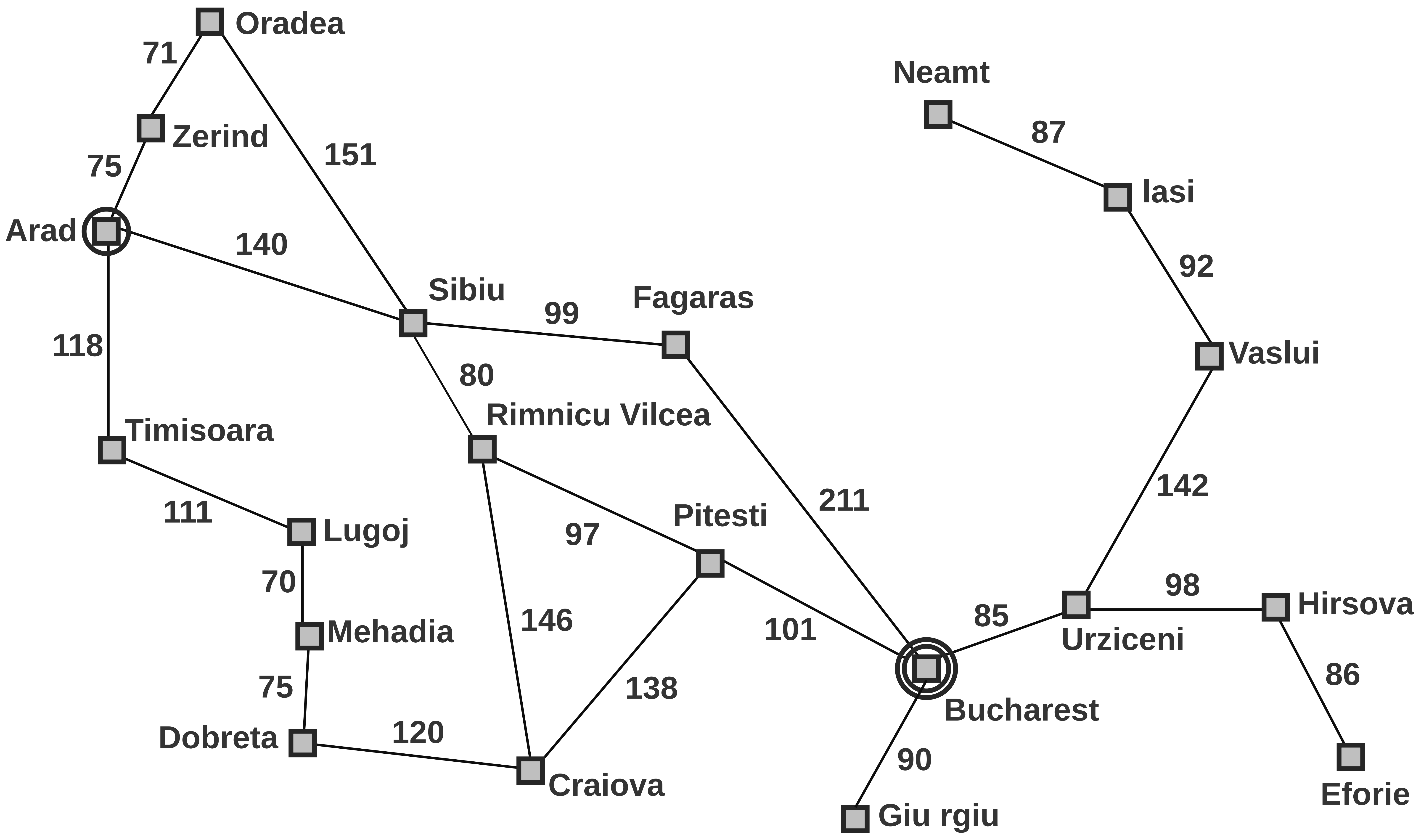
인공지능의 기초

문제 해결 및 탐색 전략

Example: Romania



MEMO

Example: Romania

- On holiday in Romania, the flight leaves tomorrow from Bucharest

- Formulate initial state and goal
  - Currently in Arad, and be in Bucharest

- Formulate problem:
  - States: various cities
  - Actions: drive between cities

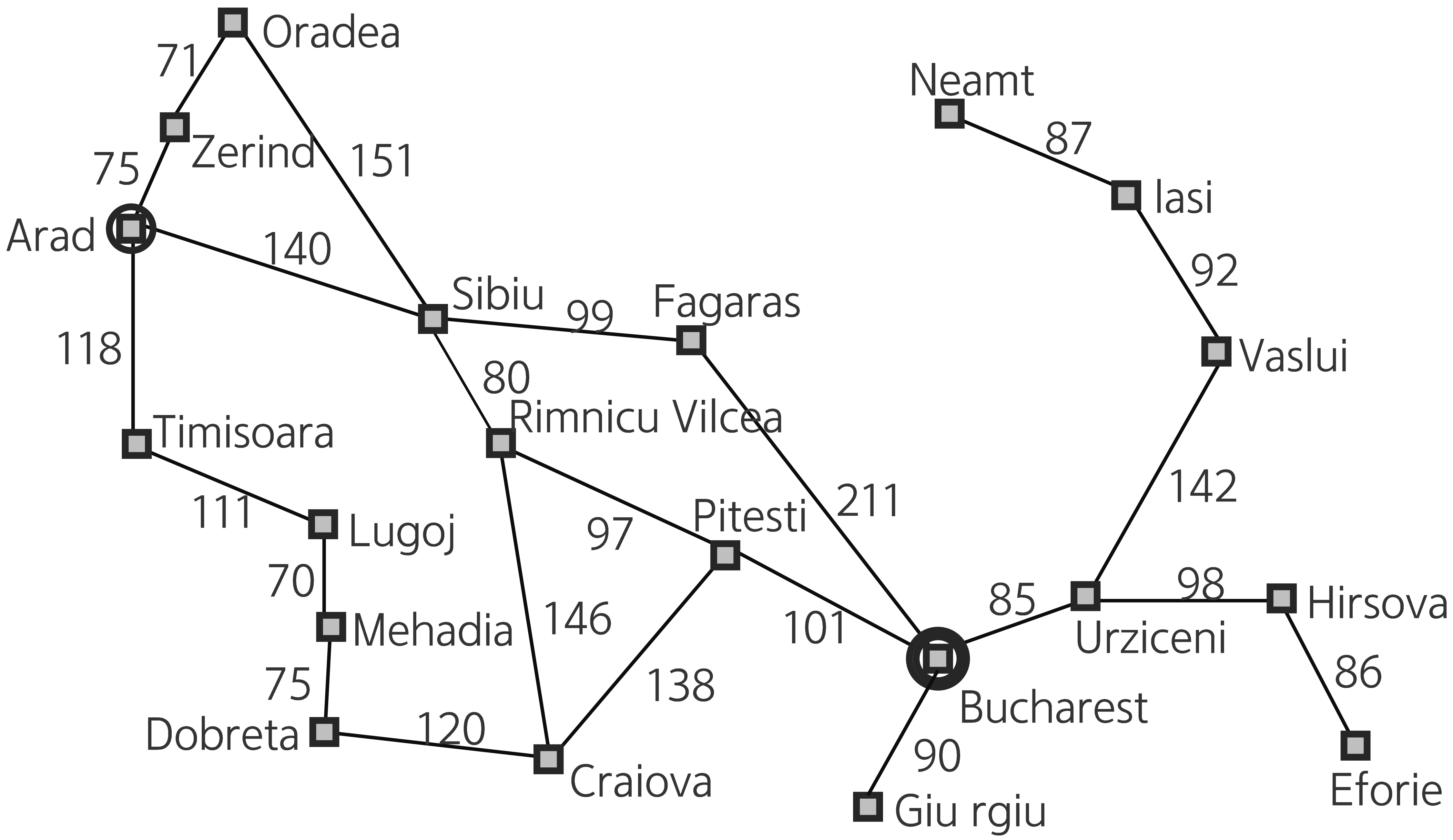
- Find solution:
  - A sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

MEMO

Search as Problem Solving

- Need to search a space of possible solutions
  - ▶ There are many sequences of actions, each with their own utility

- We want to find, or search for, the best one



MEMO

Problem

Formally defined by four components

1. Initial state
   - The state that the agent starts in
   - e.g. at Arad

2. Possible actions
   - Successor function $\mathrm{successor\_fn}(x)$: given a state x, return a set of (action, successor) ordered pairs
   - e.g. $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \cdots\}$

MEMO

Problem

Formally defined by four components

3. Goal test
   - Determines whether a given state is a goal state
   - Explicit, e.g., x = "at Bucharest”
   - Implicit, e.g., Checkmate(x)

4. Path cost
   - Assigns a numeric cost to each path
   - Step cost $c(x, a, y) \geq 0$: a cost of taking action a to go from x to y
   - e.g., sum of distances, number of actions executed, etc.

Solution: a sequence of actions leading from the initial state to a goal state

MEMO

Selecting a State Space

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

MEMO

Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

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

- States? Locations of tiles
- Actions? Move blank left, right, up, down
- Goal test? Goal state (given)
- Path cost? 1 per move

MEMO

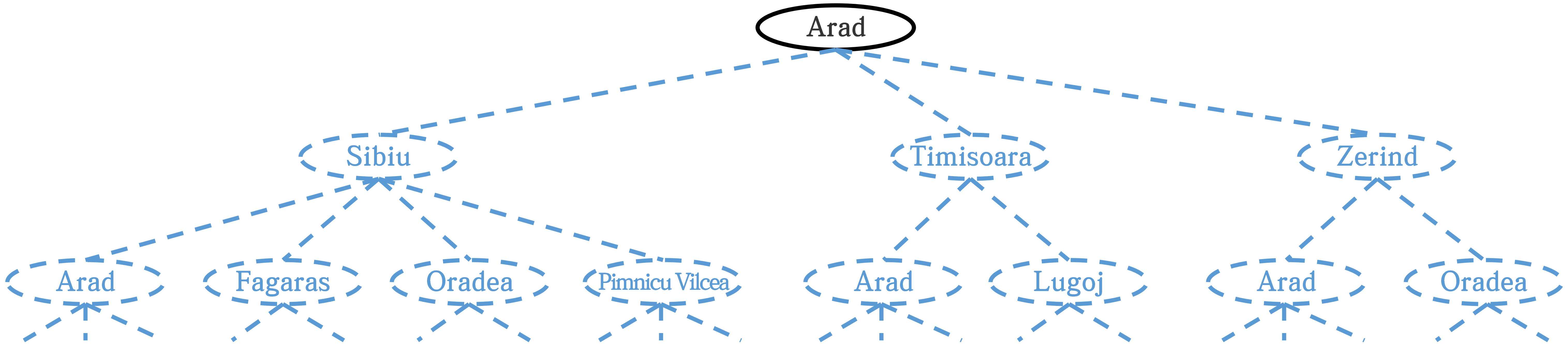
Tree Search Algorithms

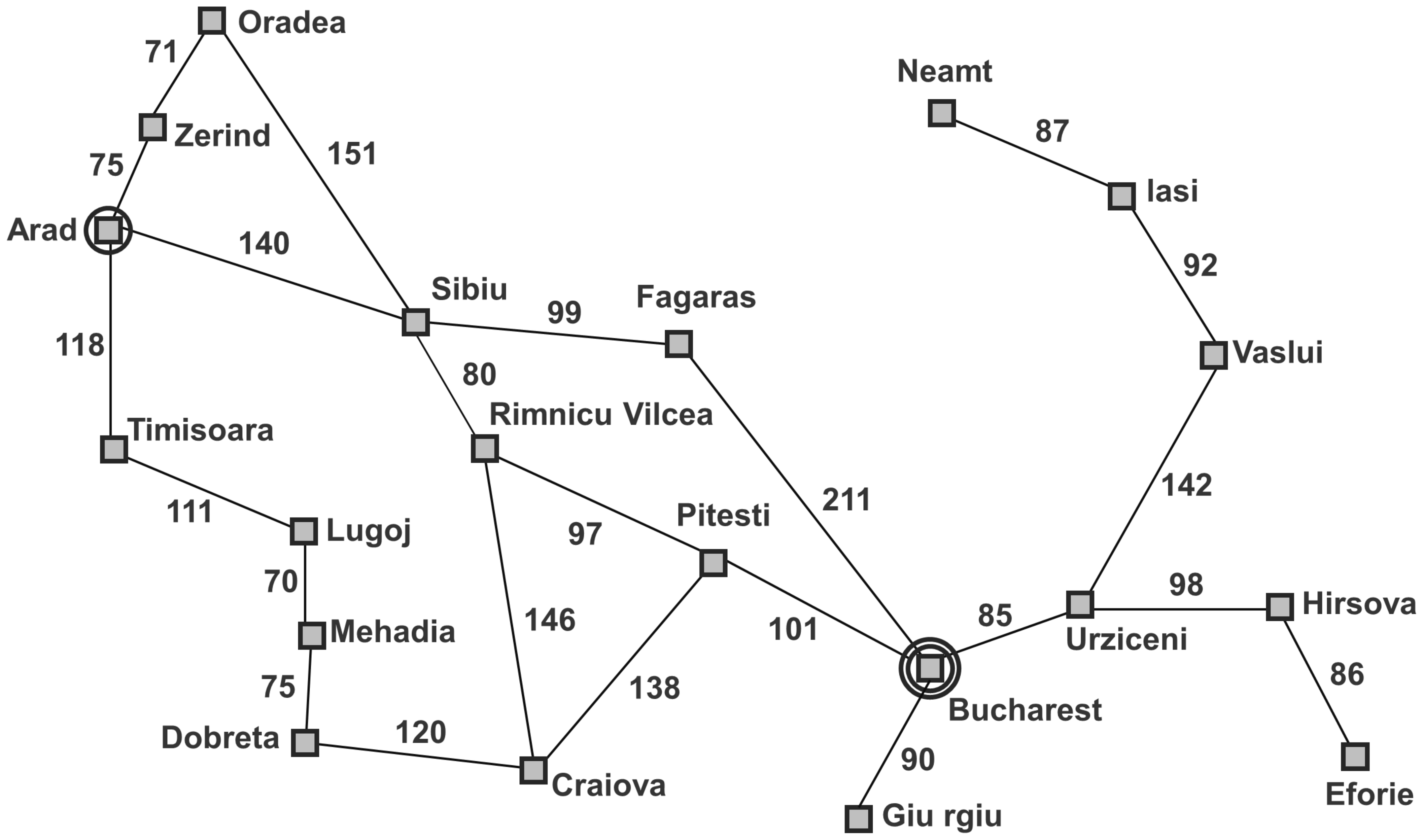
Basic idea

- Exploration of state space by generating successors of already-explored states (a.k.a. ~ expanding states)
- Every state is evaluated: is it a goal state?

MEMO

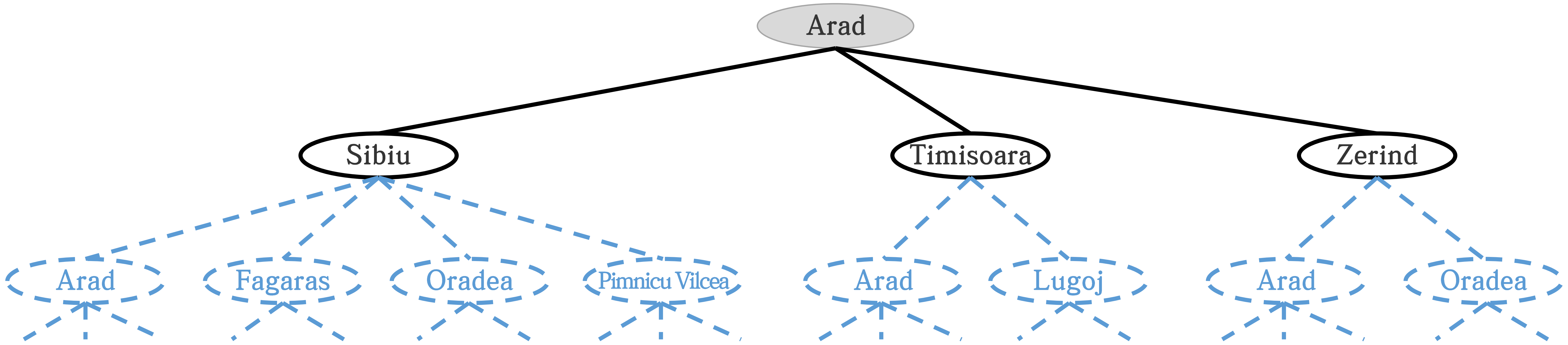
Tree Search Example

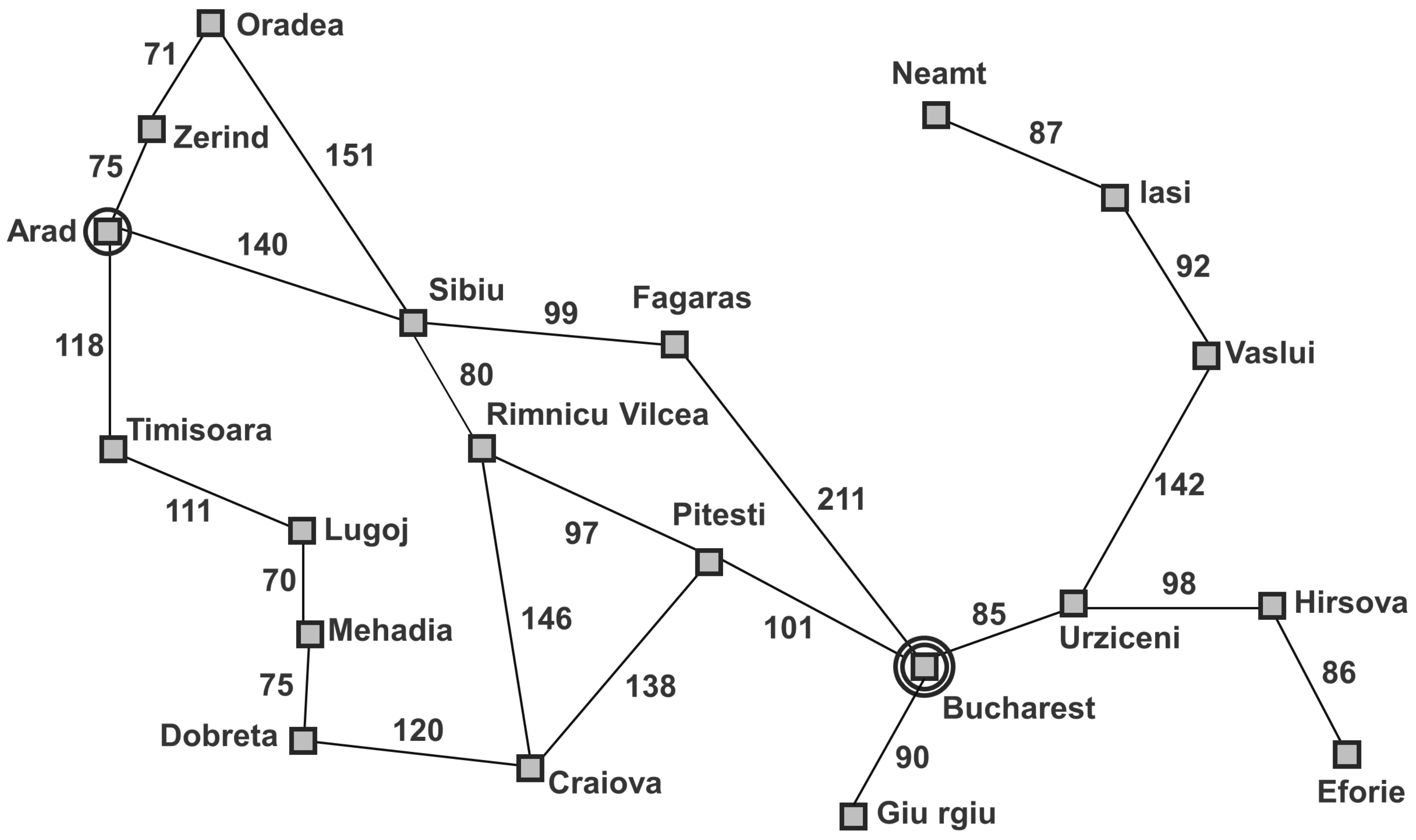




MEMO

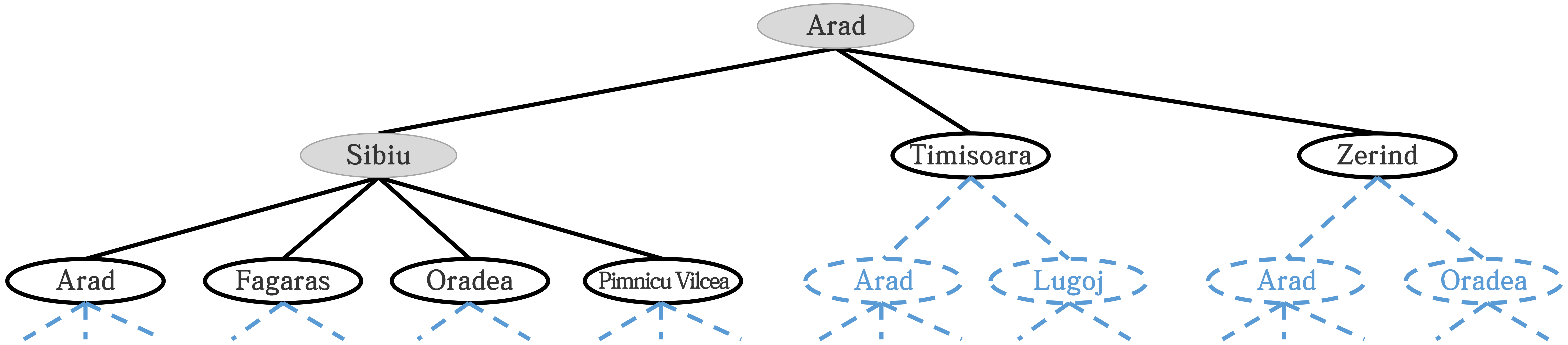
Tree Search Example





MEMO

Tree Search Example



State space forms a tree structure

- Root = start state
- Each node represents a state
- Actions are branches, children are all possible next-states

Search involves expanding a frontier of potential next states

MEMO

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?

MEMO

Search Strategies

- Uninformed 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

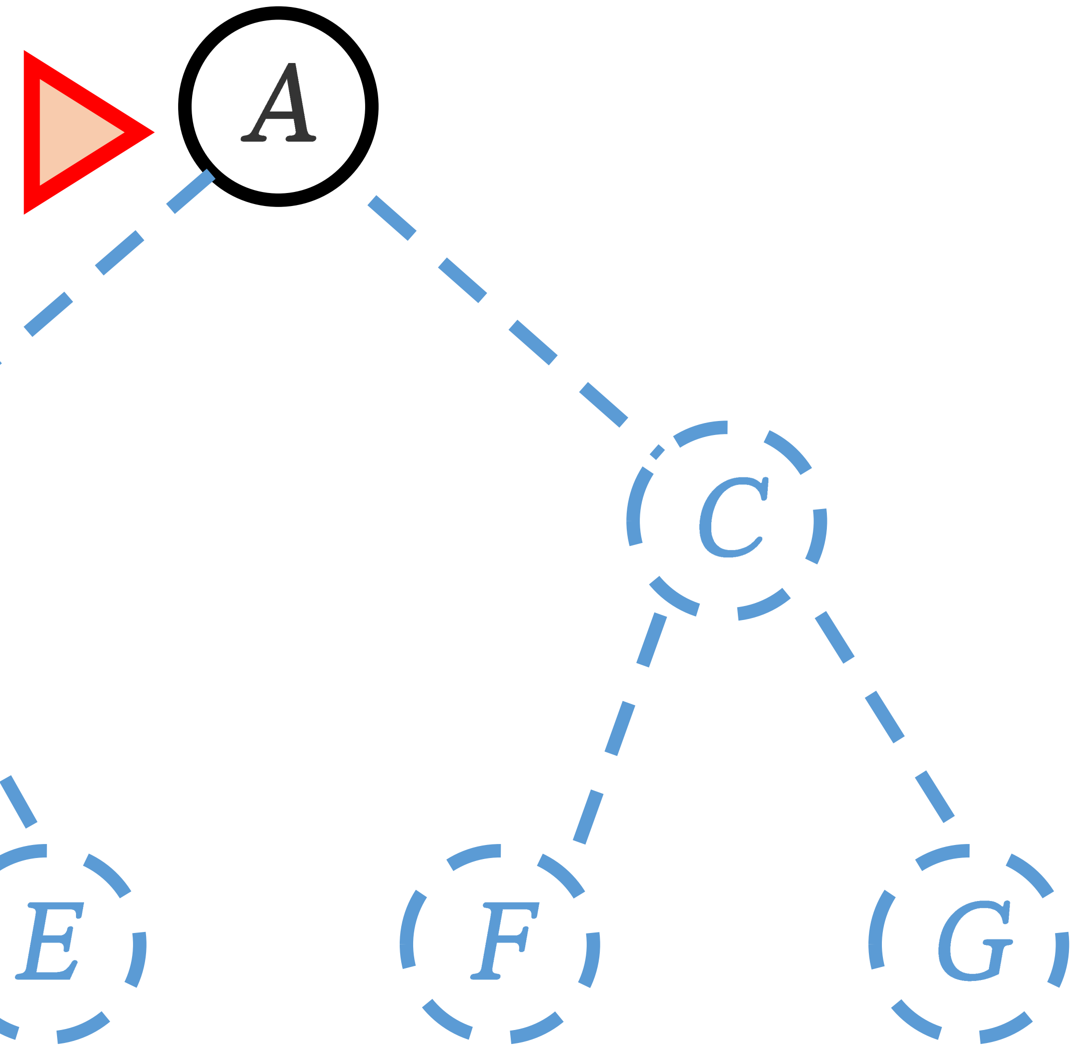
MEMO

Breadth-First Search

- Expand shallowest unexpanded node
- Implementation:
  - ▶ Fringe = FIFO queue, i.e., new successors go at end search

FIFO queue

A				



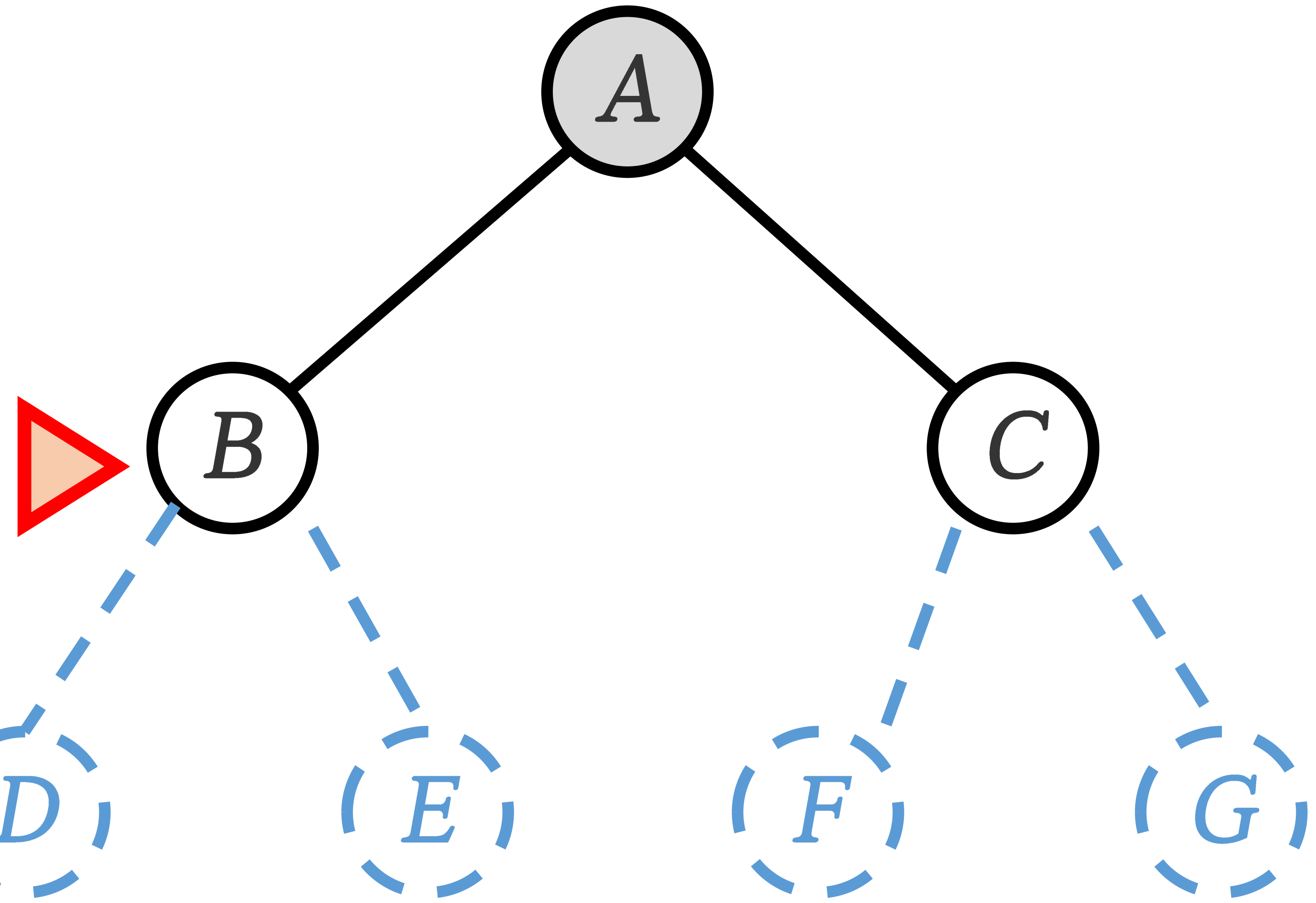
MEMO

Breadth-First Search

- Expand shallowest unexpanded node
- Implementation:
  - ▸ Fringe = FIFO queue, i.e., new successors go at end search

FIFO queue

B	C			



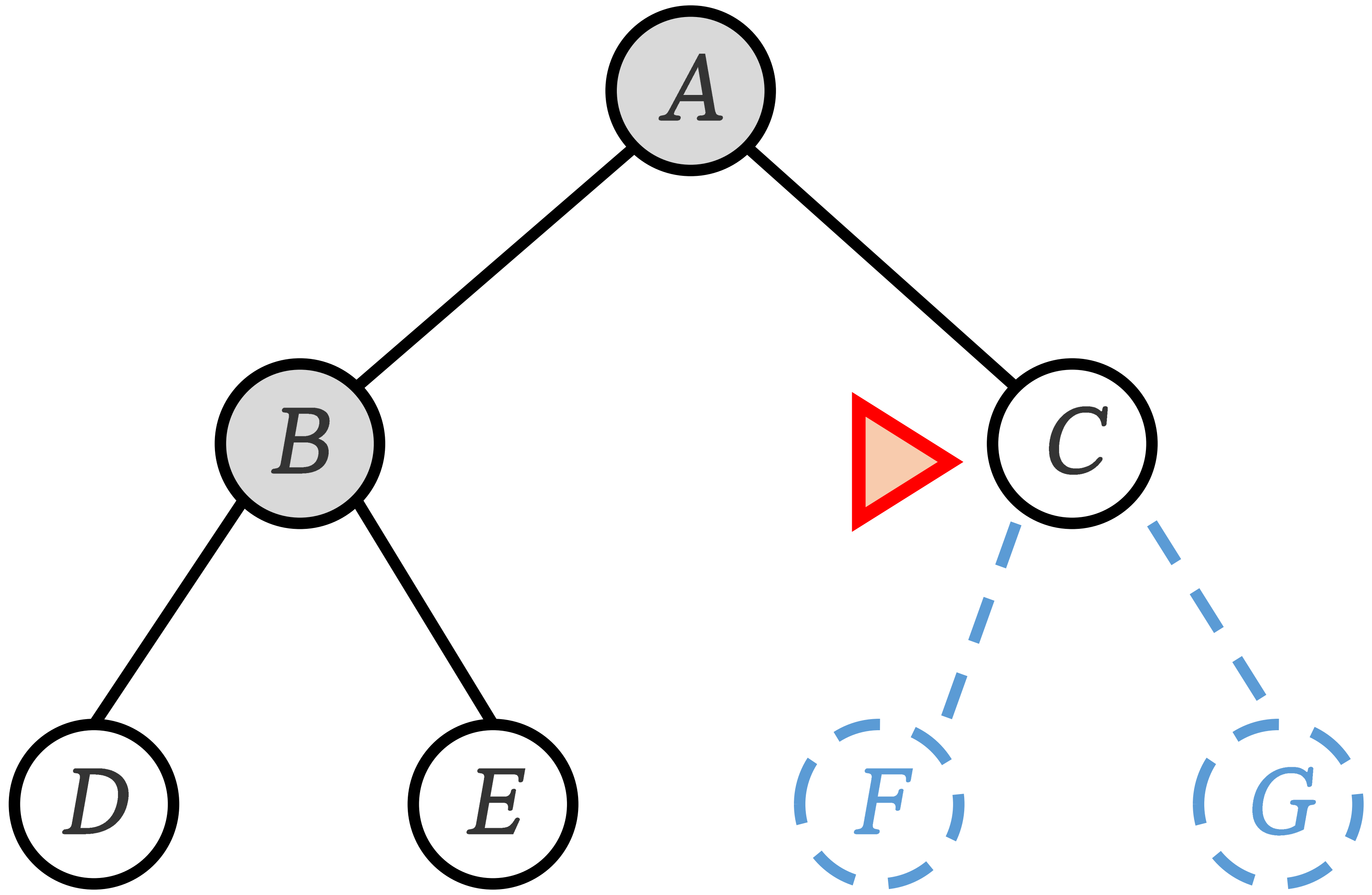
MEMO

Breadth-First Search

- Expand shallowest unexpanded node
- Implementation:
  - Fringe = FIFO queue, i.e., new successors go at end search

FIFO queue

C	D	E		



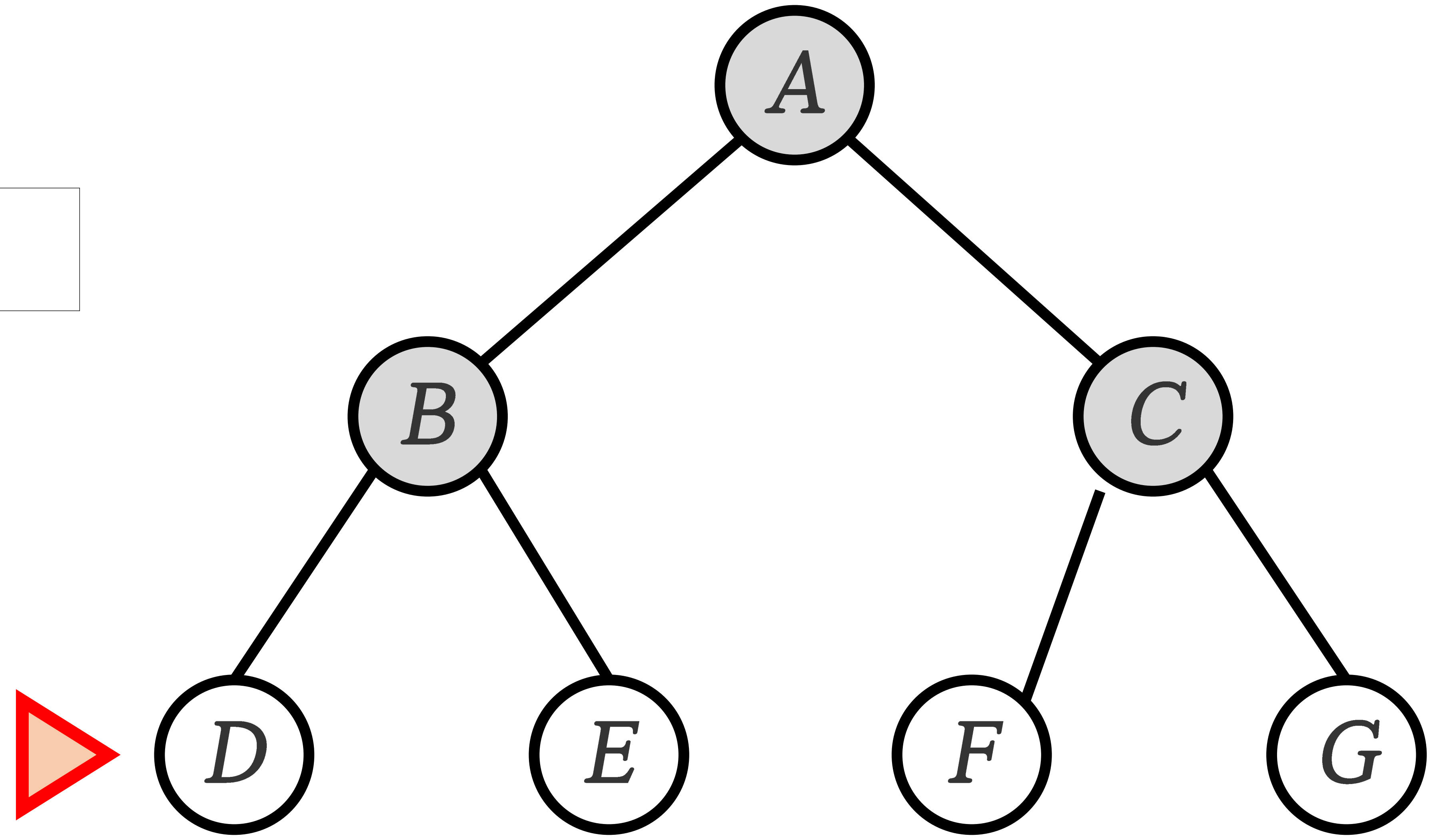
MEMO

Breadth-First Search

- Expand shallowest unexpanded node
- Implementation:
  - ▸ Fringe = FIFO queue, i.e., new successors go at end search

FIFO queue

D	E	F	G	



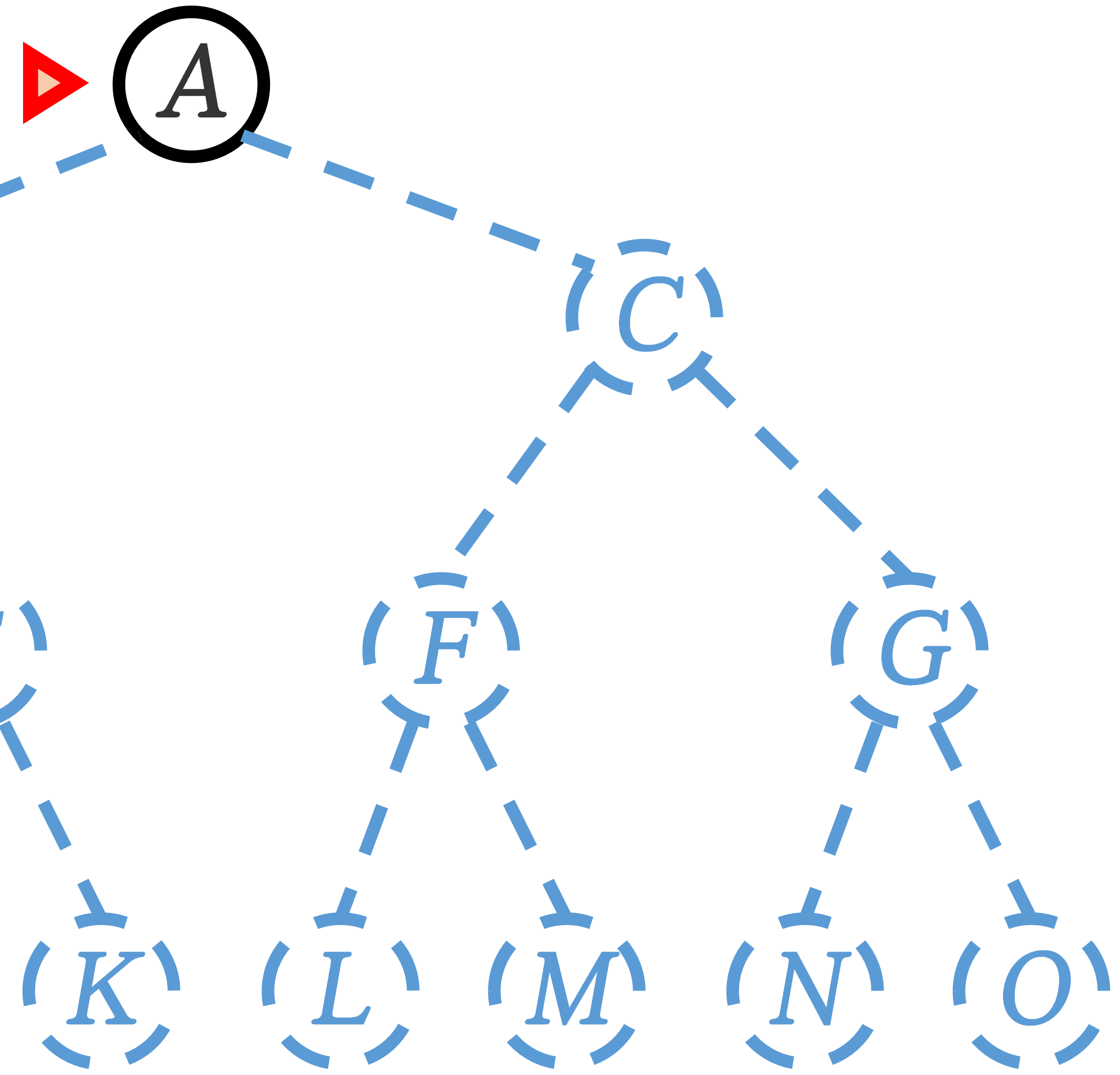
MEMO

Depth-First search

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

LIFO queue

A				



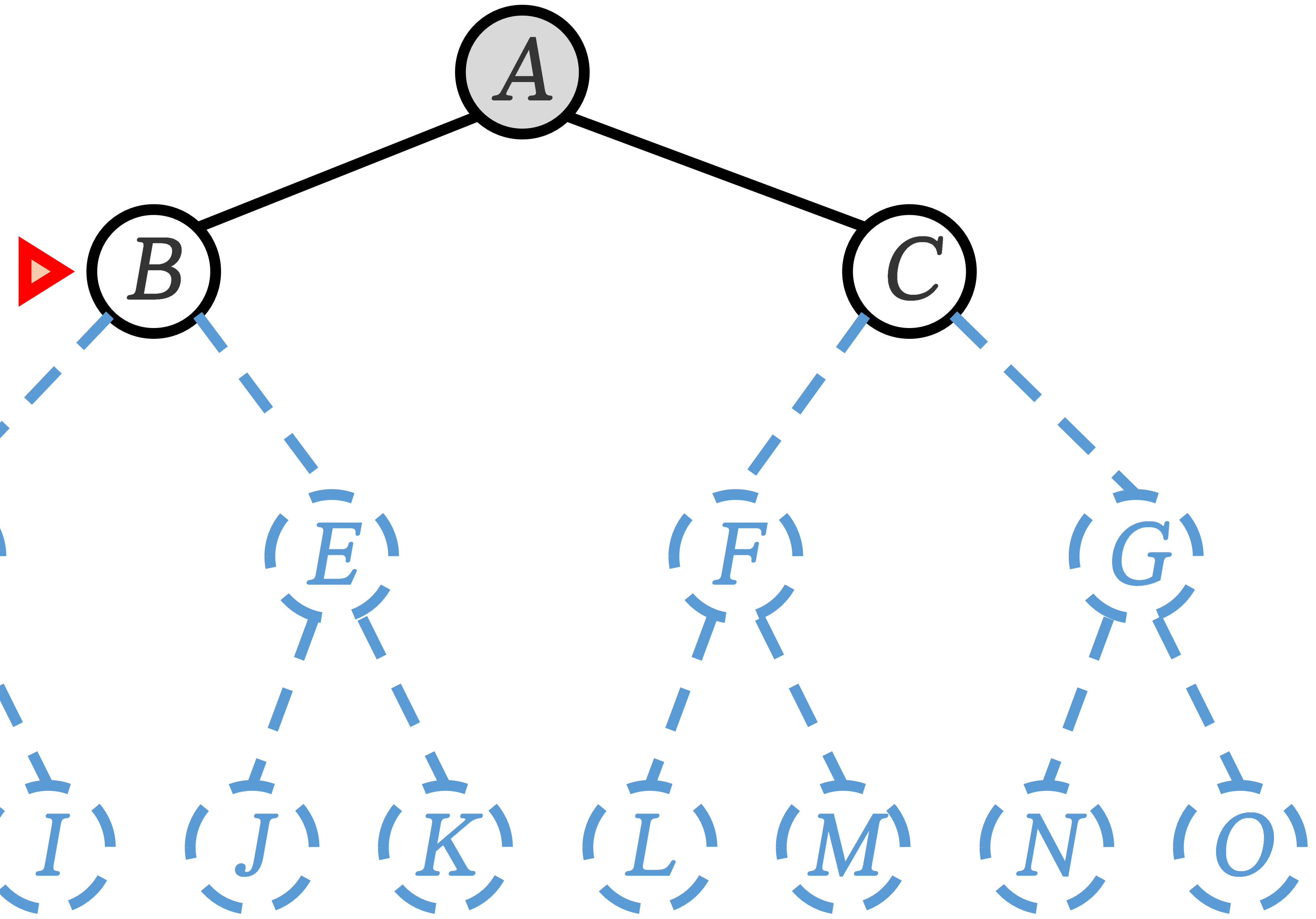
MEMO

Depth-First search

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

LIFO queue

B	C			



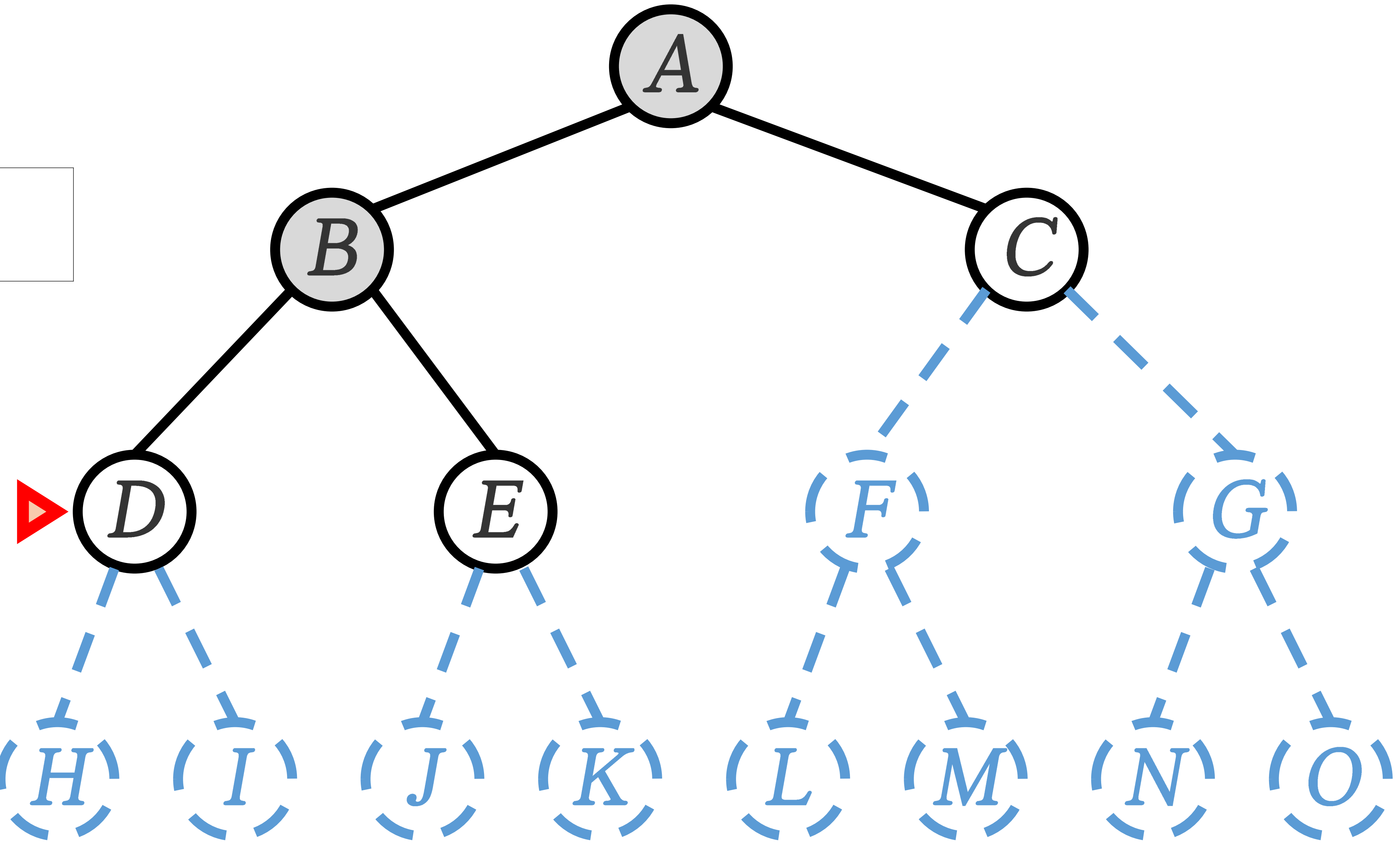
MEMO

Depth-First search

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

LIFO queue

D	E	C		



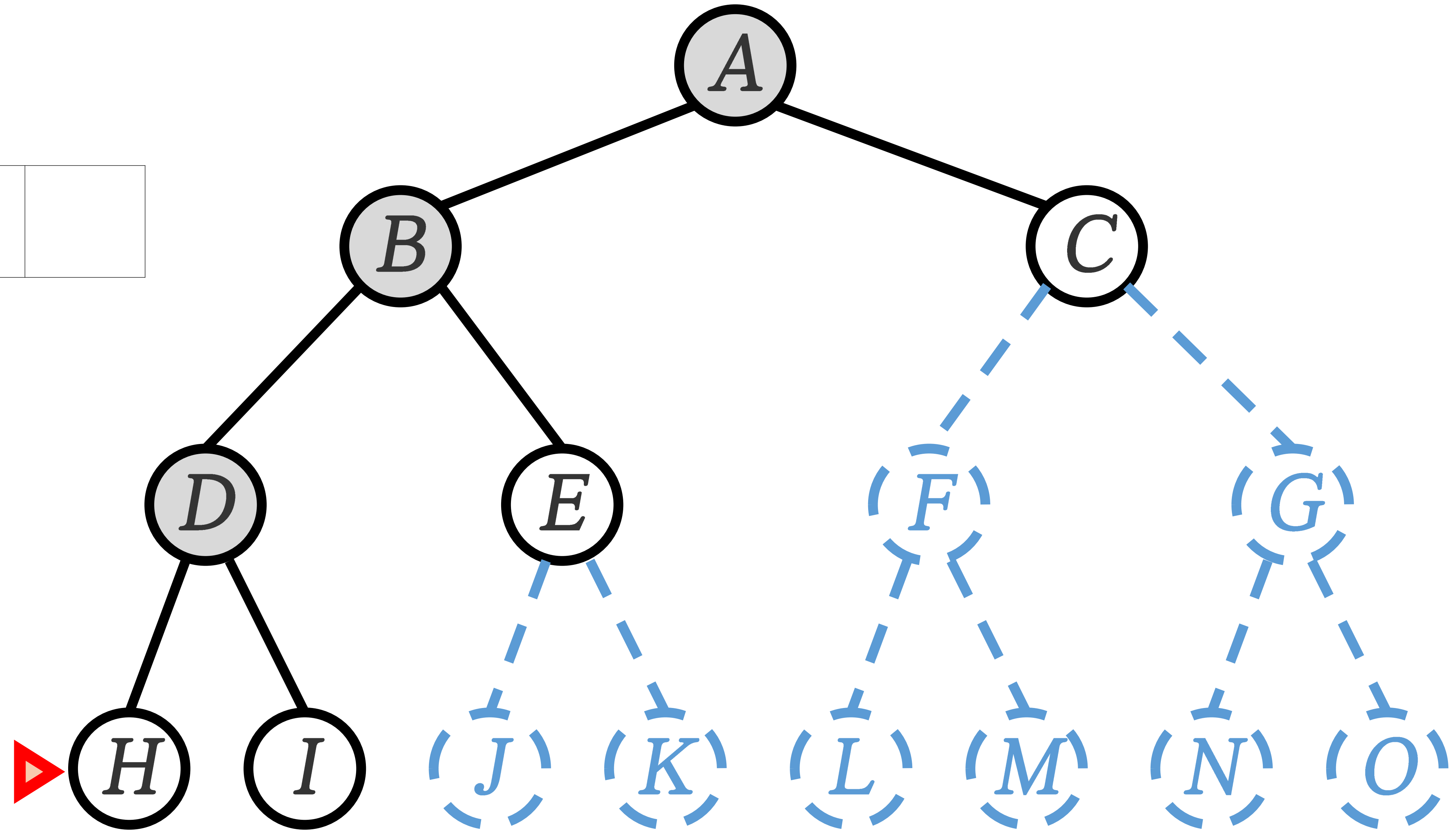
MEMO

Depth-First search

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

LIFO queue

H	I	E	C	



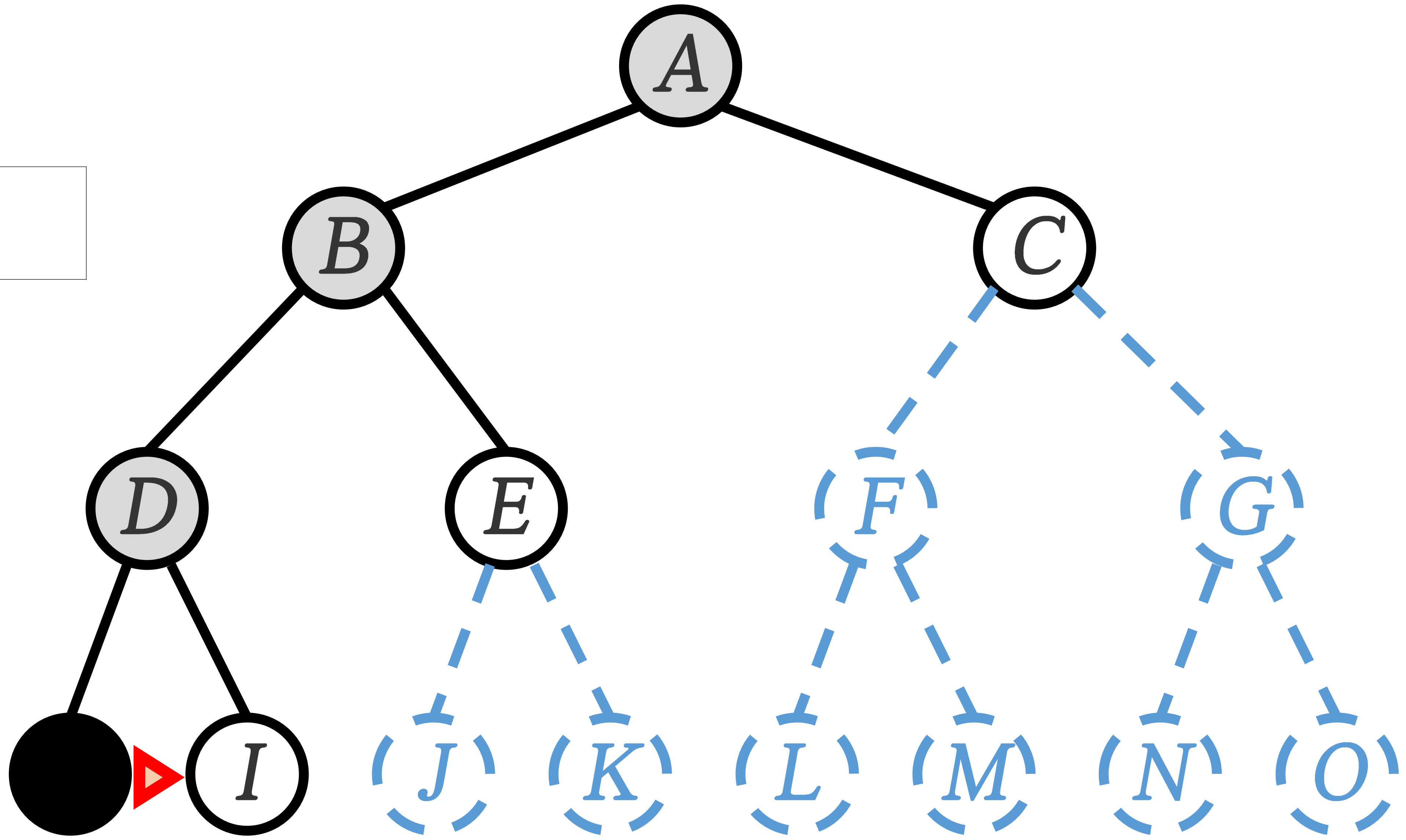
MEMO

Depth-First search

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

LIFO queue

I	E	C		



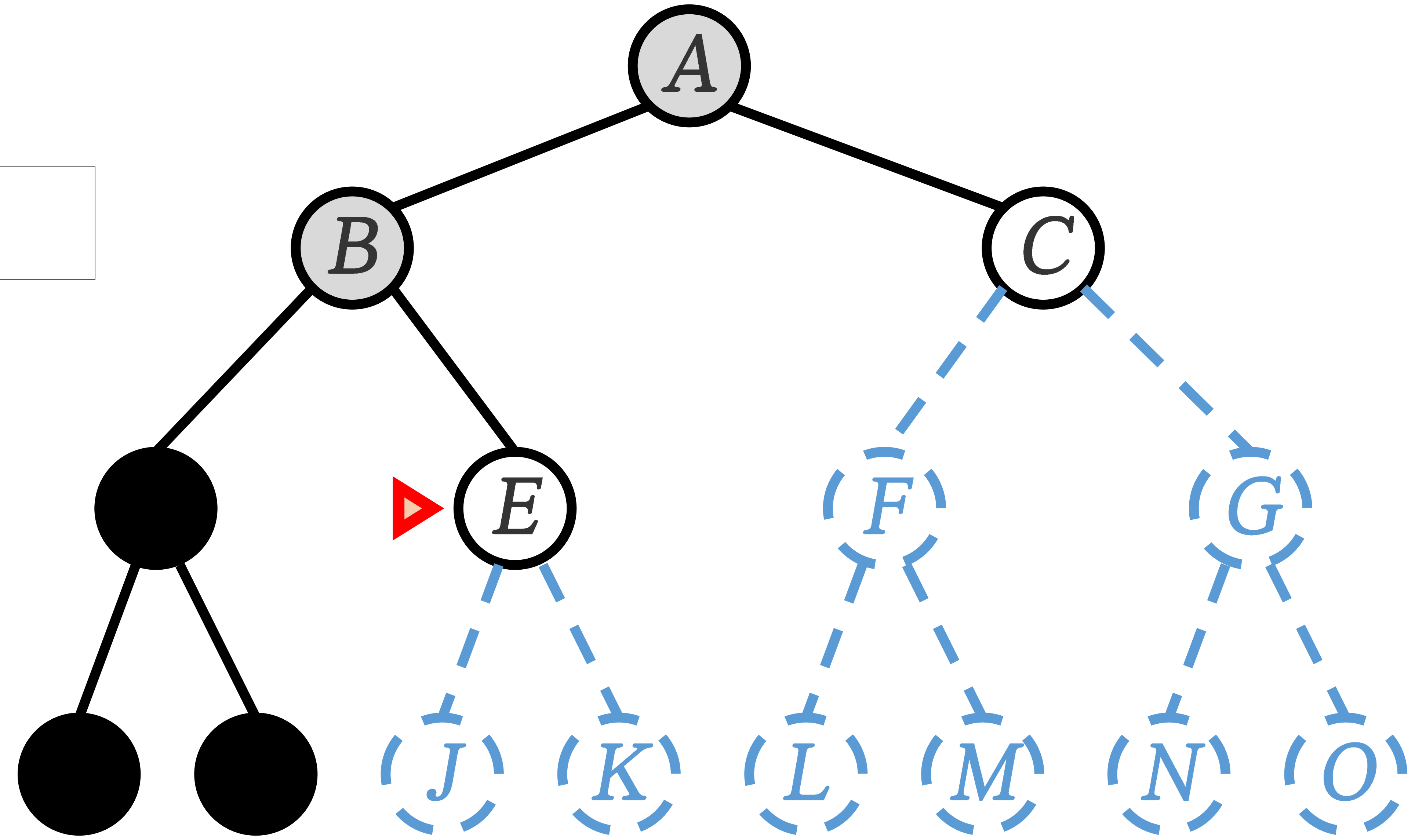
MEMO

Depth-First search

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

LIFO queue

E	C			



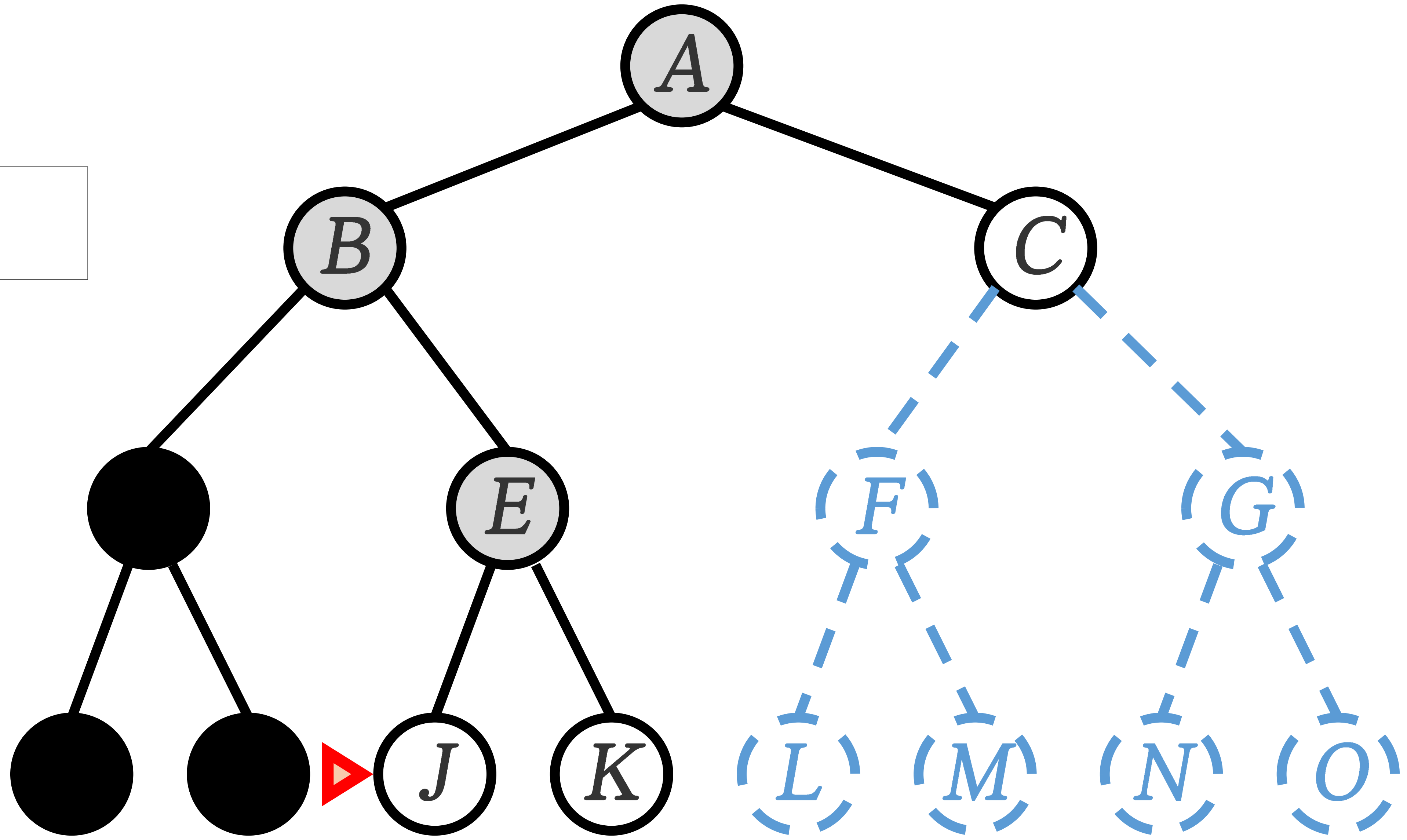
MEMO

Depth-First search

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

LIFO queue

J	K	C		



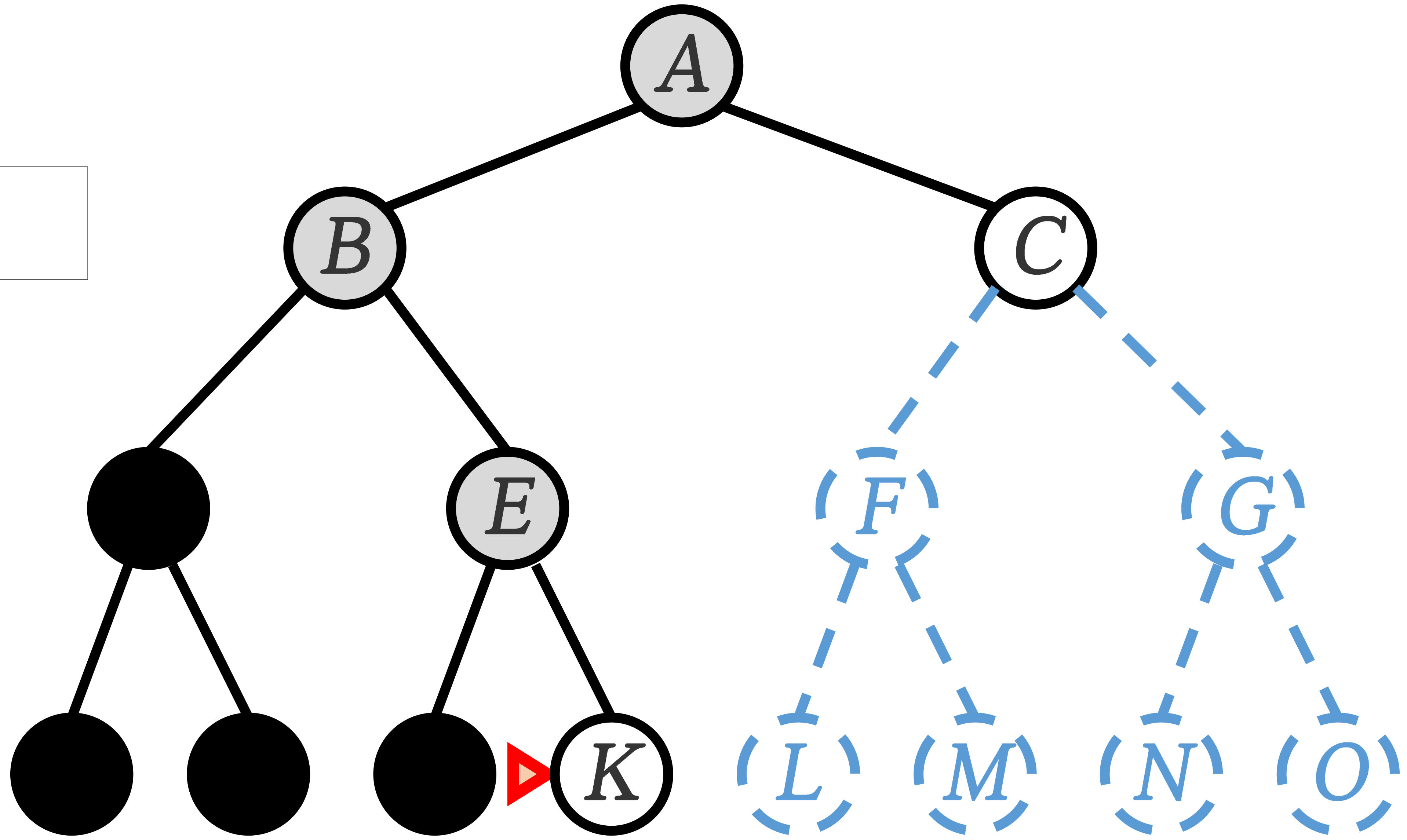
MEMO

Depth-First search

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

LIFO queue

K	C			



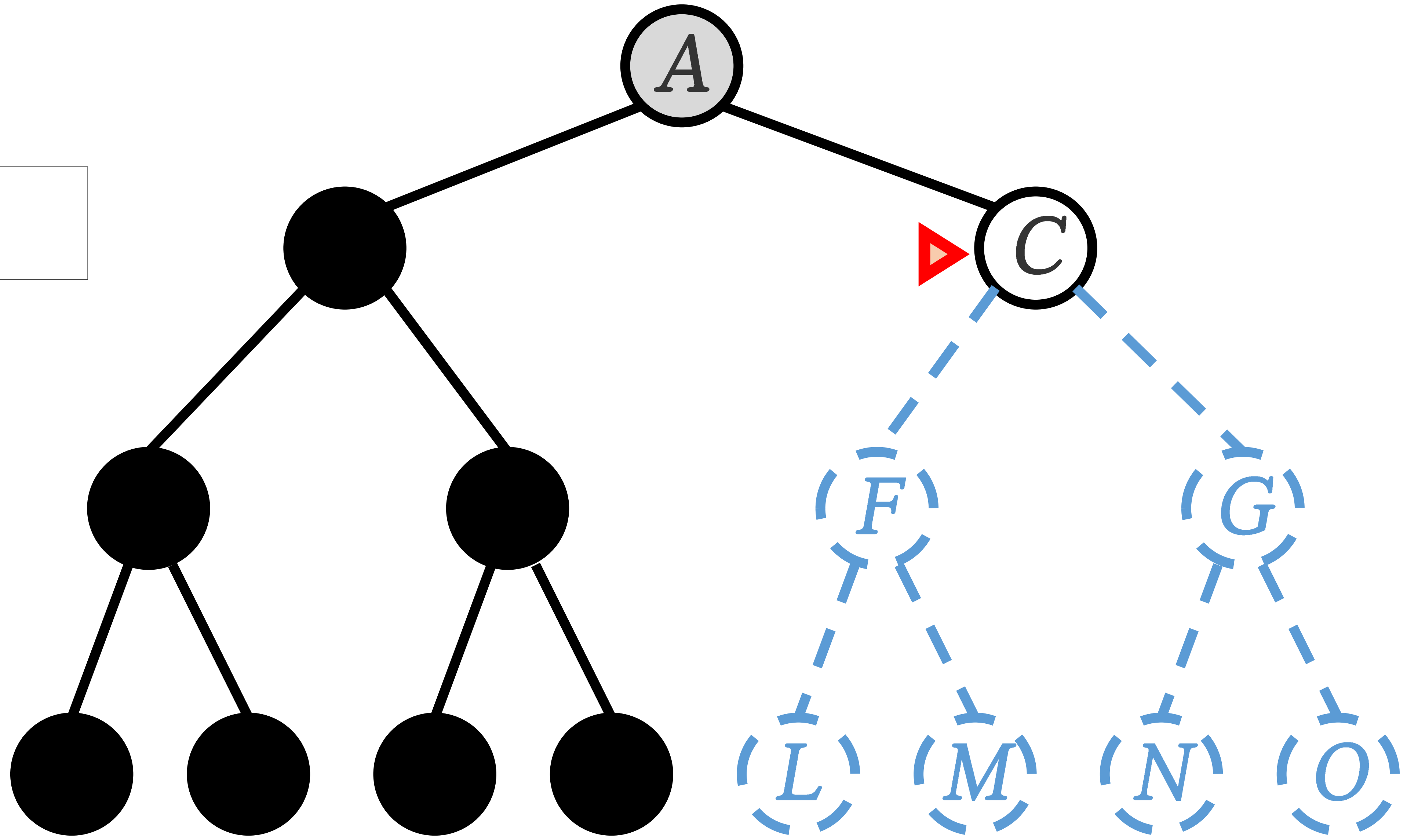
MEMO

Depth-First search

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

LIFO queue

C				



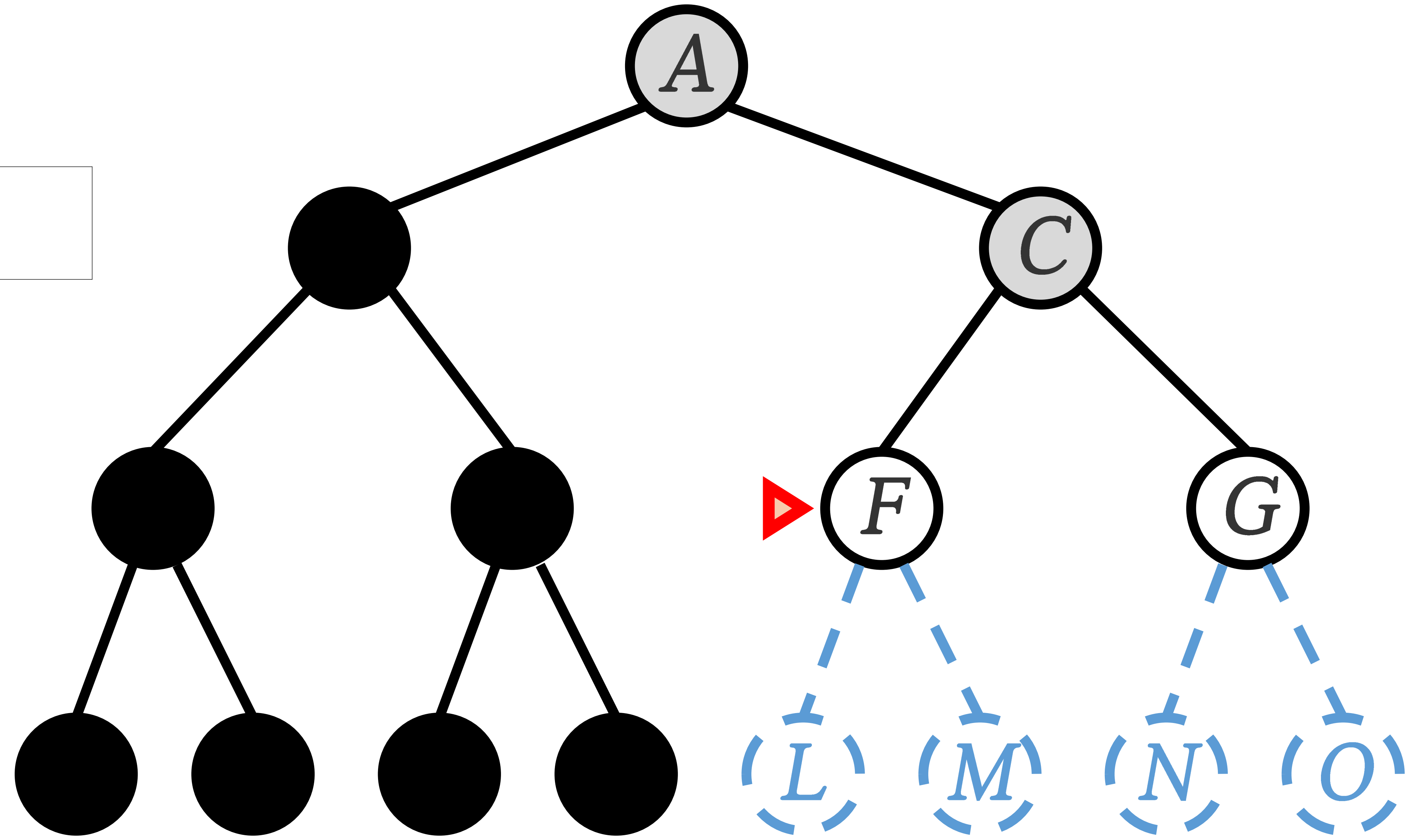
MEMO

Depth-First search

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

LIFO queue

F	G			



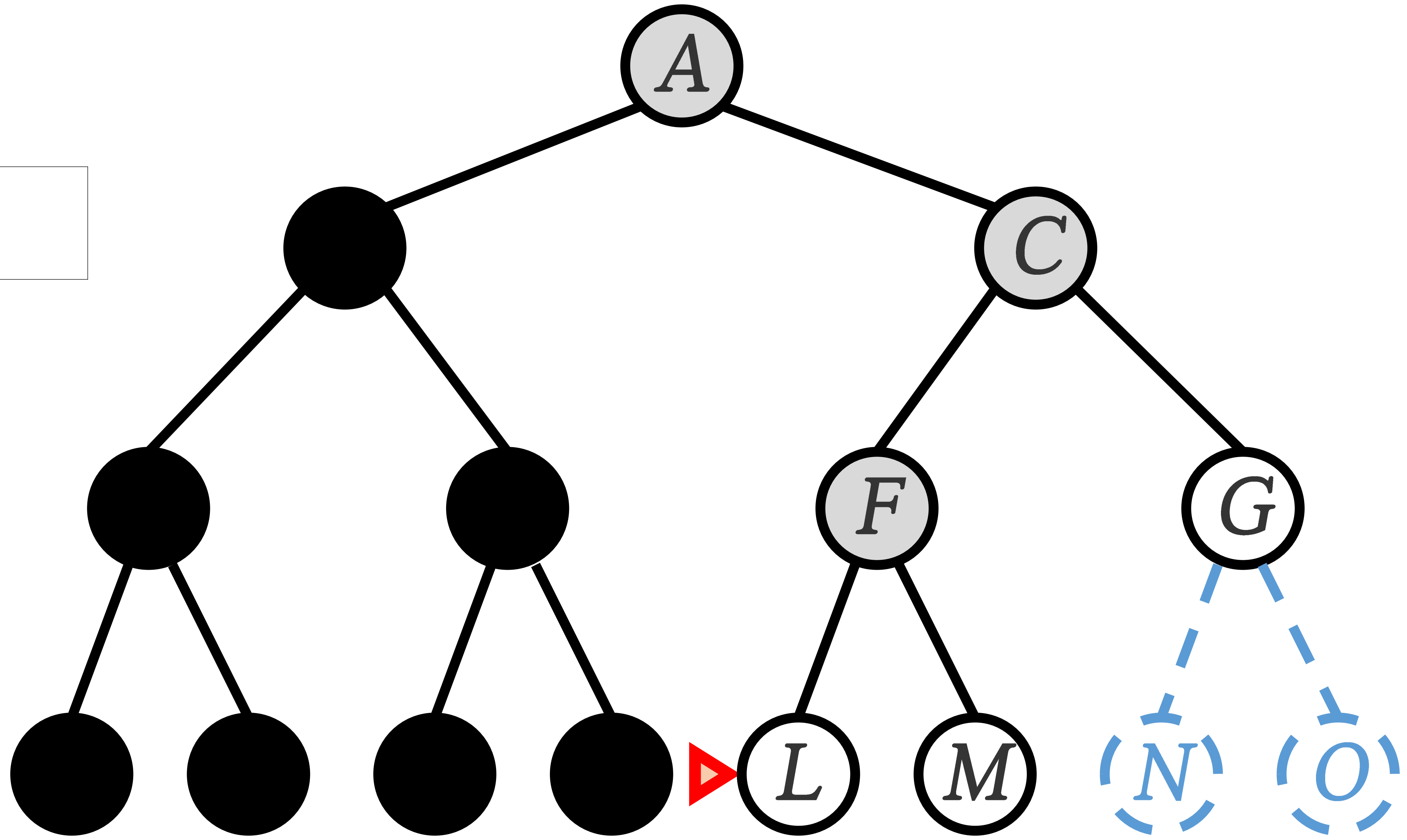
MEMO

Depth-First search

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

LIFO queue

L	M	G		



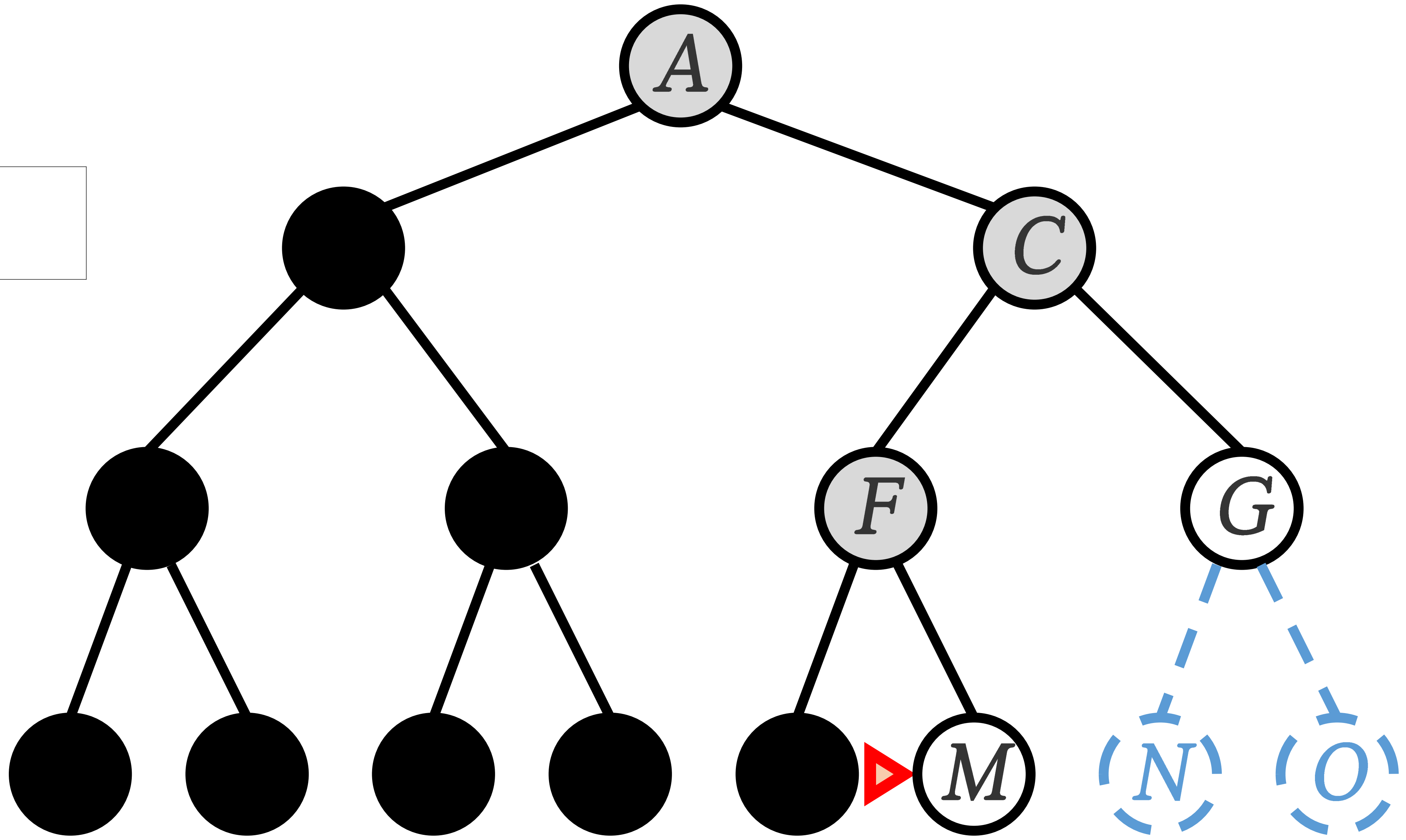
MEMO

Depth-First search

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

LIFO queue

M	G			



MEMO

Summary of BFS and DFS

Comparison

Criterion	BFS	DFS
Complete?	Yes	No
Time	$O(b^{d+1})$	$O(b^m)$
Space	$O(b^{d+1})$	$O(bm)$
Optimal?	Yes	No

- Breadth-first search is complete but expensive
- Depth-first search is cheap but incomplete

Can’t we do better than this?

MEMO