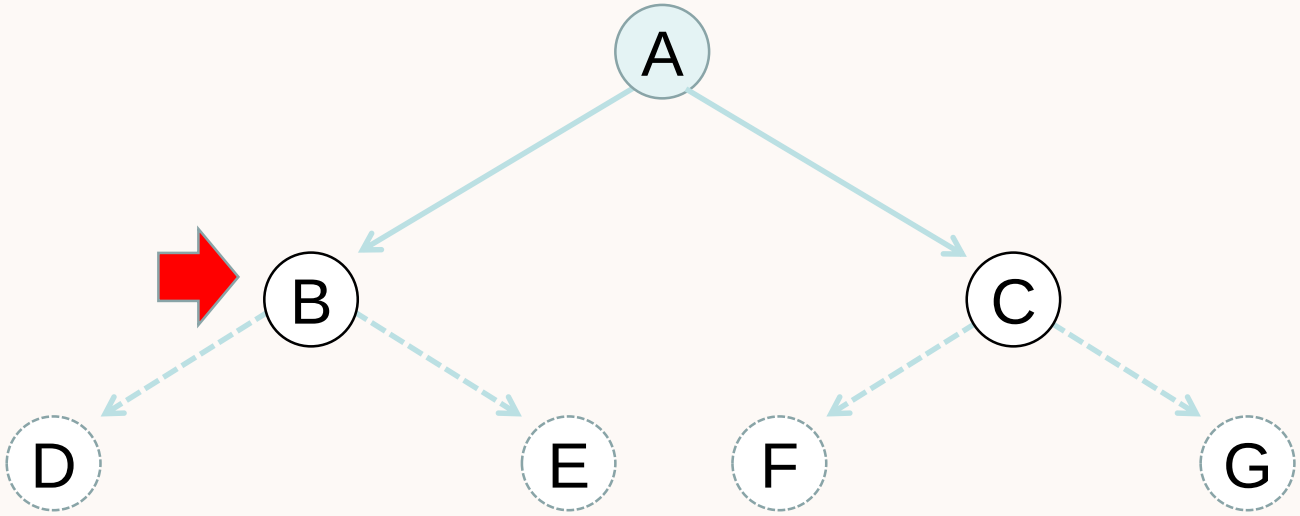
Breath First Search

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



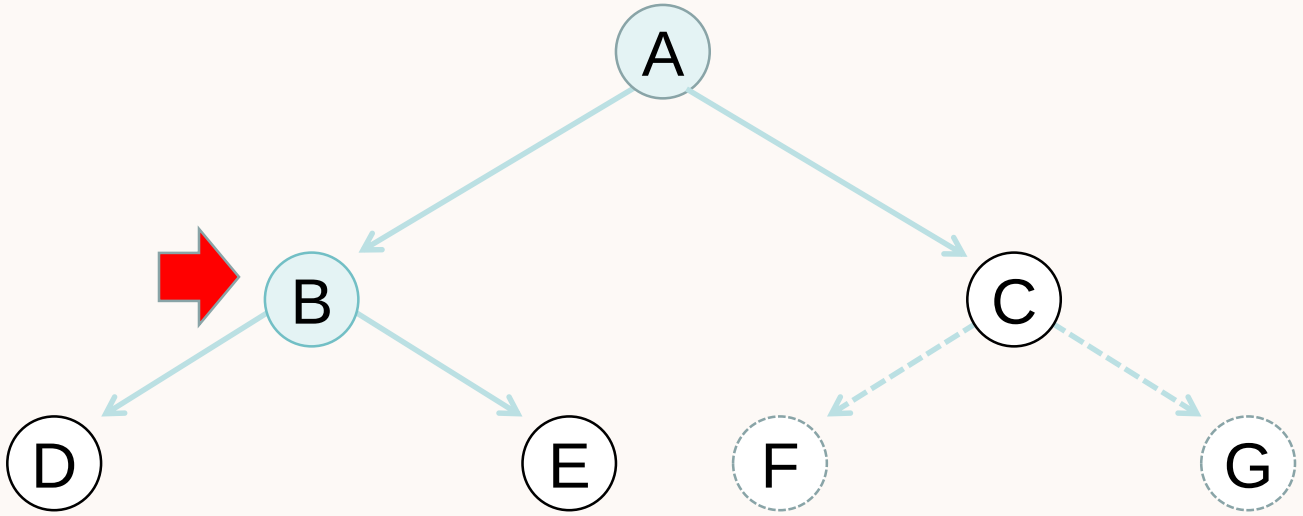
Breath First Search

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



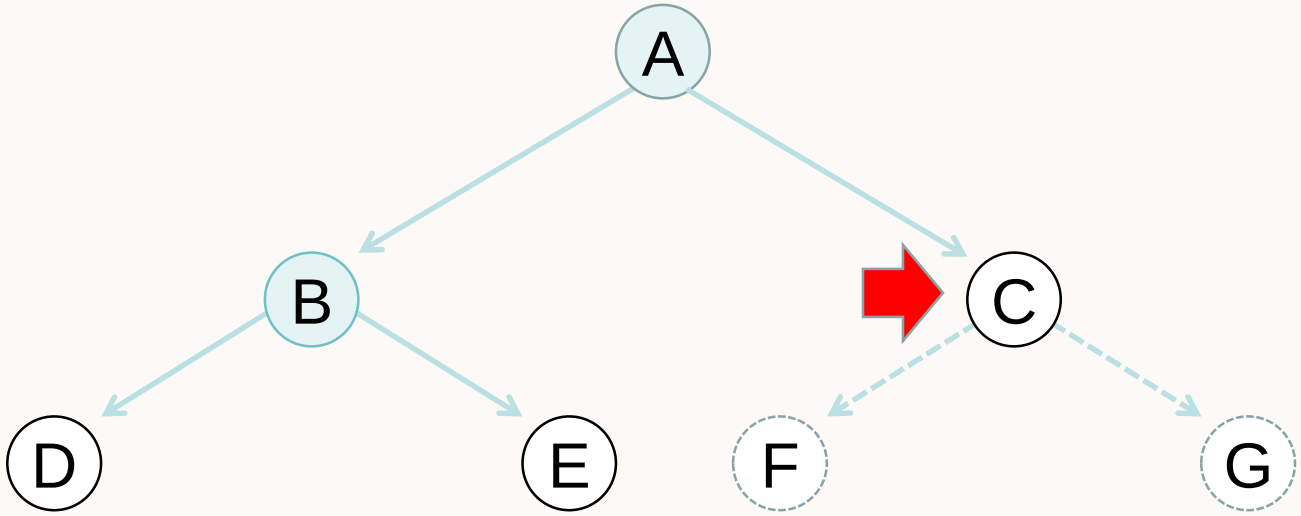
Breath First Search

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



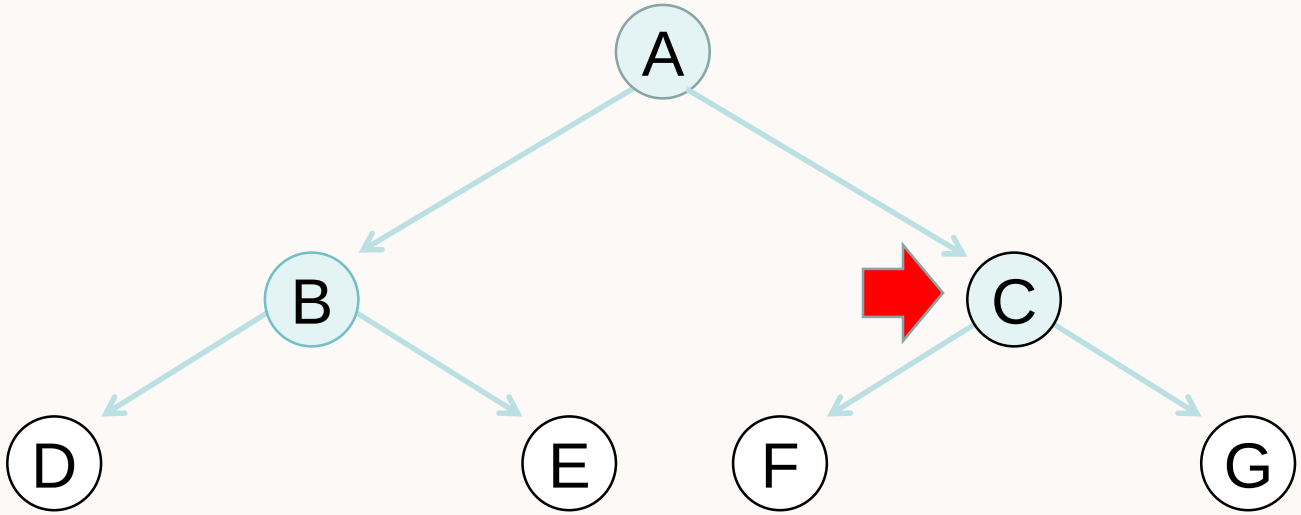
Breath First Search

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



Breath First Search

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



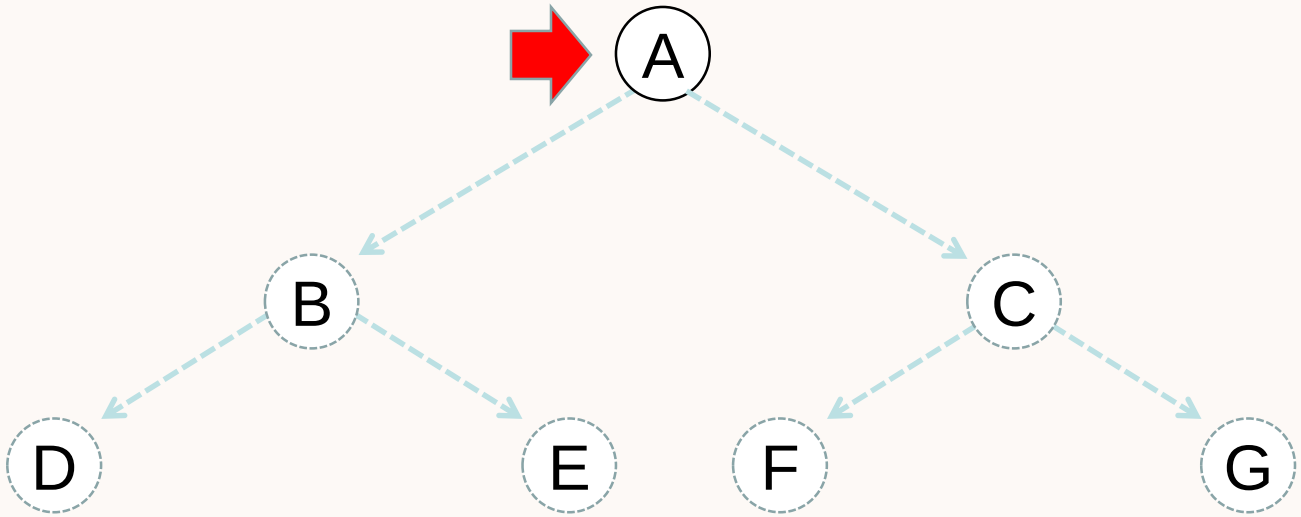
Properties of breadth-first search

- **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)

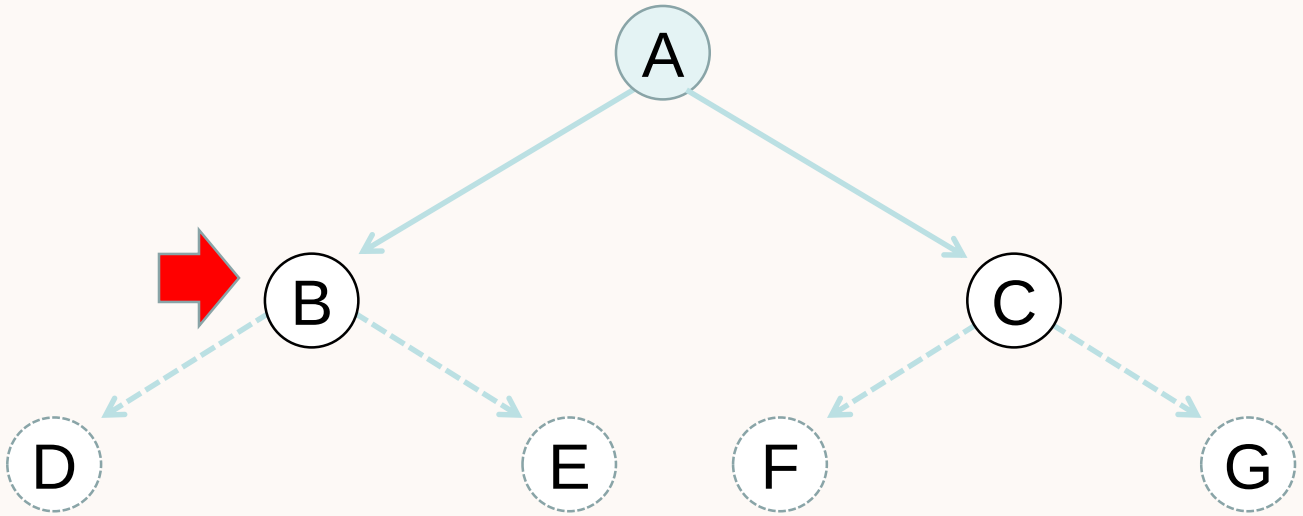
Depth First Search

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



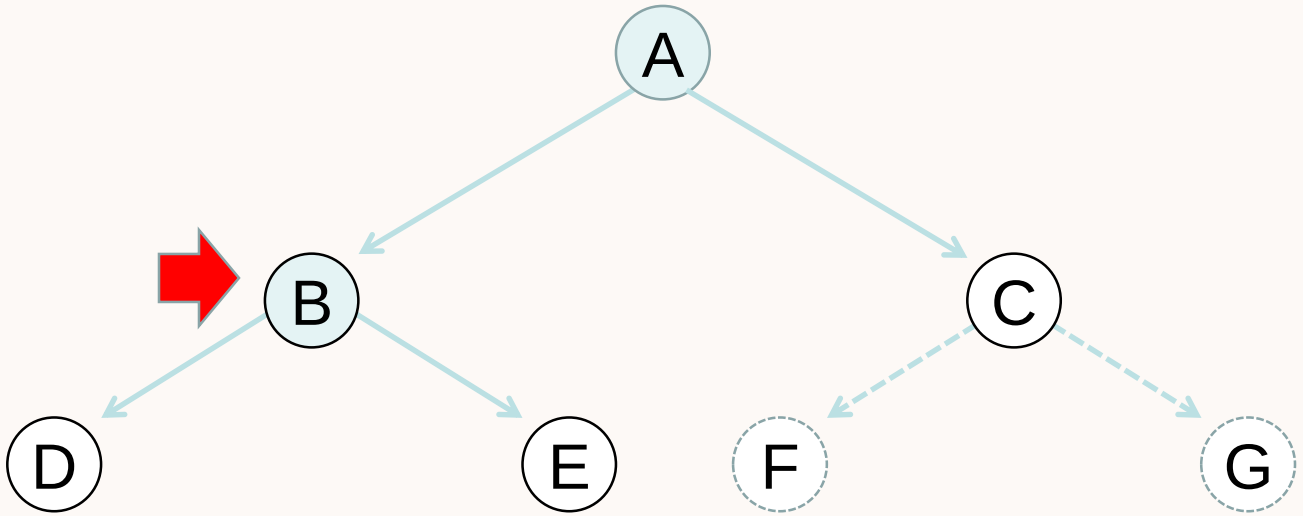
Depth First Search

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



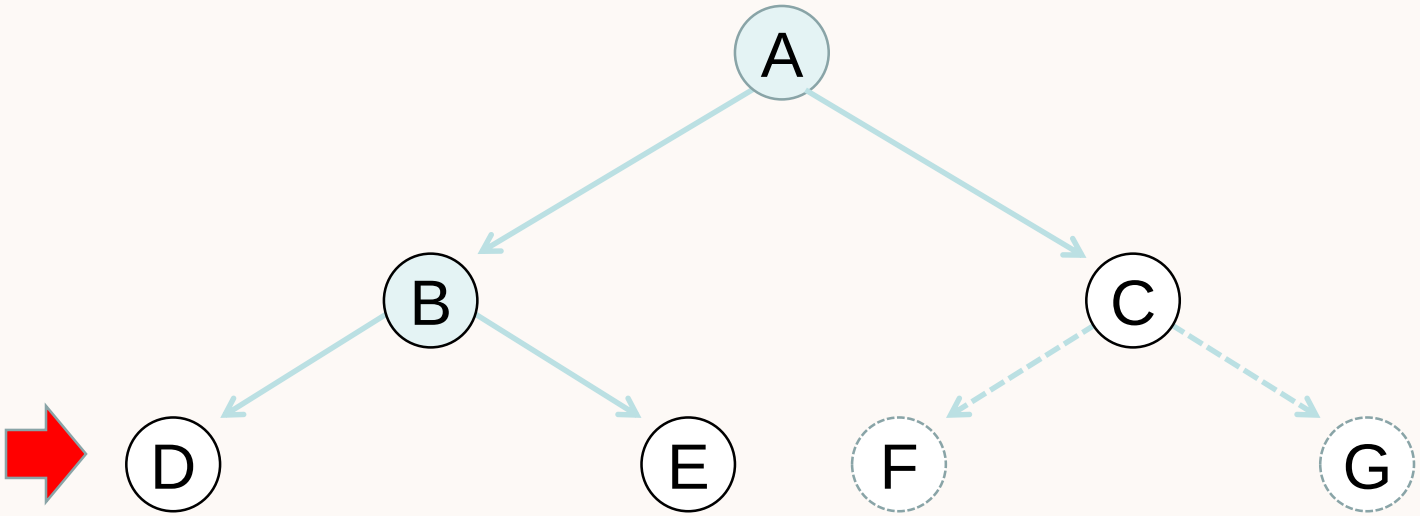
Depth First Search

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



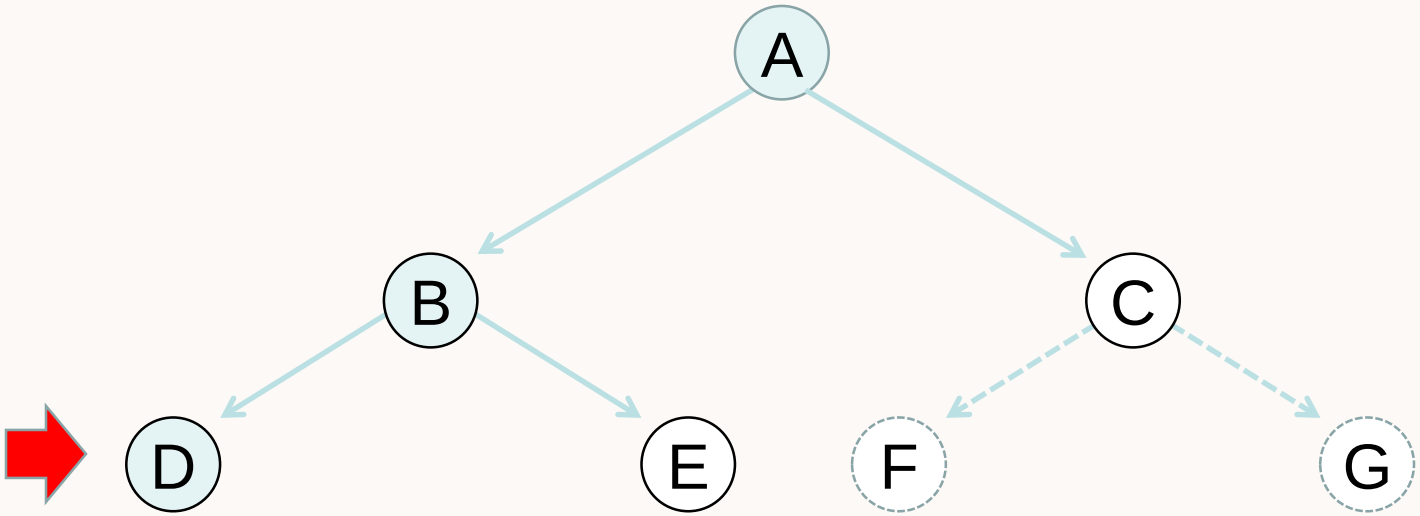
Depth First Search

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



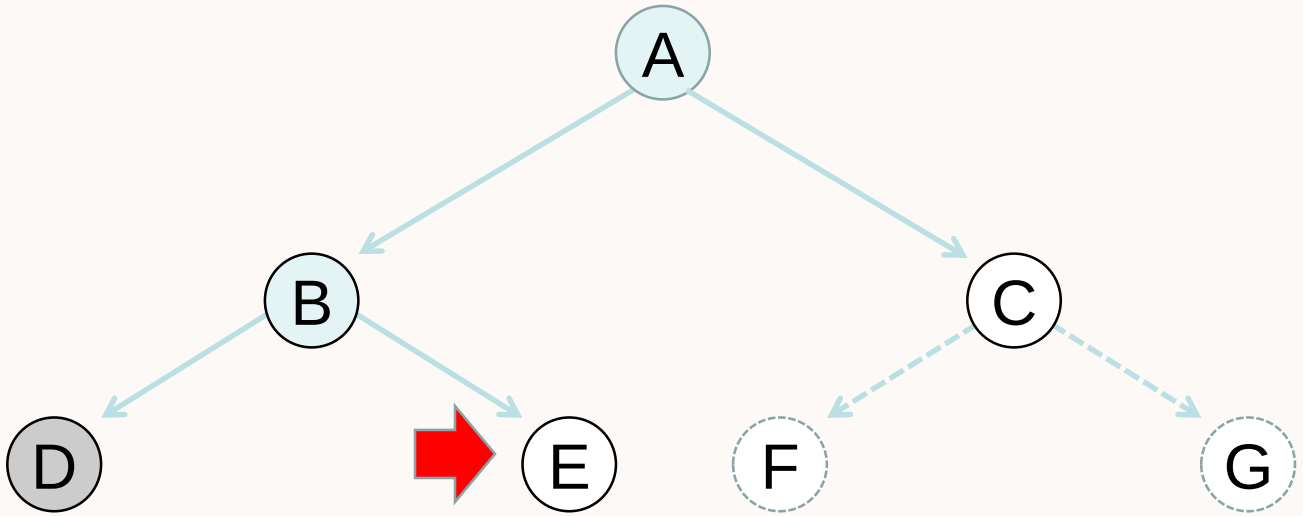
Depth First Search

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



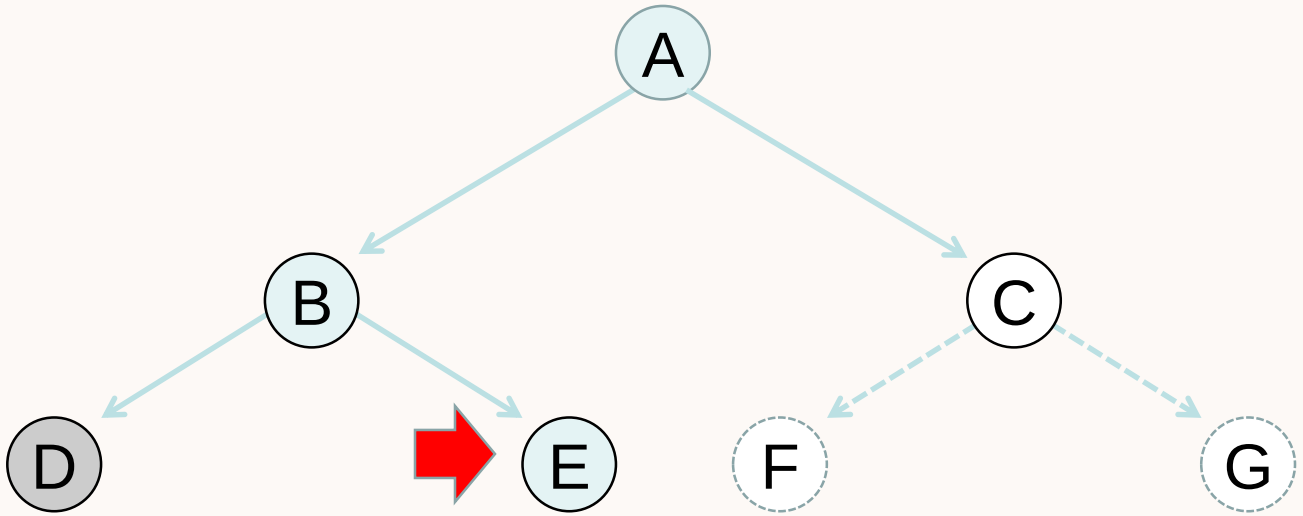
Depth First Search

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



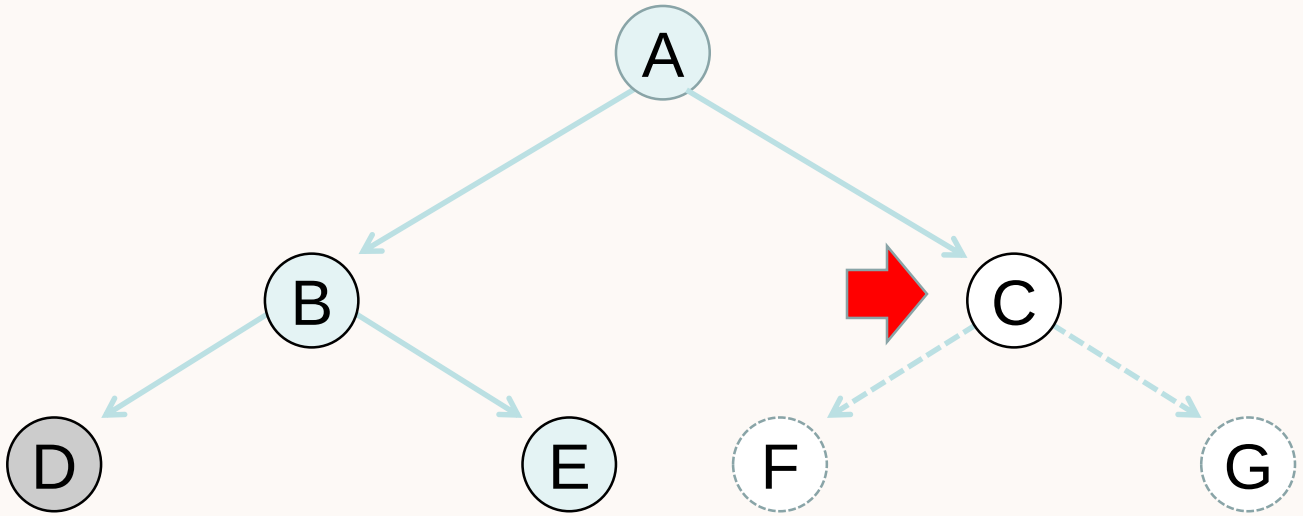
Depth First Search

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



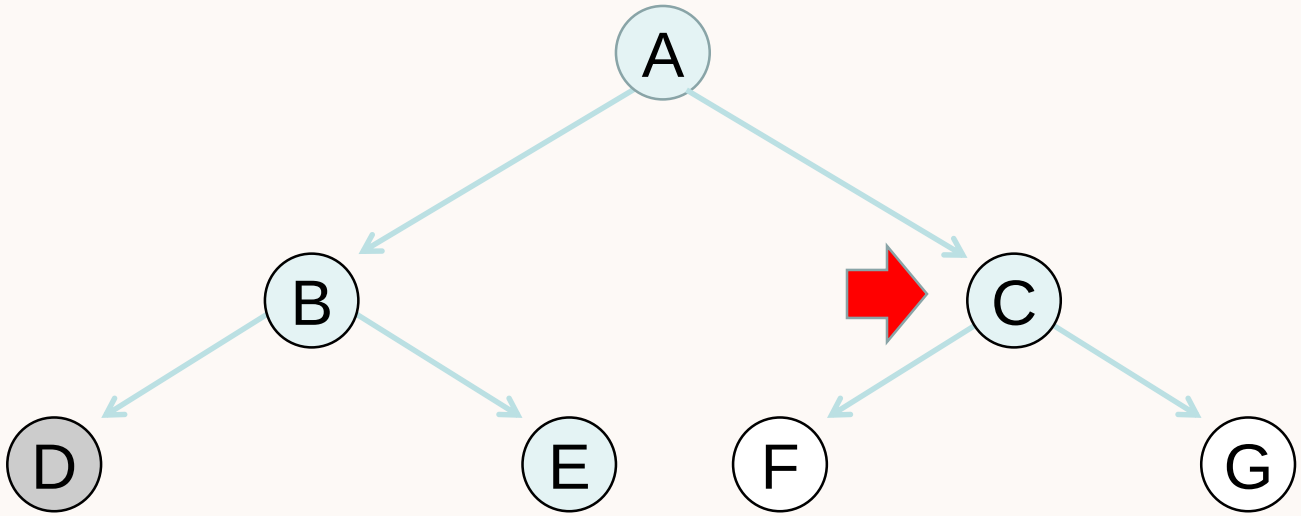
Depth First Search

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



Depth First Search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = FIFO 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!

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…

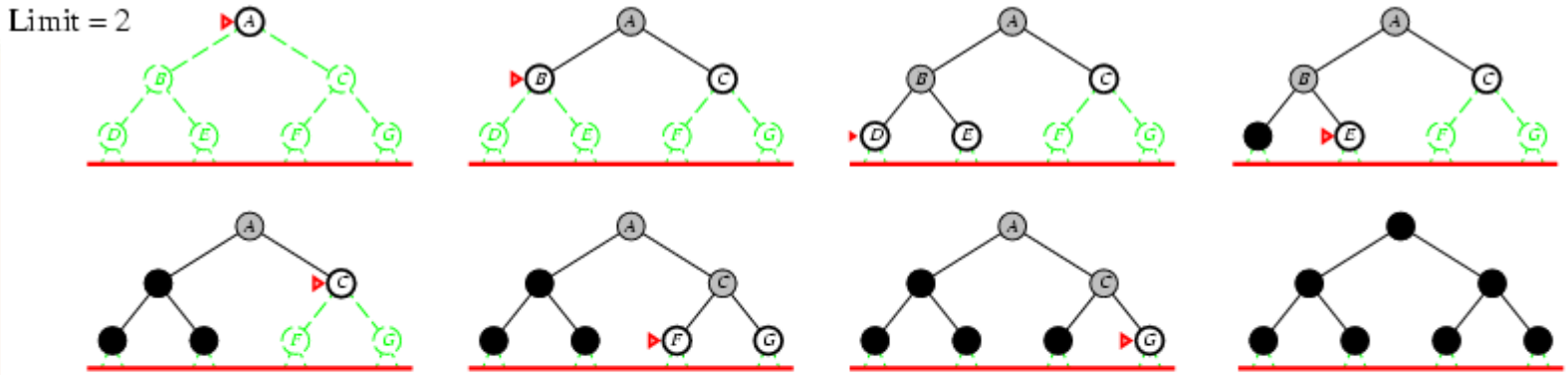
Iterative deepening search

Limit = 0

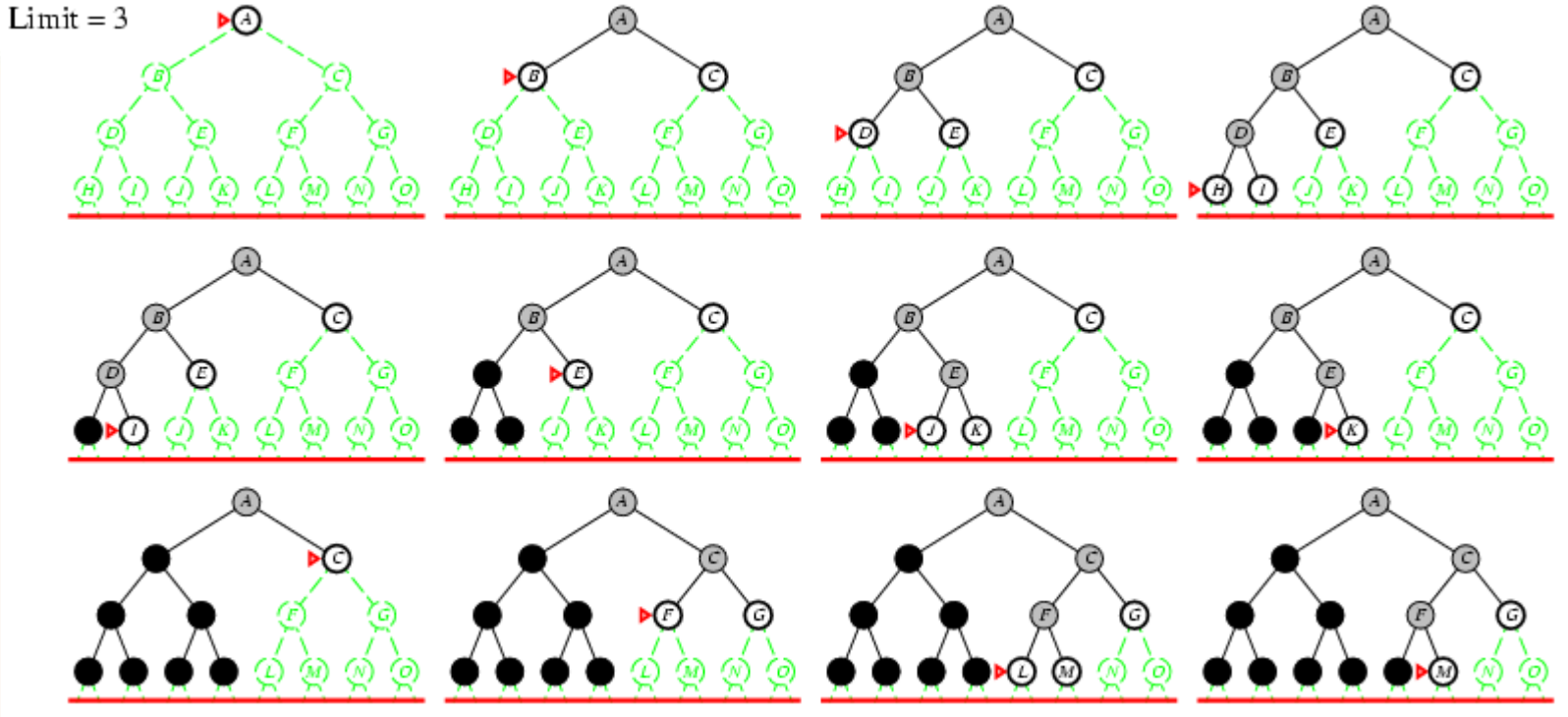
Iterative deepening search

Limit = 1

Iterative deepening search



Iterative deepening search



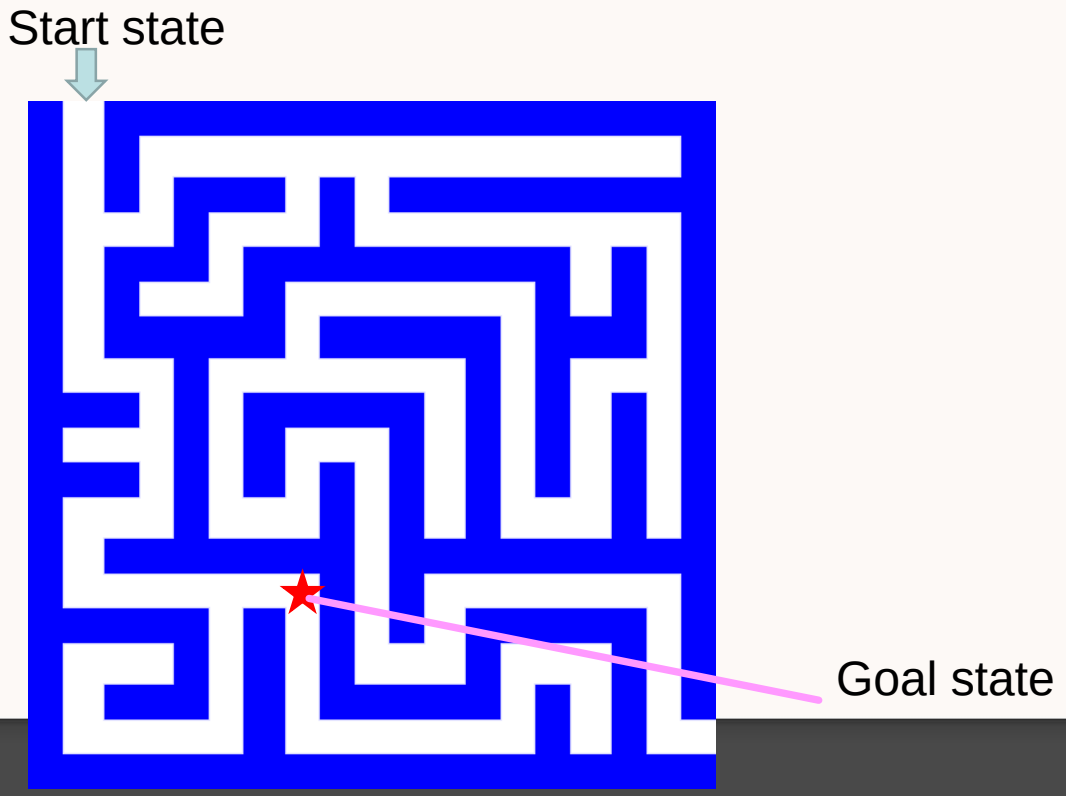
Informed search

- 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

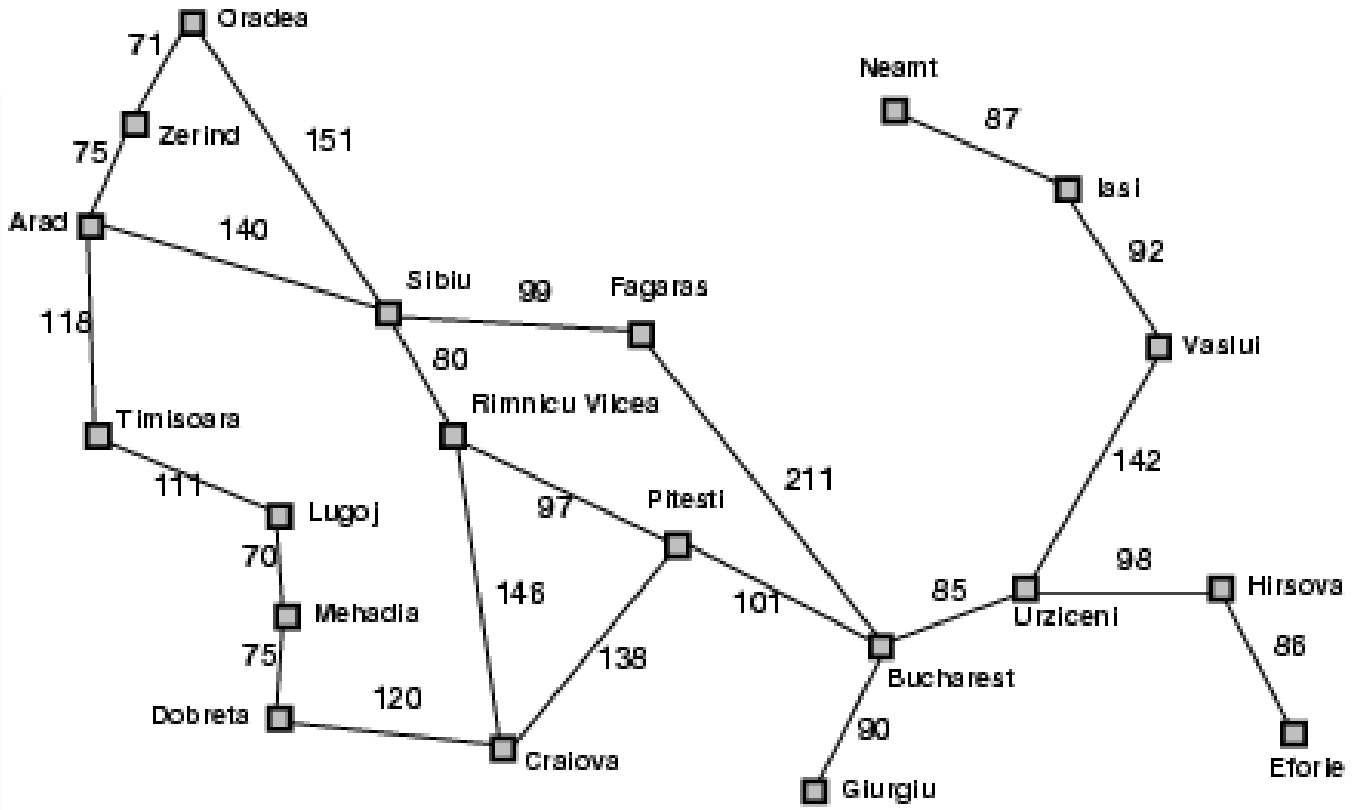
- Greedy best-first search
- A* search

Heuristic function

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



Heuristic Function



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

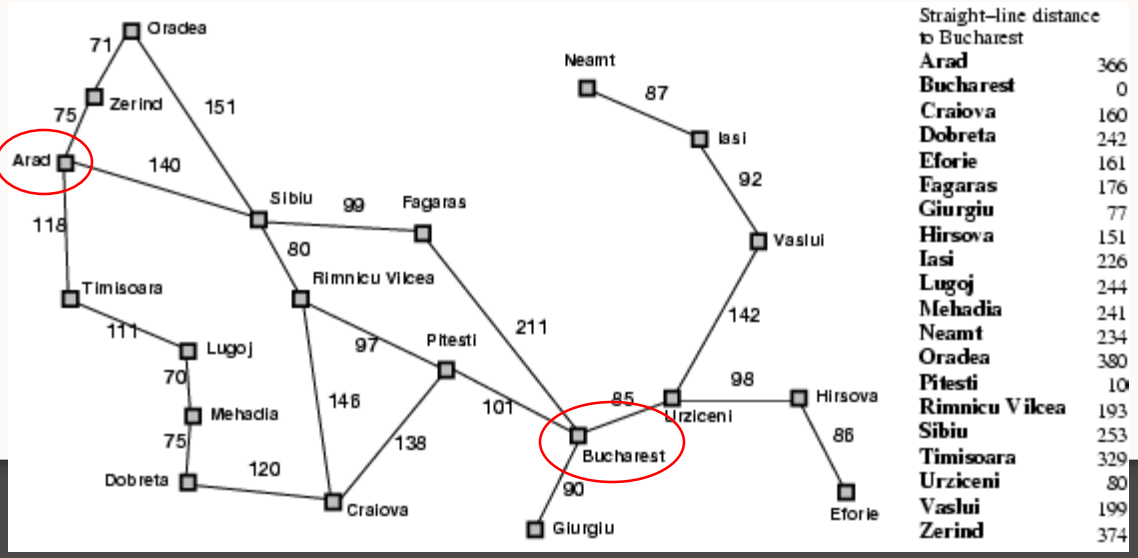
Greedy best-first search

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

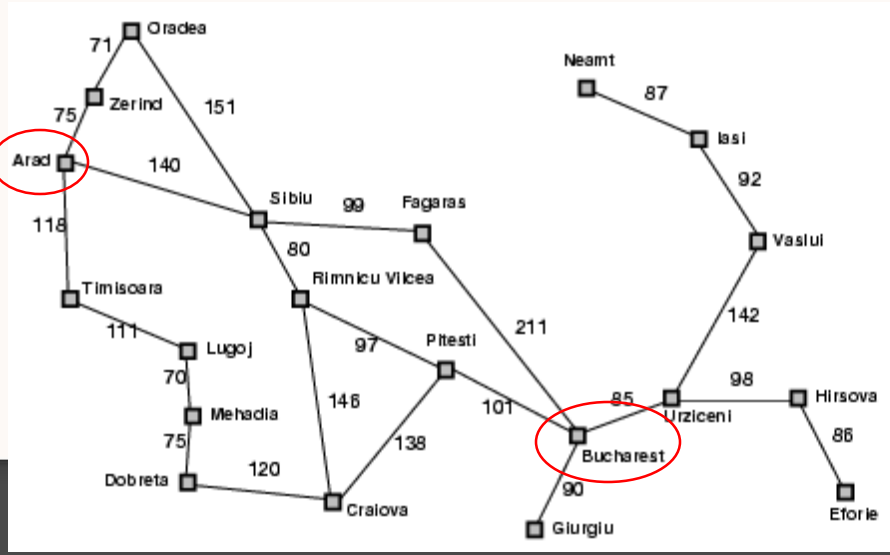
Greedy best-first search

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

Greedy best-first search example

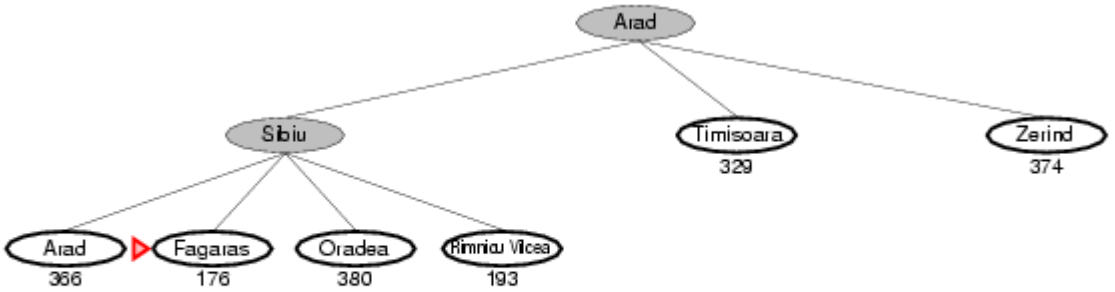


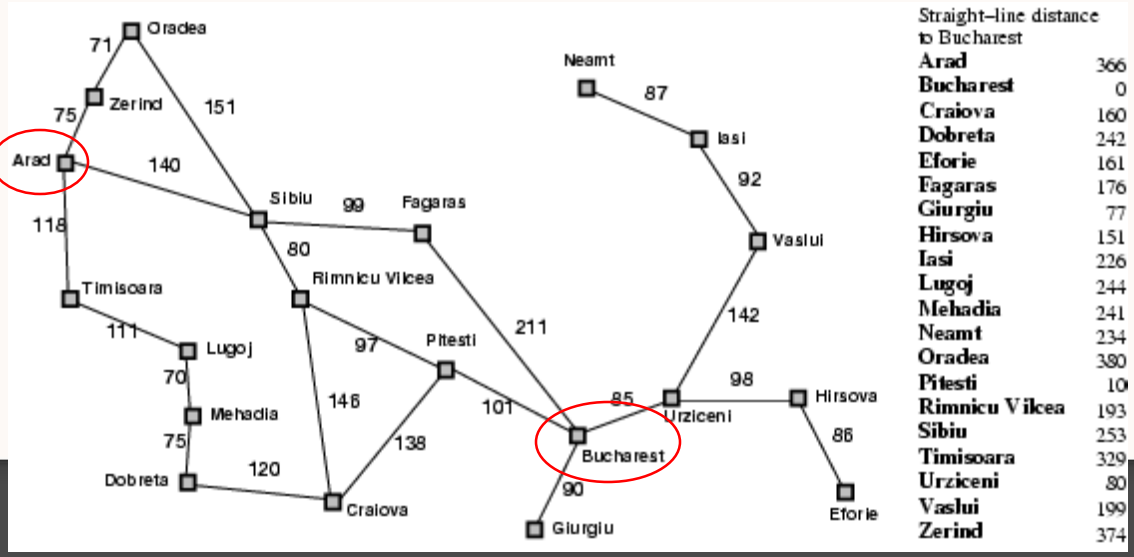
Greedy best-first search example



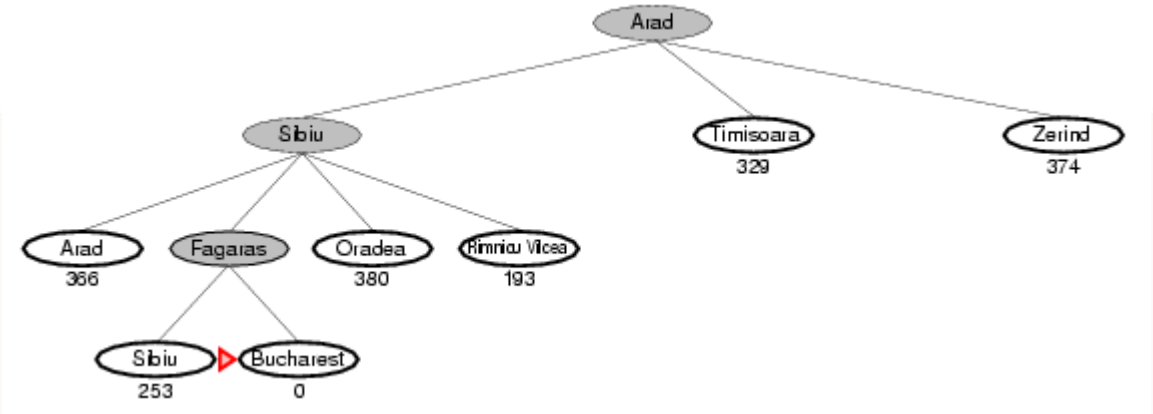
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

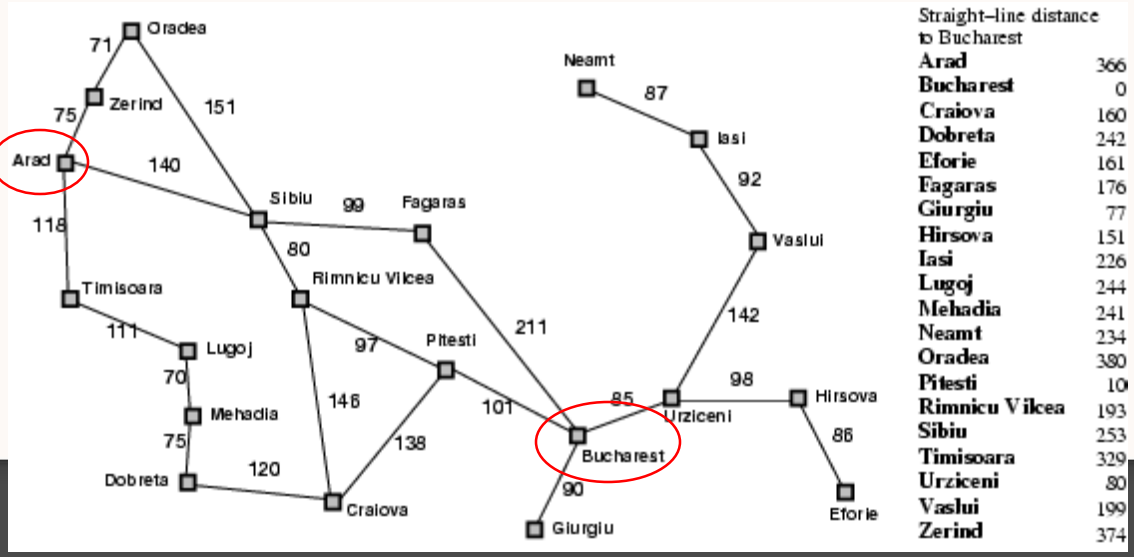
Greedy best-first search example





Greedy best-first search example



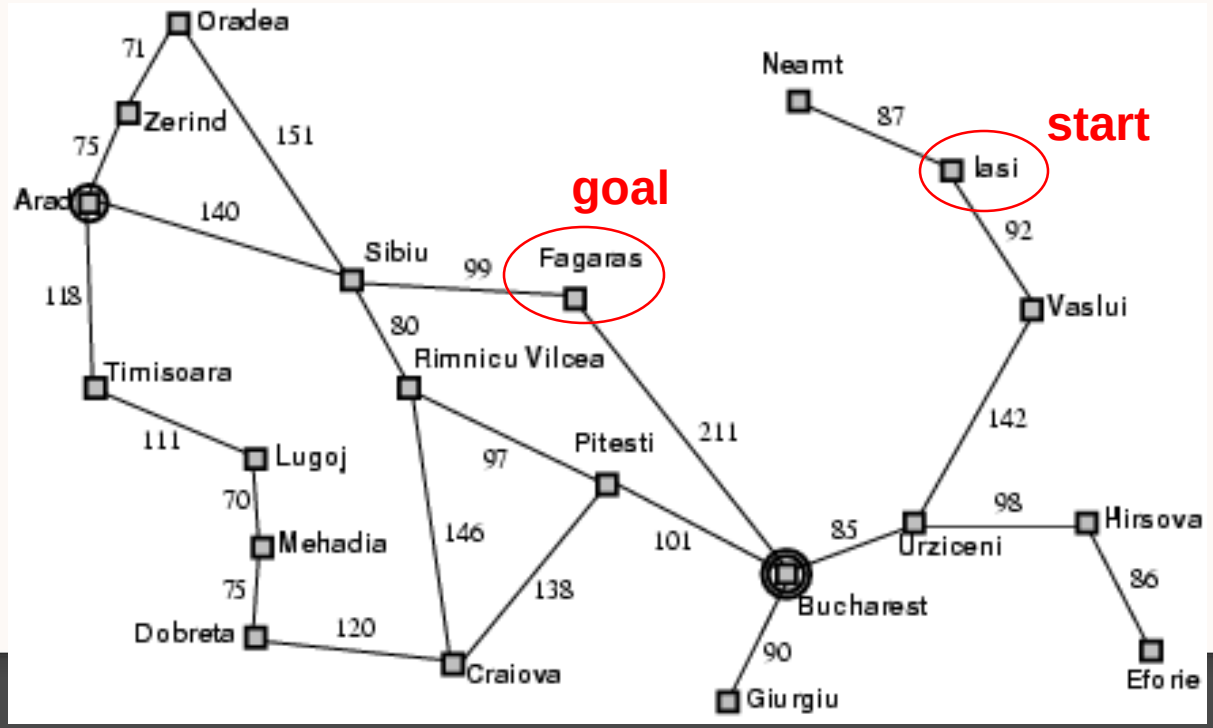


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops



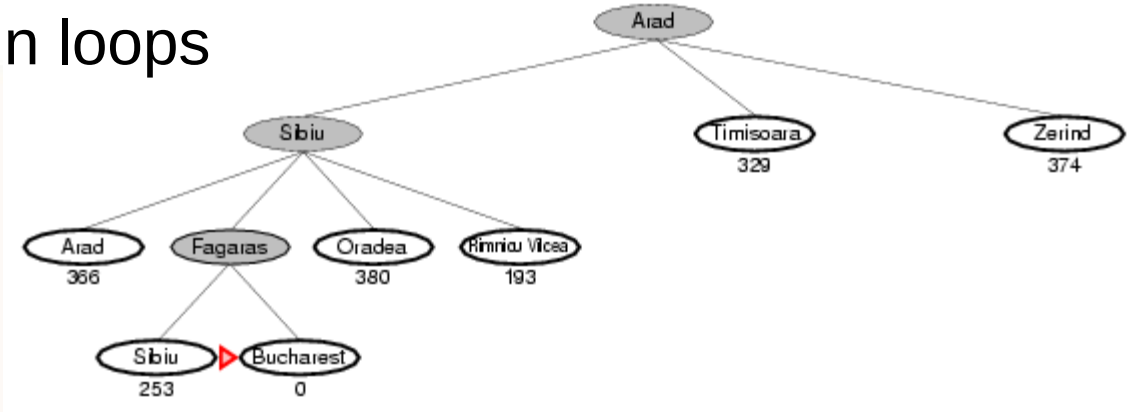
Properties of greedy best-first search

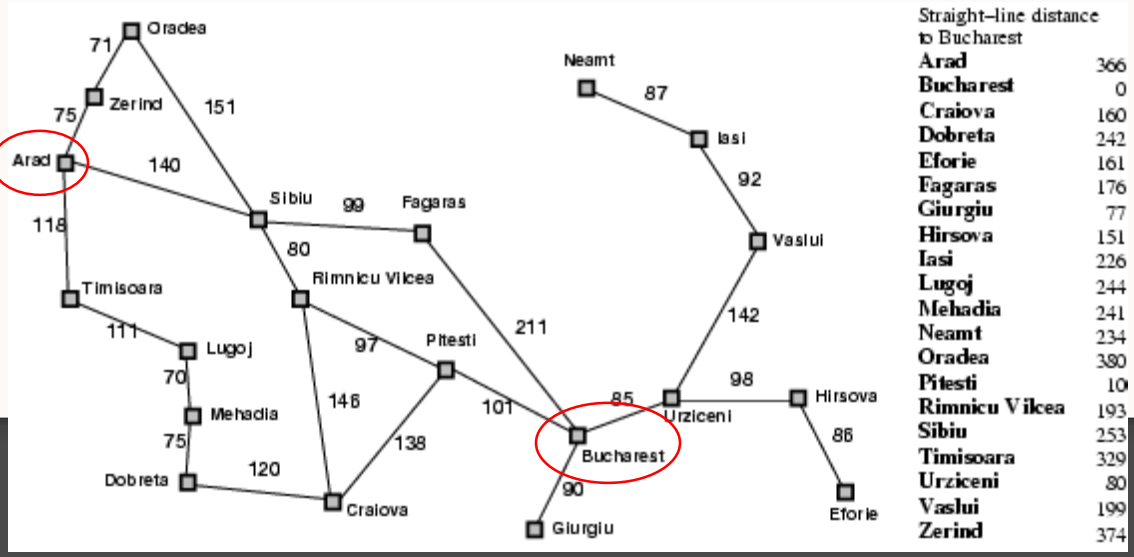
- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No





A* search

BFS

Greedy

A* search

A* Search

A* search

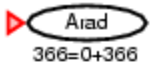
- Idea: avoid expanding paths that are already expensive
- The evaluation function $f(n)$ is the estimated total cost of the path through node n to the goal:

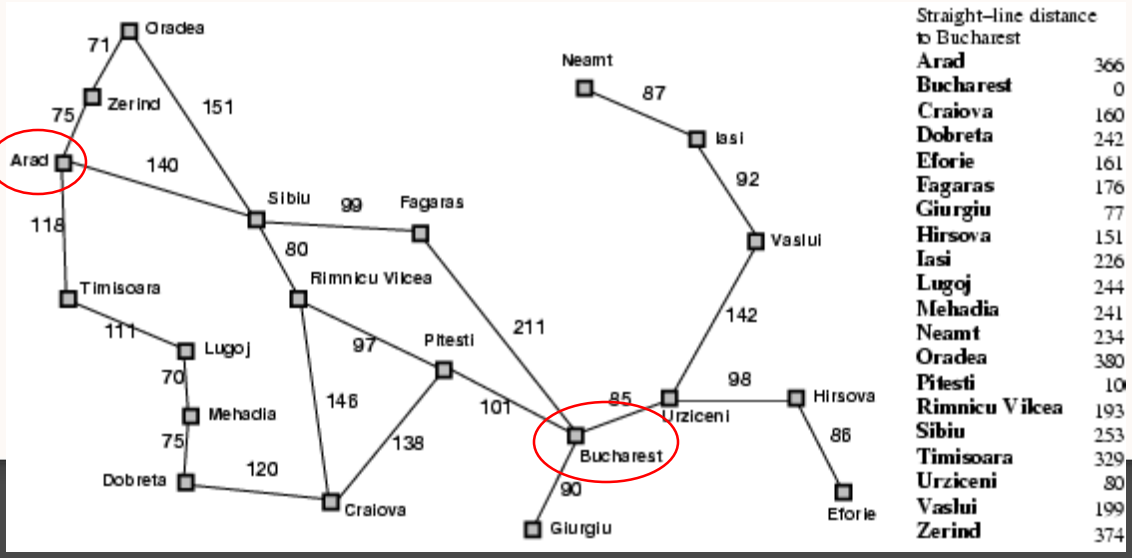
$$f(n) = g(n) + h(n)$$

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

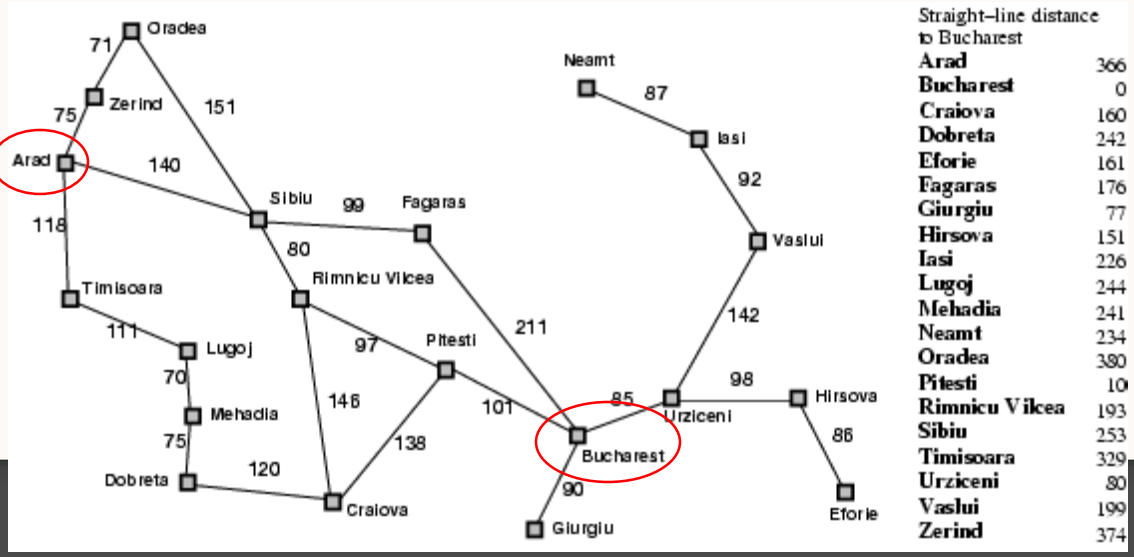
$h(n)$: estimated cost from n to goal (heuristic)

A* search example

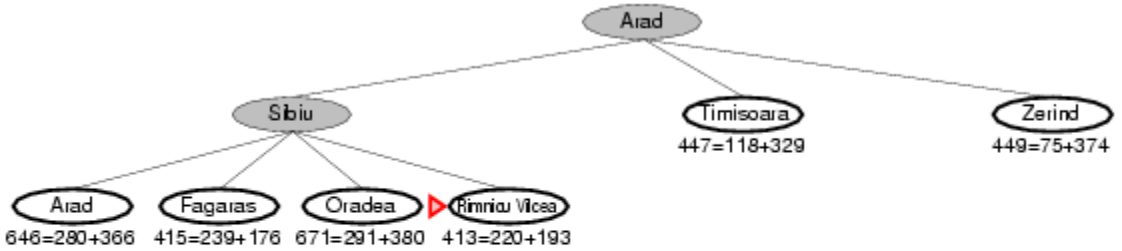


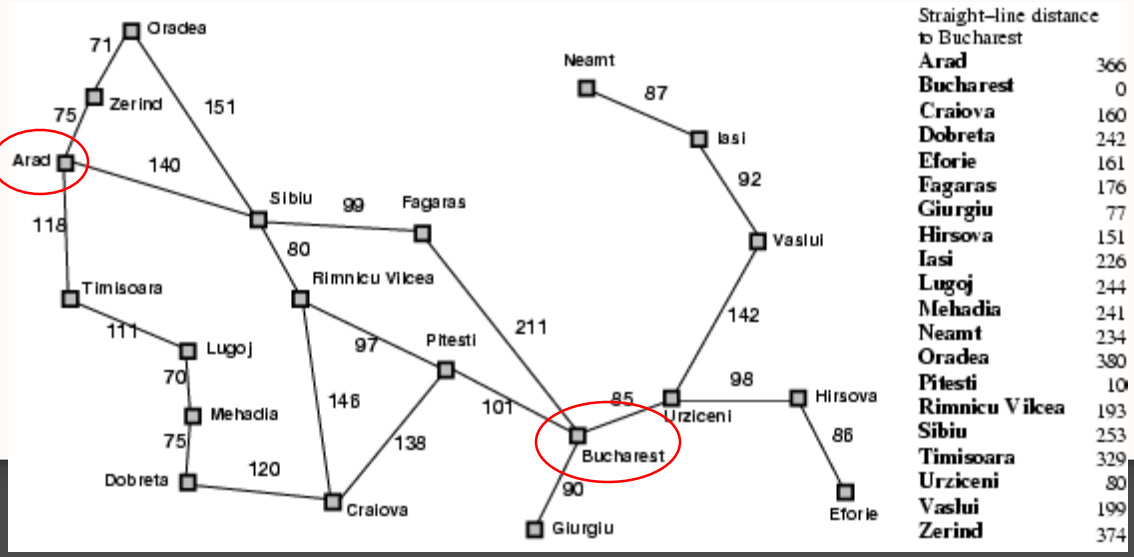


A* search example



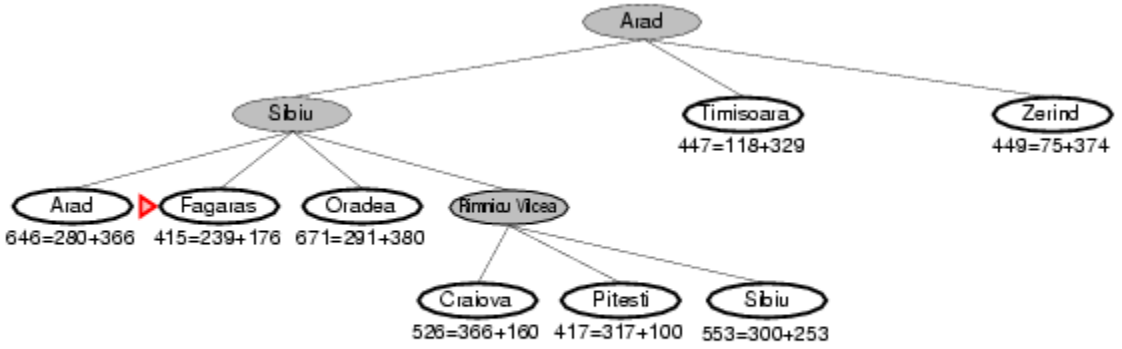
A* search example

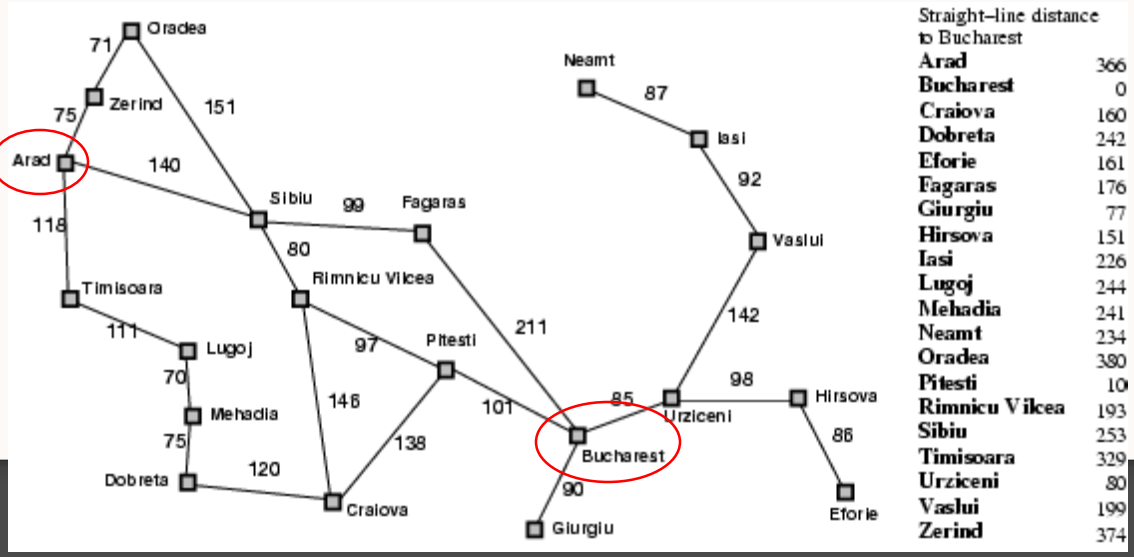




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

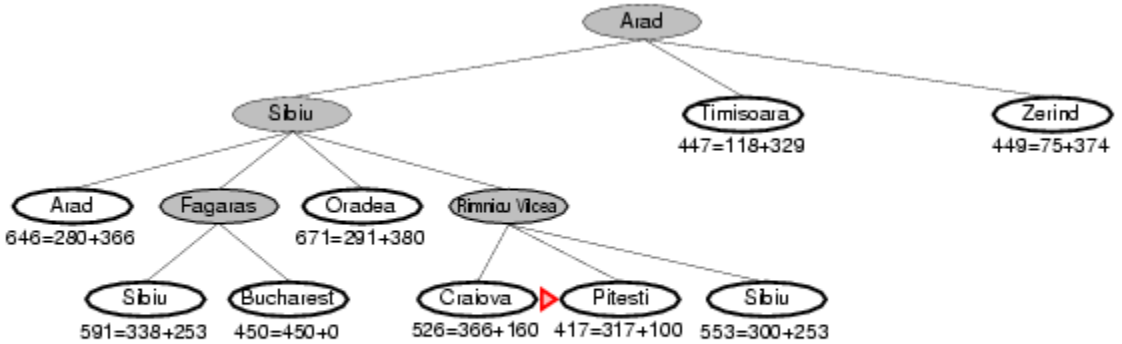
A* search example

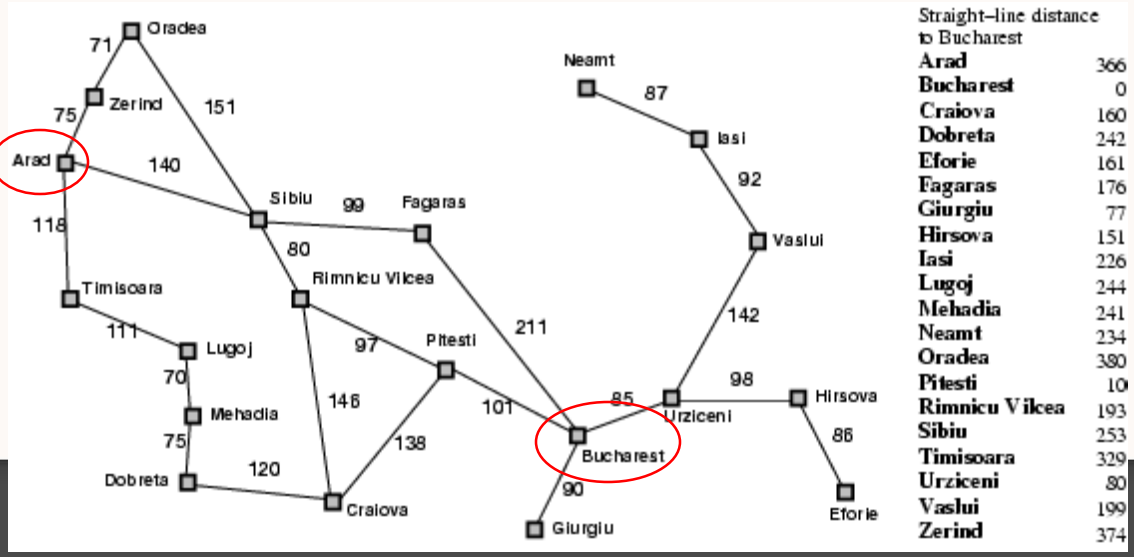




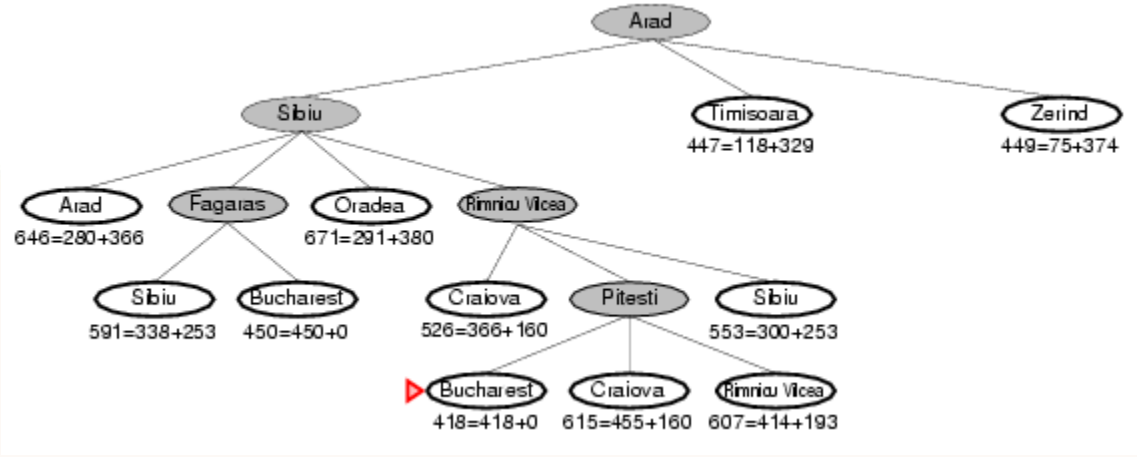
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

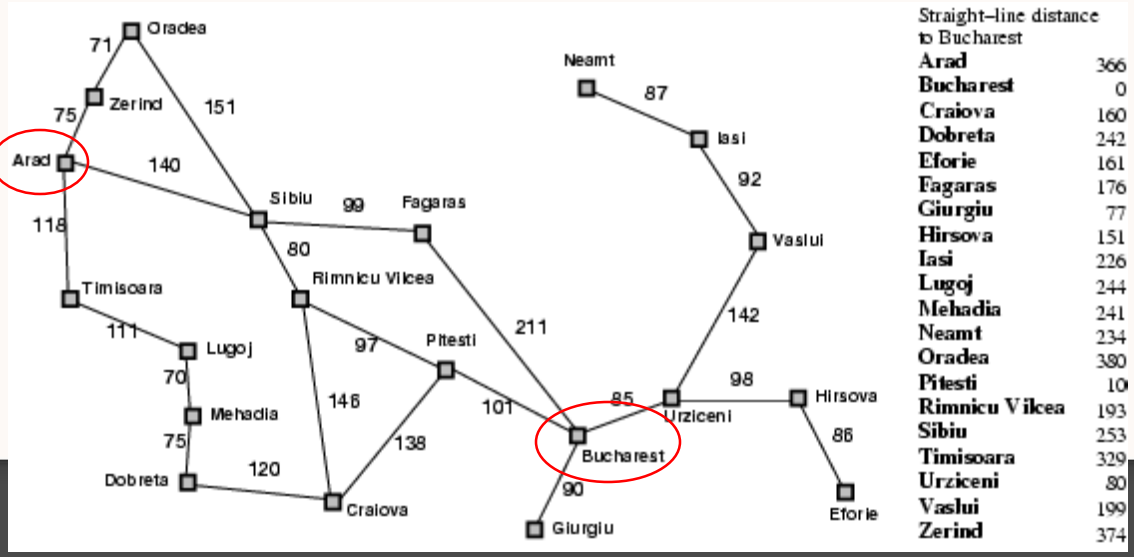
A* search example





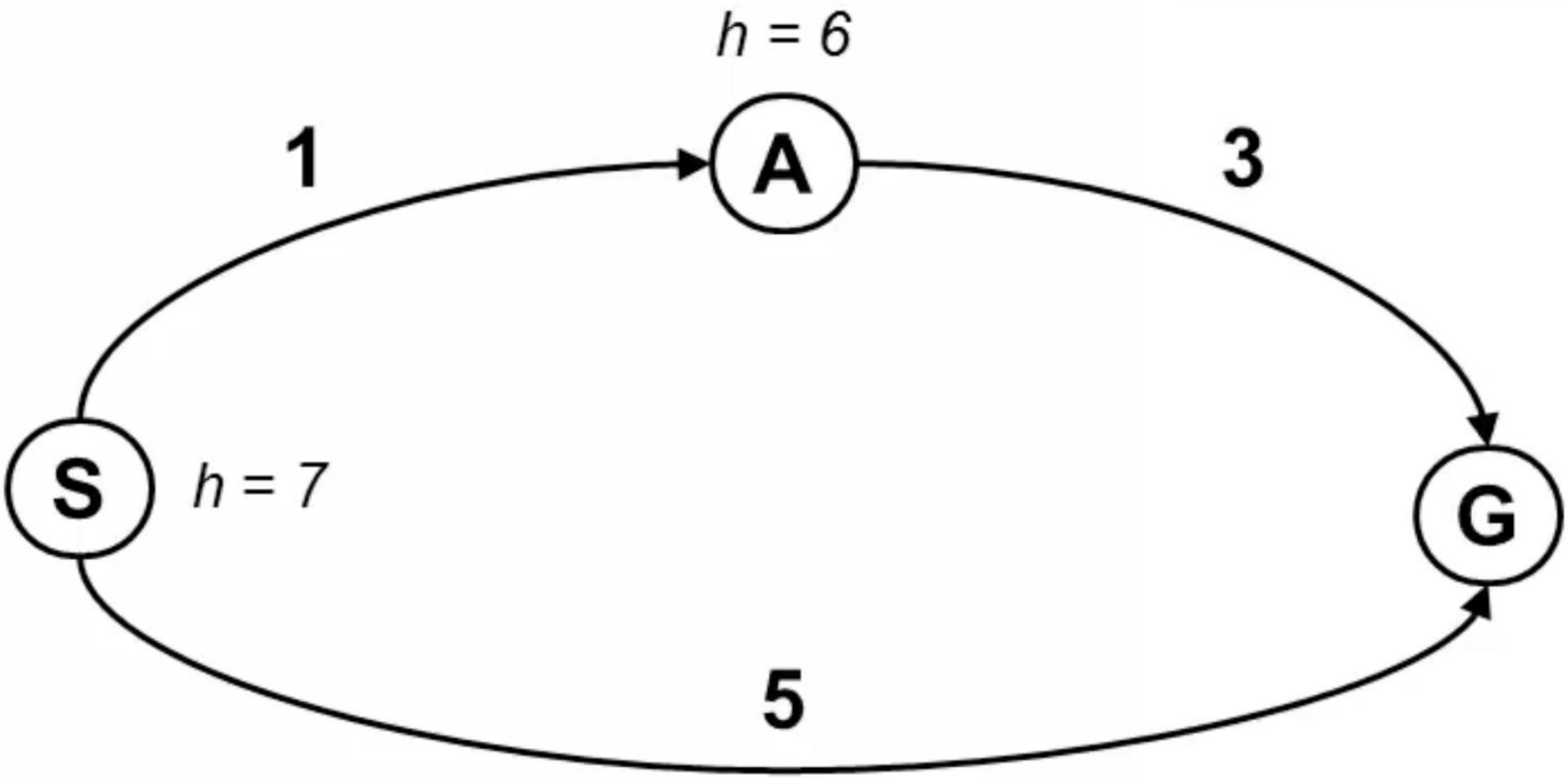
A* search example





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

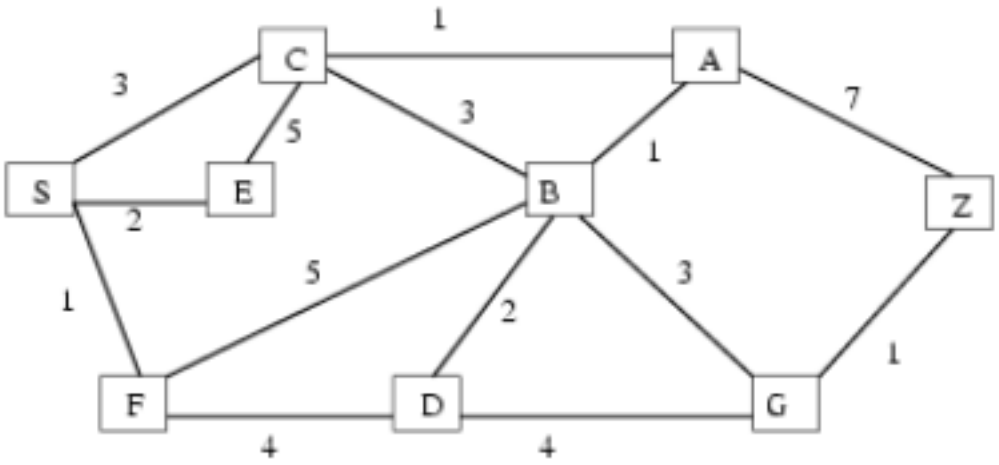


Admissible heuristics

- 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

Exercise 1:

Consider the following graph with start state **S** and goal state **Z**. The numbers on the arcs indicate the cost of traversing that arc.



(a) Same as (a), but using the greedy search with the following heuristic functions:

n	S	F	E	C	D	B	A	G	Z
$h(n)$	8	6	6	5	4	3	2	1	0

(b) Same as (a), but using the A* search with the following heuristic functions:

n	S	F	E	C	D	B	A	G	Z
$h(n)$	8	6	6	5	4	3	2	1	0