# COOPERATIVE AND ADAPTIVE ALGORITHMS

WEEK 2

Otman A. Al-Basir, Spring 2016

# TYPES OF SEARCH

Uninformed Search

- Only has the information provided by the problem formulation (initial state, set of actions, goal test, cost)

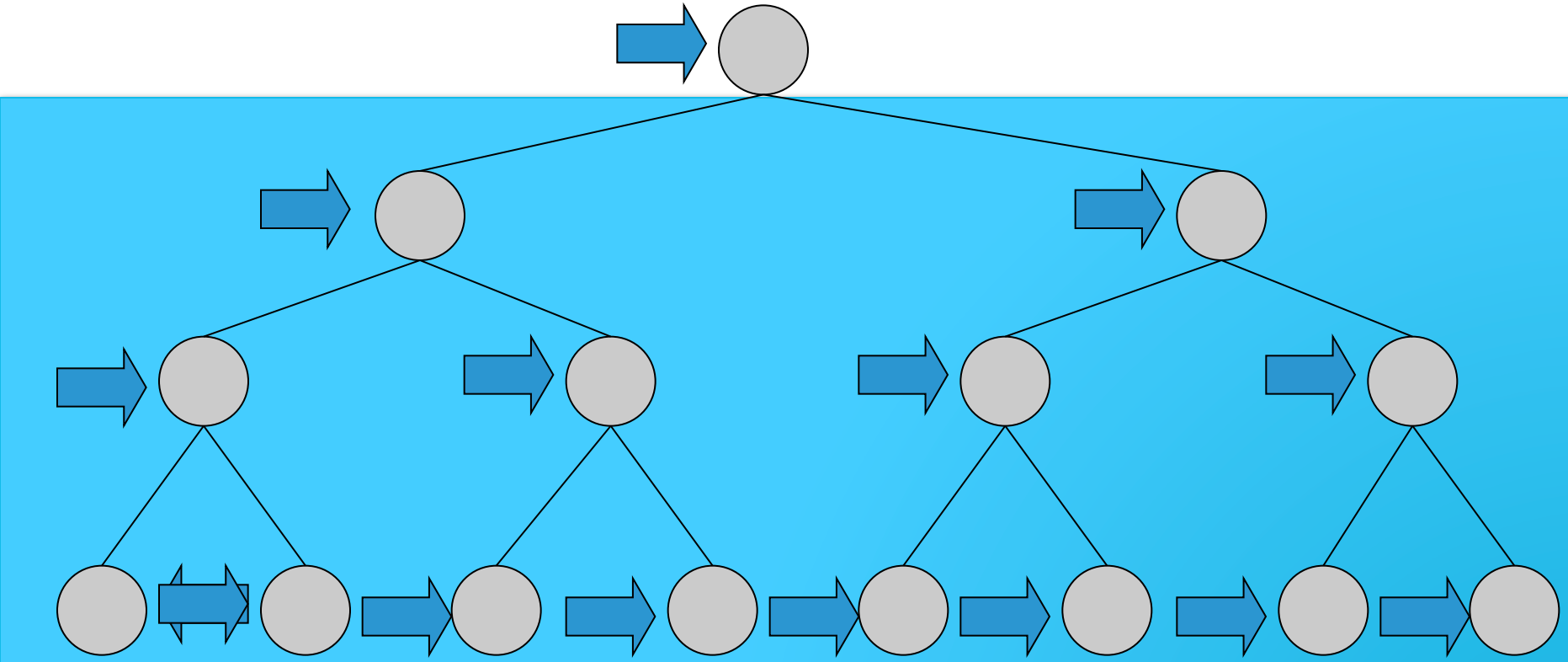
Informed Search

- Has additional information that allows it to judge the promise of an action, i.e. the estimated cost from a state to a goal

# UNINFORMED SEARCH

- Uninformed: While searching you have no clue whether one non-goal state is better than any other. Your search is blind. You don't know if your current exploration is likely to be fruitful.
- Various blind strategies:
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Iterative deepening search

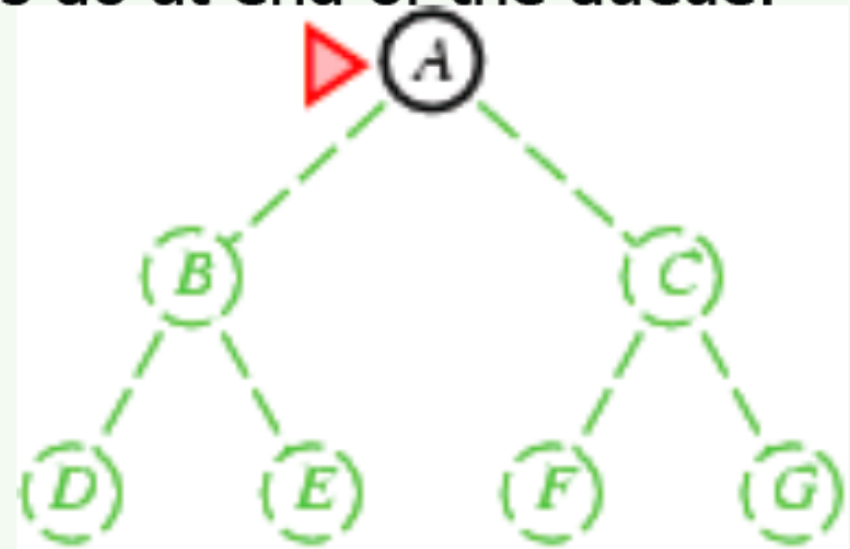
# BREATH-FIRST SEARCH



# STRATEGY

- Expand shallowest unexpanded node
- Fringe: nodes waiting in a queue to be explored
- Implementation:
  - *fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

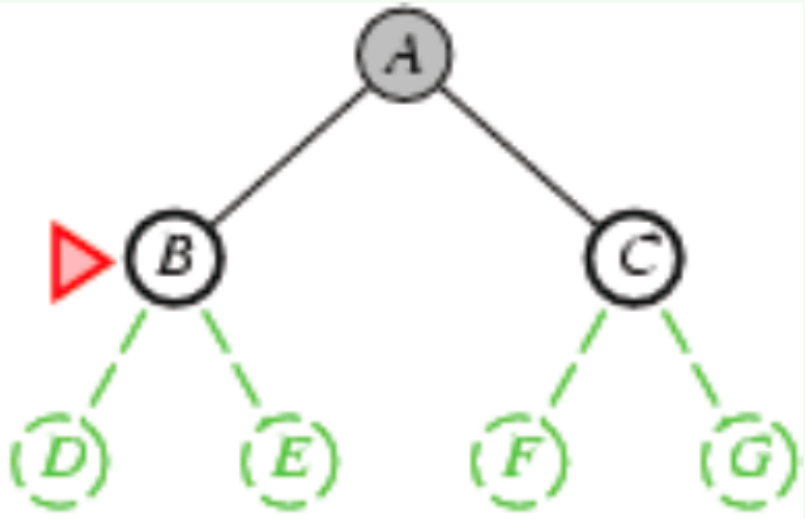
Is A a goal state?



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

Expand:
fringe = [B,C]

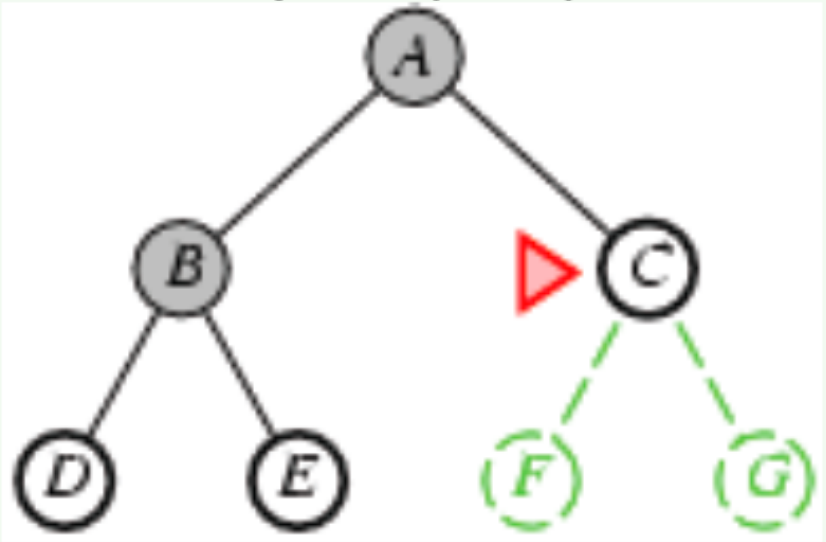
Is B a goal state?



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

Expand:
fringe=[C,D,E]

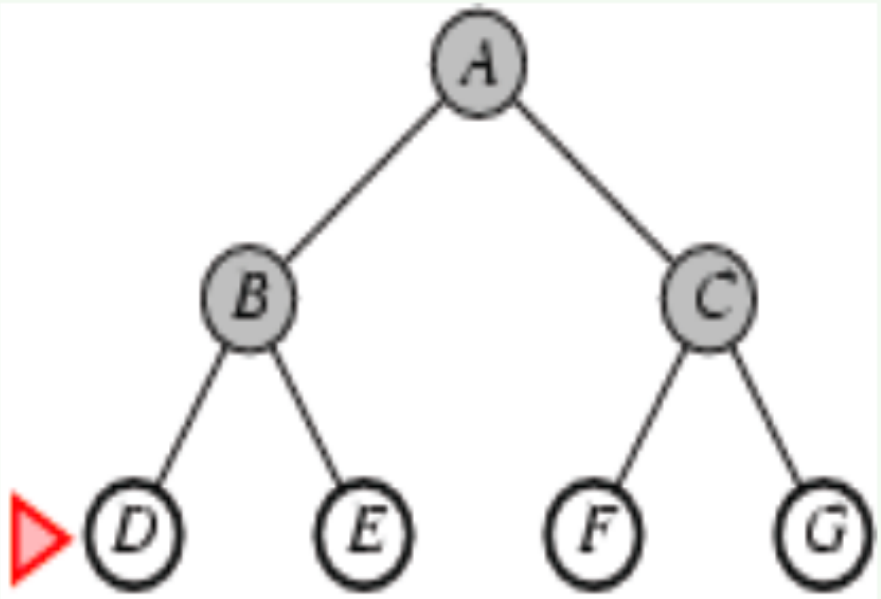
Is C a goal state?



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

Expand:
fringe=[D,E,F,G]

Is D a goal state?



# BREATH-FIRST SEARCH

Complete, if b is finite

Optimal, if path cost is equal to depth

- Guaranteed to return the shallowest goal (depth d)

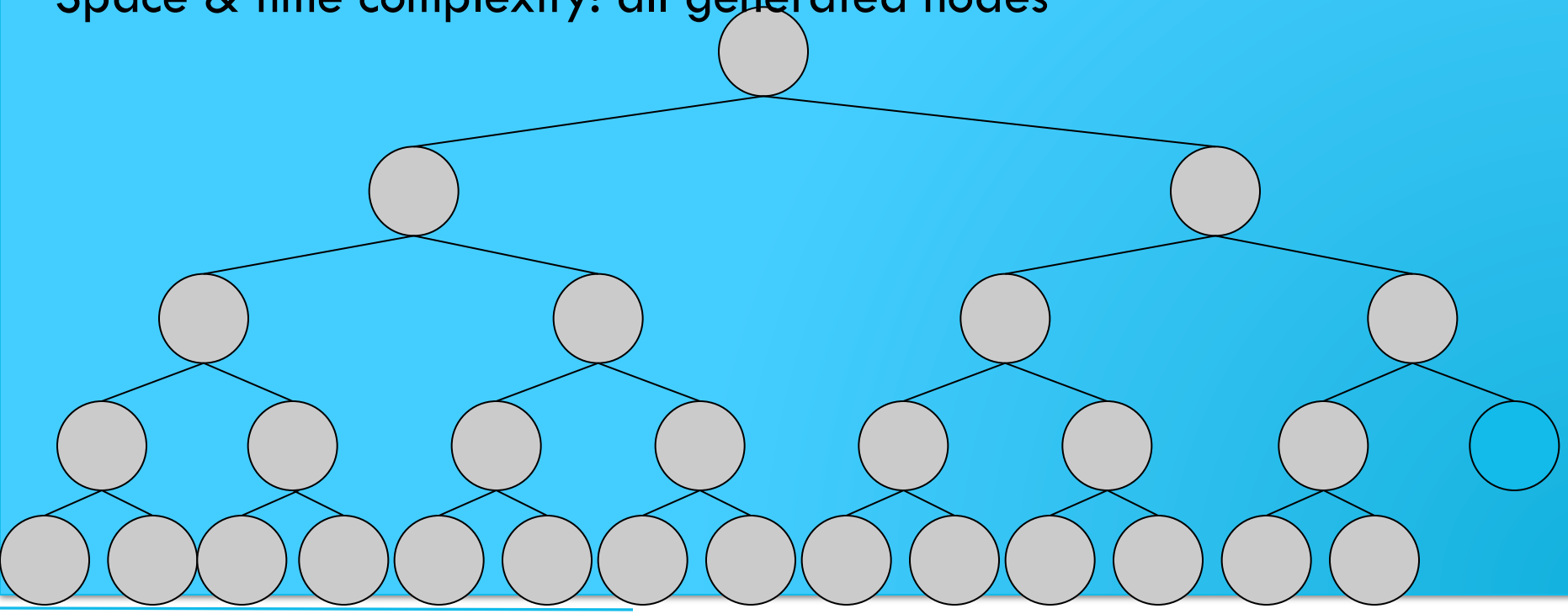
Time complexity = $O(b^{d+1})$

Space complexity = $O(b^{d+1})$

# BREATH-FIRST SEARCH

Upper-bound case: goal is last node of depth d

- Number of generated nodes:
  $1+b+b^2+b^3+\ldots+b^d+(b^{d+1}-b) = O(b^{d+1})$
- Space & time complexity: all generated nodes



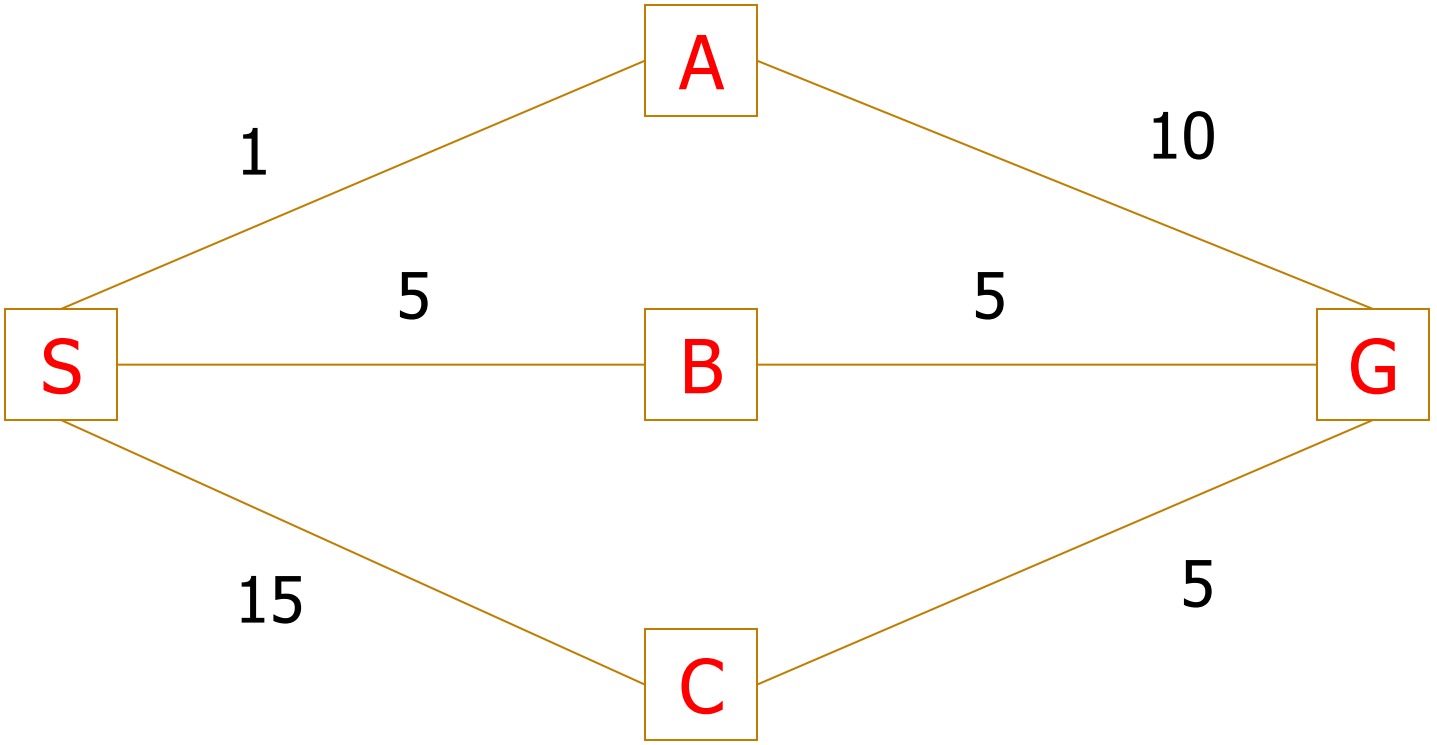
# UNIFORM COST SEARCH

A breadth-first search finds the shallowest goal state and will therefore be the cheapest solution provided the *path cost is a function of the depth of the solution*. But, if this is not the case, then breadth-first search is not guaranteed to find the best (i.e. cheapest solution).

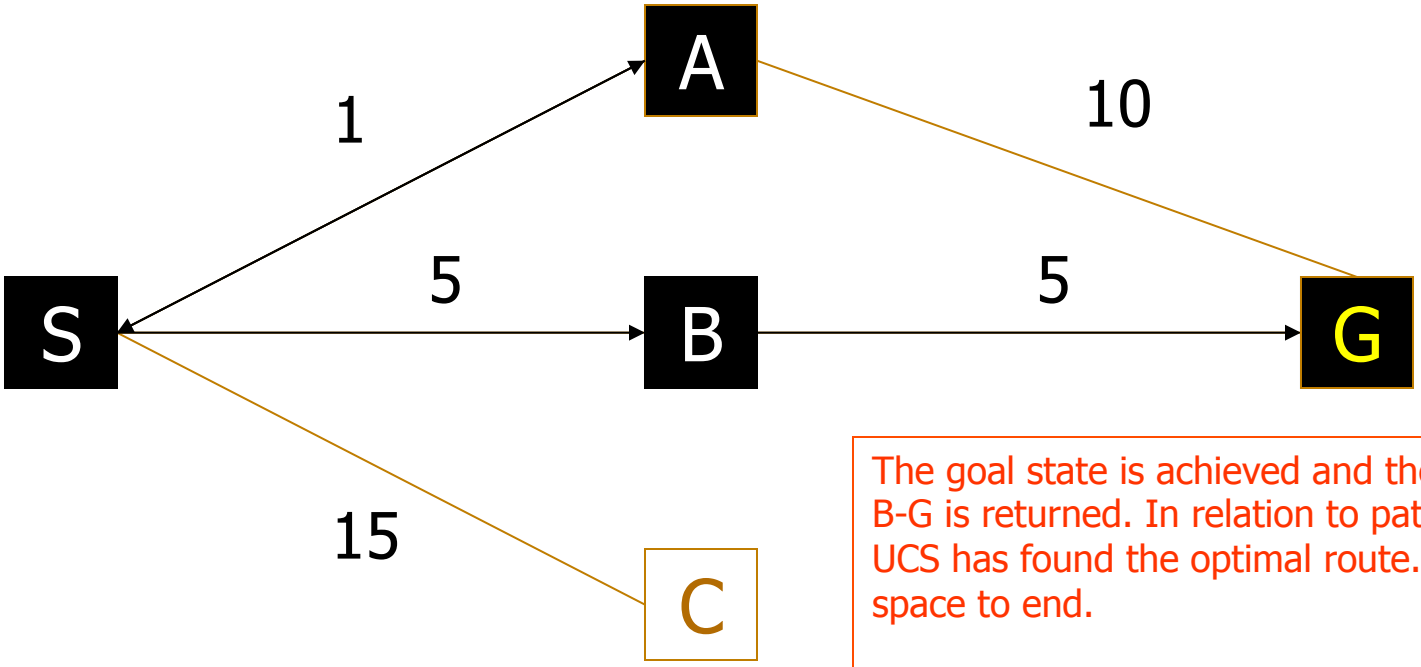
Uniform cost search remedies this by expanding the lowest cost node on the fringe, where cost is the path cost, $g(n)$.

In the following slides those values that are attached to paths are the cost of using that path.

Consider the following problem…




We wish to find the shortest route from node S to node G; that is, node S is the initial state and node G is the goal state. In terms of path cost, we can clearly see that the route *SBG* is the cheapest route. However, if we let breadth-first search loose on the problem it will find the non-optimal path *SAG*, assuming that A is the first node to be expanded at level 1. Press space to see a UCS of the same node set…




The goal state is achieved and the path S-B-G is returned. In relation to path cost, UCS has found the optimal route. Press space to end.

**Press space to begin the search**

| | | |
|---|---|---|
| **Size of Queue: 0** | **Queue: Empty** | |
| **Nodes expanded: 3** | **FINISHED SEARCH** | **Current level: 2** |

**UNIFORM COST SEARCH PATTERN**

# UNIFORM-COST SEARCH

Expansion of Breath-First Search

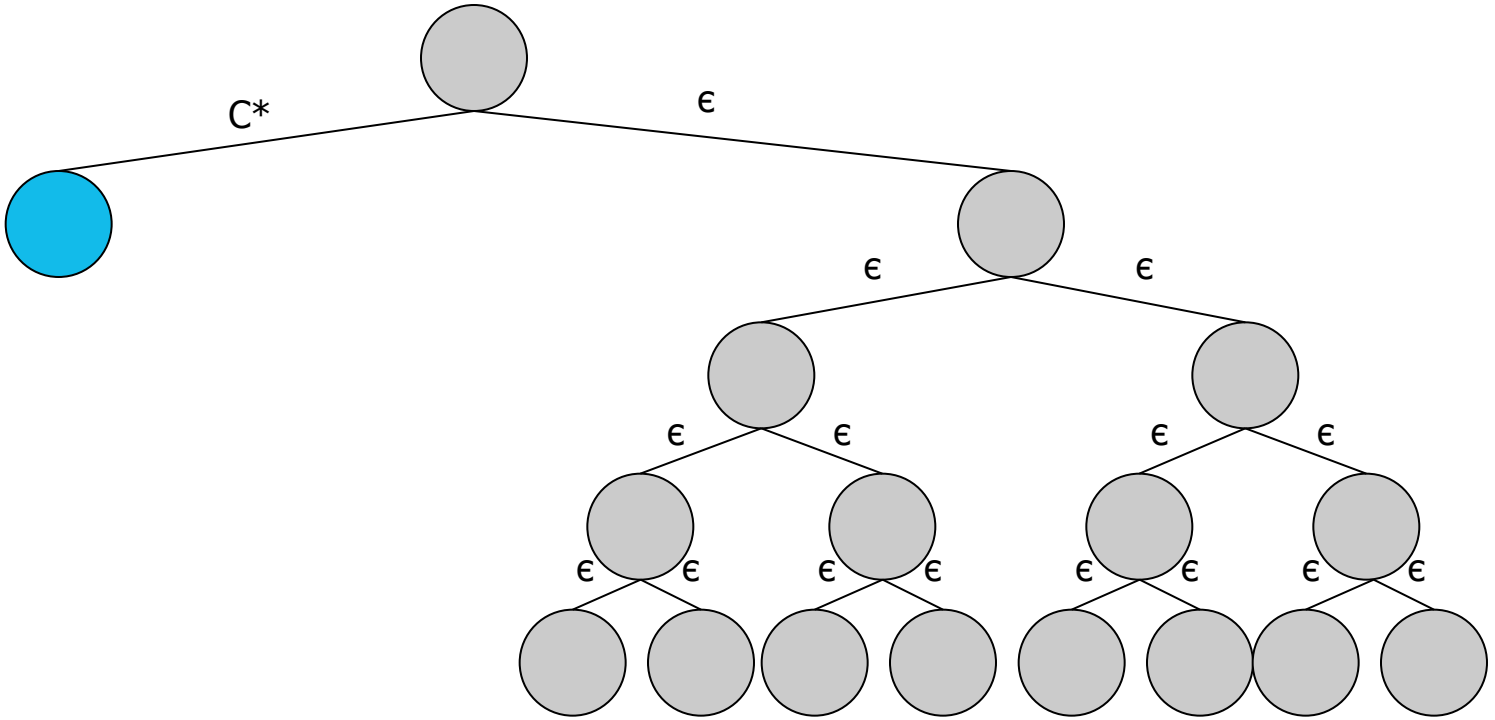
Explore the cheapest node first (in terms of path cost)

Condition: No zero-cost or negative-cost edges.

- Minimum cost is $\epsilon$

Breath-First Search is Uniform-Cost Search with constant-cost edges

# UNIFORM-COST SEARCH




Upper-bound case: goal has path cost C*, all other actions have minimum cost of ϵ

- Depth explored before taking action C*: $C^*/\epsilon$
- Depth of fringe nodes: $C^*/\epsilon + 1$
- Space & time complexity: all generated nodes: $O(b^{C^*/\epsilon+1})$

# UNIFORM-COST SEARCH

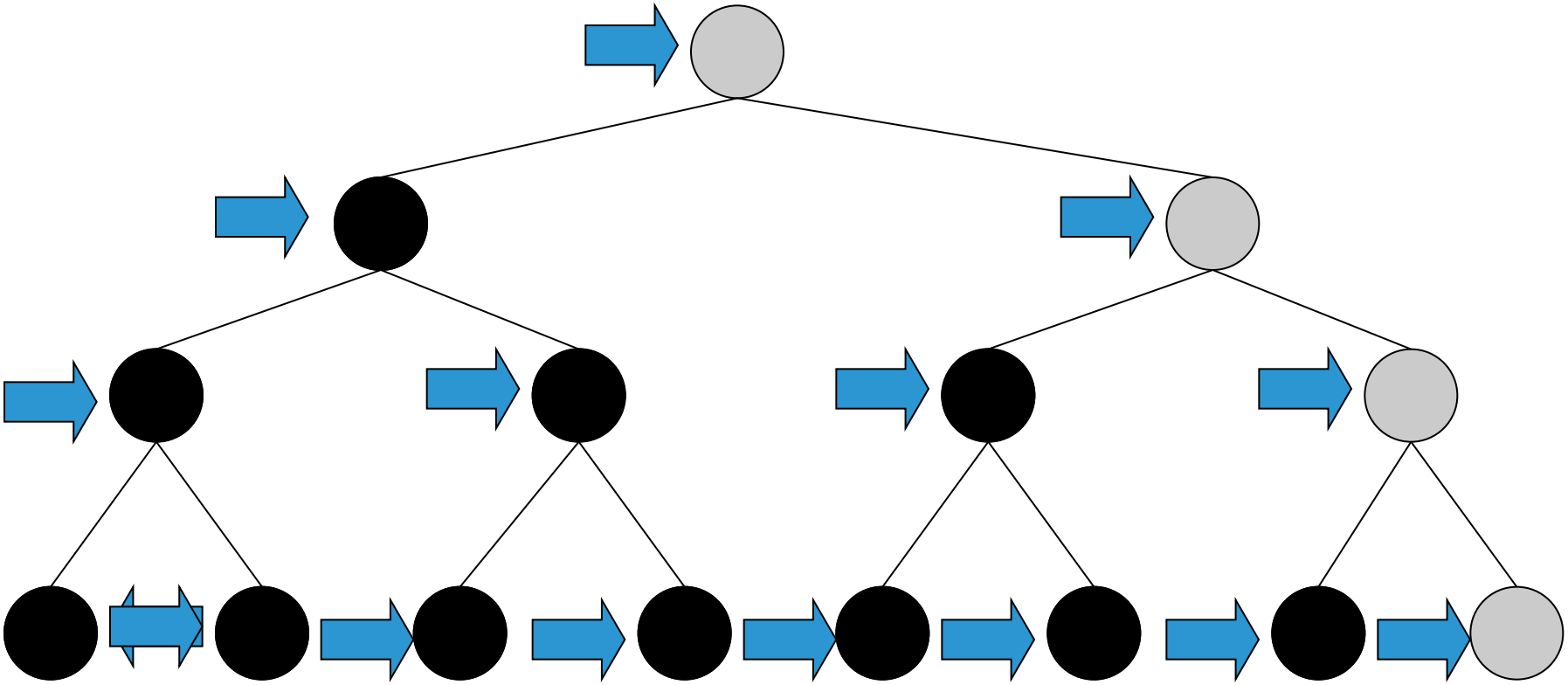
Complete given a finite tree

Optimal

Time complexity = $O(b^{C^*/\epsilon+1}) \geq O(b^{d+1})$ ← under equal steps

Space complexity = $O(b^{C^*/\epsilon+1}) \geq O(b^{d+1})$

# DEPTH-FIRST SEARCH



# DEPTH FIRST SEARCH

Expand deepest unexpanded node

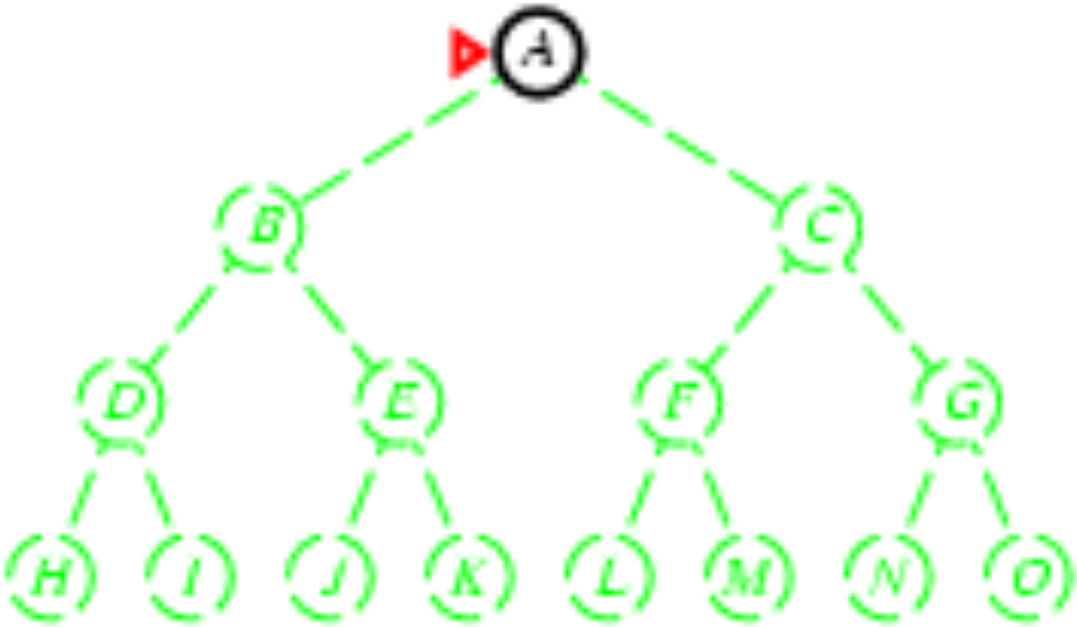
- The root is examined first; then the left child of the root; then the left child of this node, etc. until a leaf is found. At a leaf, backtrack to the lowest right child and repeat.

Does not have to keep all nodes on the open list, only retains the children of a single state

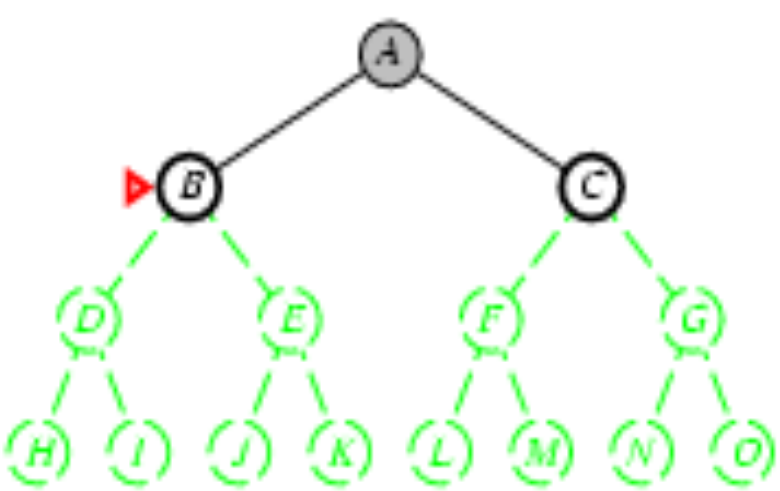
May get stuck in a deep search space

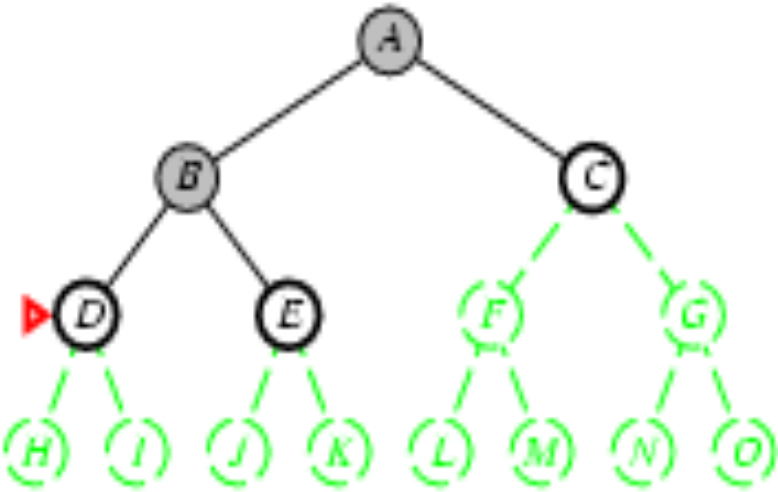
- *Implementation :*
- Fringe = LIFO queue, i.e., put successors at front

# DEPTH-FIRST SEARCH

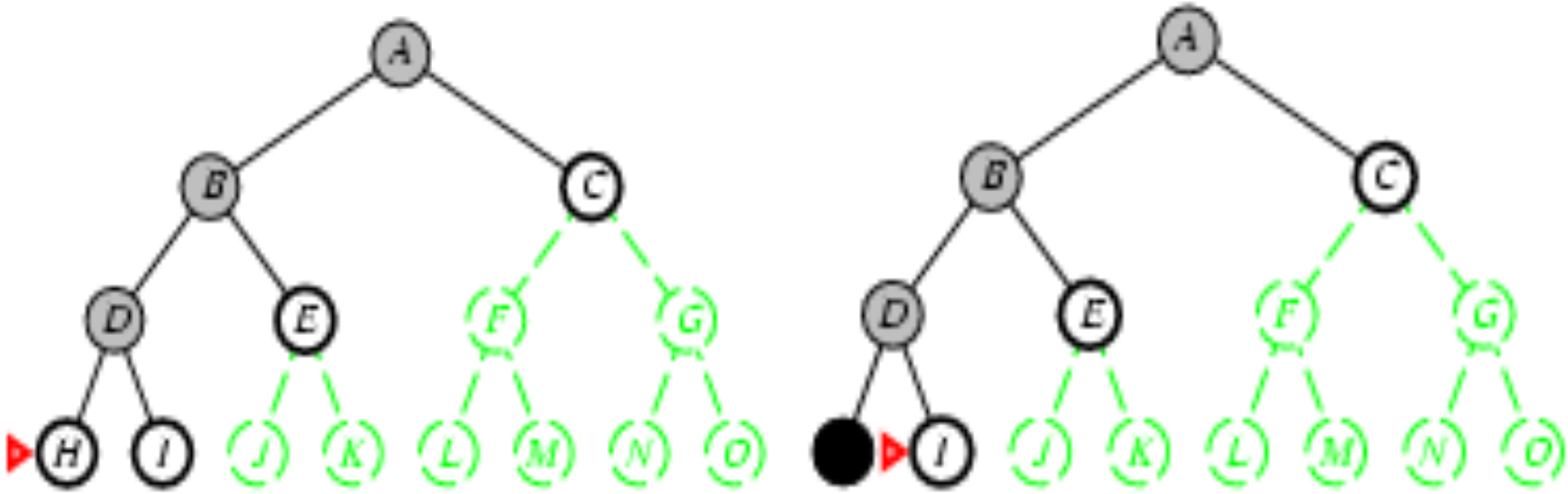


# DEPTH-FIRST SEARCH

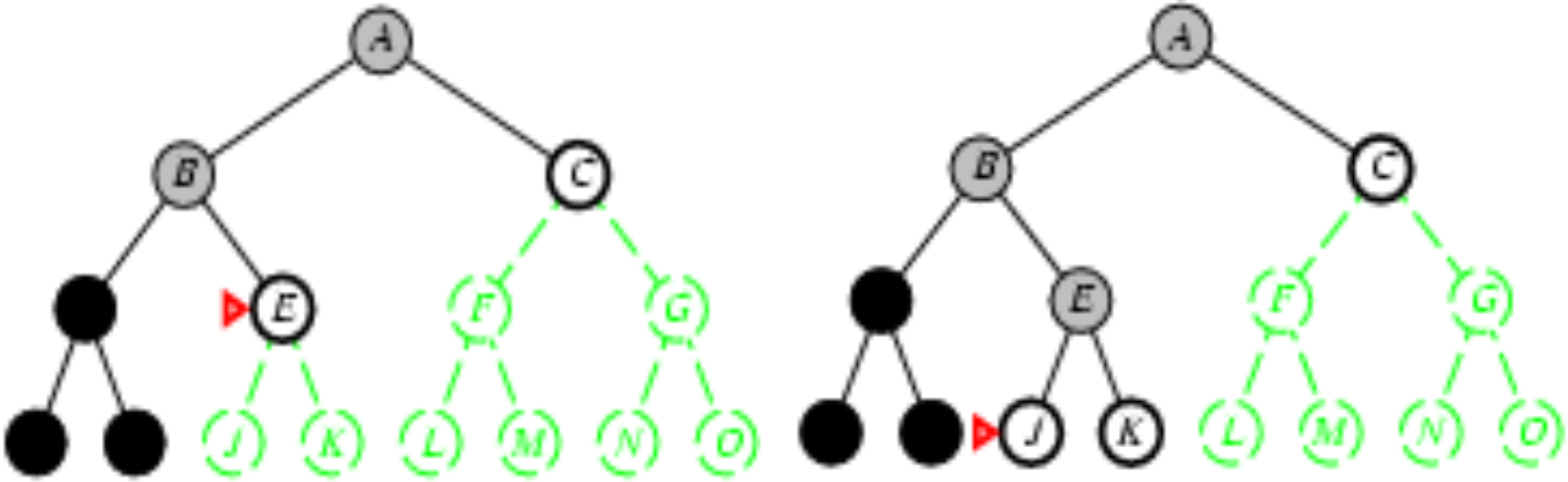


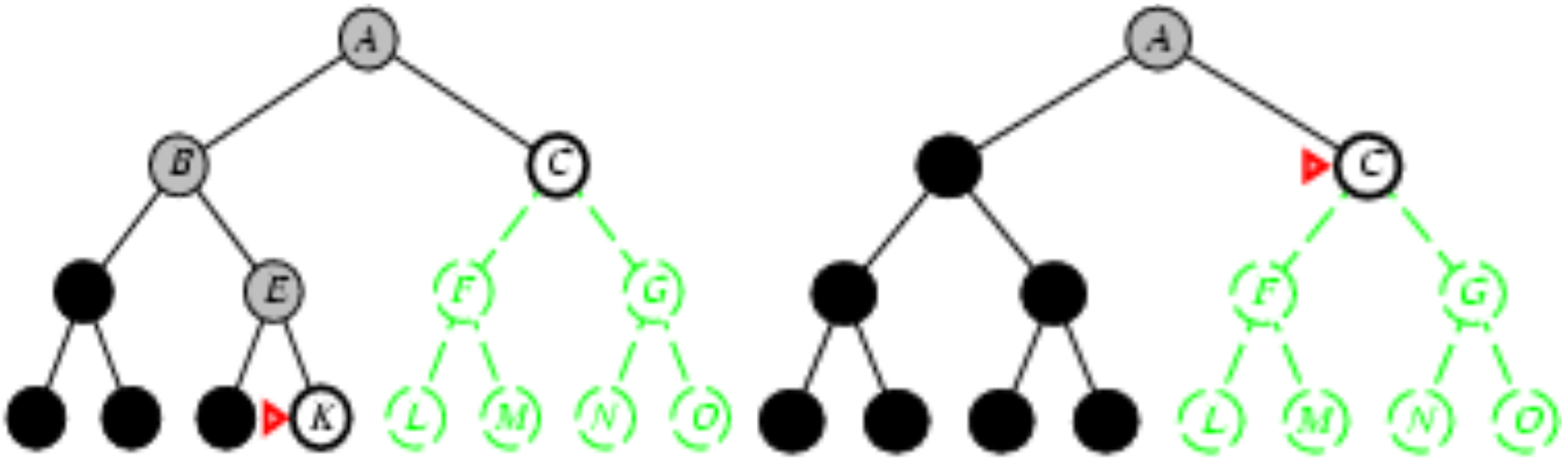
# DEPTH-FIRST SEARCH



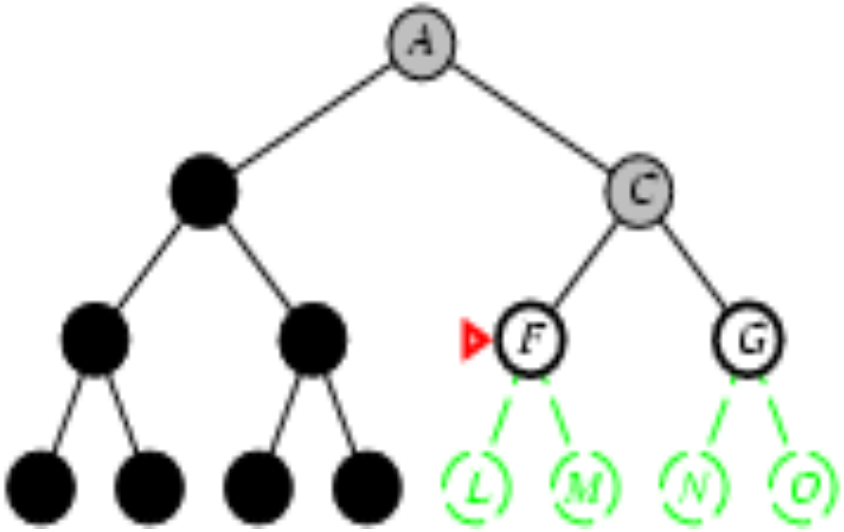
# DEPTH-FIRST SEARCH

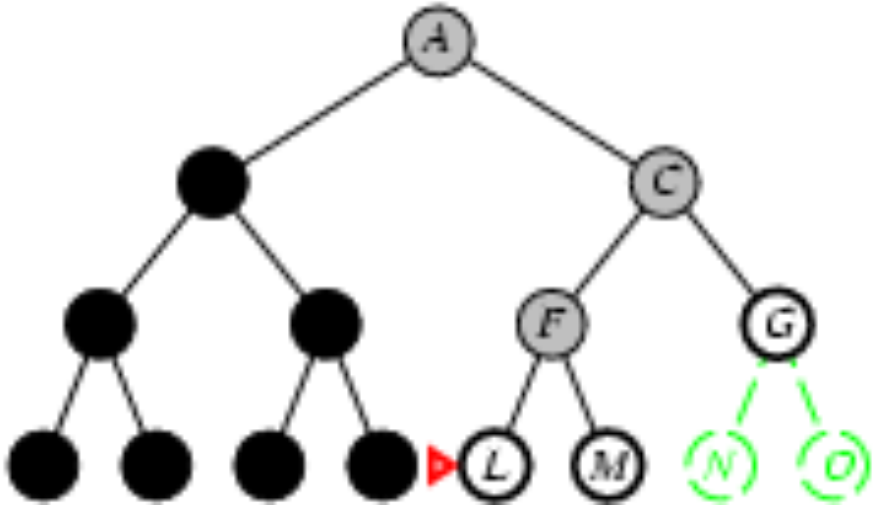


# DEPTH-FIRST SEARCH

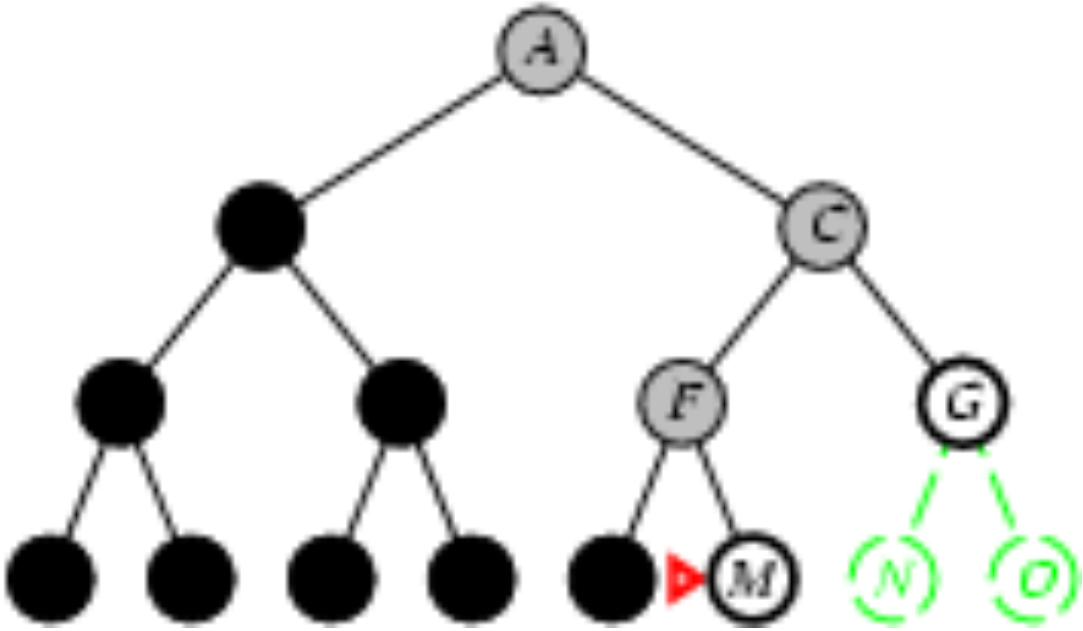


# DEPTH-FIRST SEARCH



# DEPTH-FIRST SEARCH



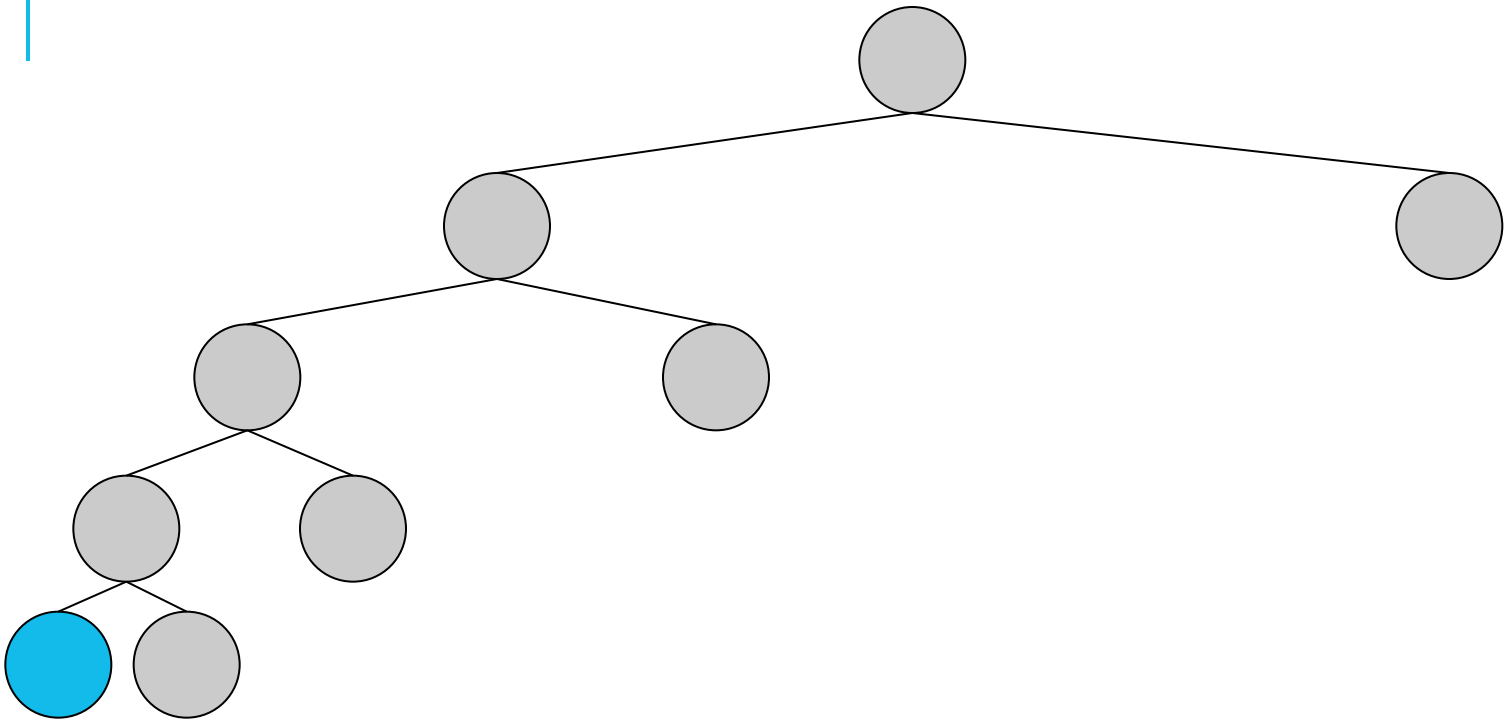
# DEPTH-FIRST SEARCH

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - → complete in finite spaces
  - → Complete only if m is finite
- Time? $O(b^m)$: terrible if *m* is much larger than *d* *where m is maximum depth of any node,*

  *d is depth of the least-cost solution*
  - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space!

  *where b is maximum branching factor of the search tree*
- Optimal? No

# DEPTH-FIRST SEARCH



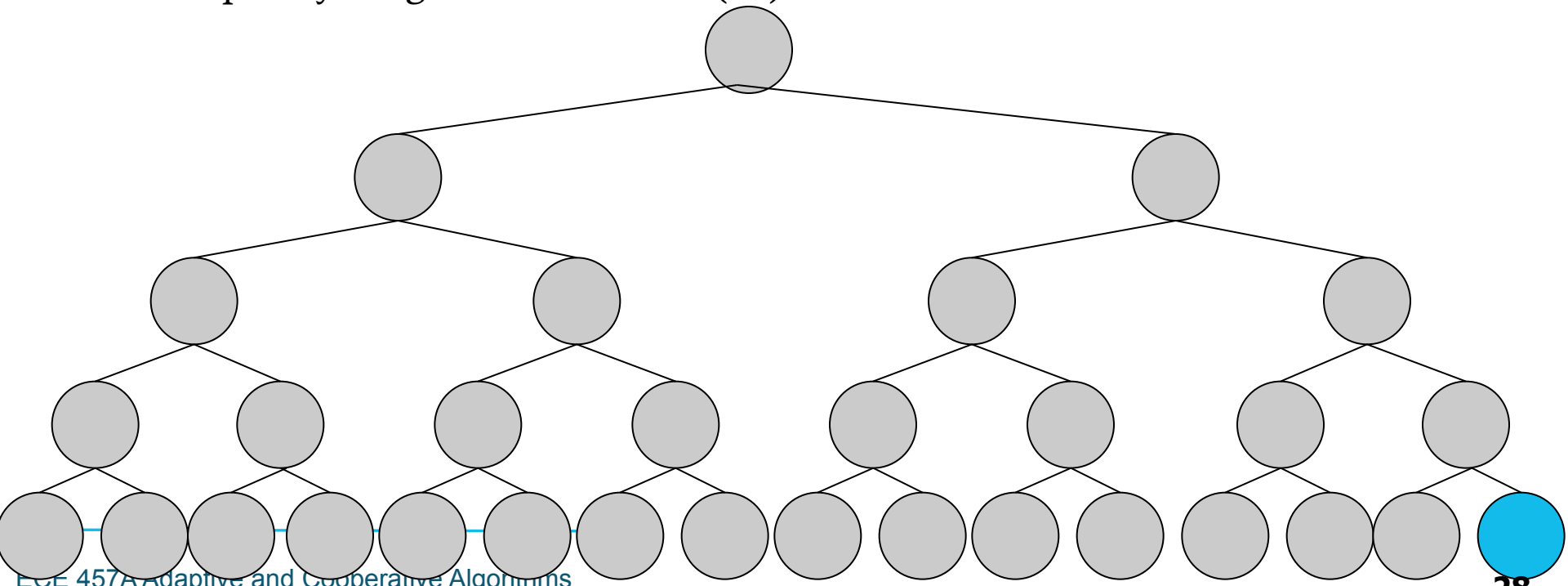
Upper-bound case for space: goal is last node of first branch

- After that, we start deleting nodes
- Number of generated nodes: b nodes at each of m levels
- Space complexity: all generated nodes = O(bm)

# DEPTH-FIRST SEARCH

Upper-bound case for time: goal is last node of last branch

- Number of nodes generated:
  b nodes for each node of m levels (entire tree)
- Time complexity: all generated nodes $O(b^m)$



# DEPTH-LIMITED SEARCH

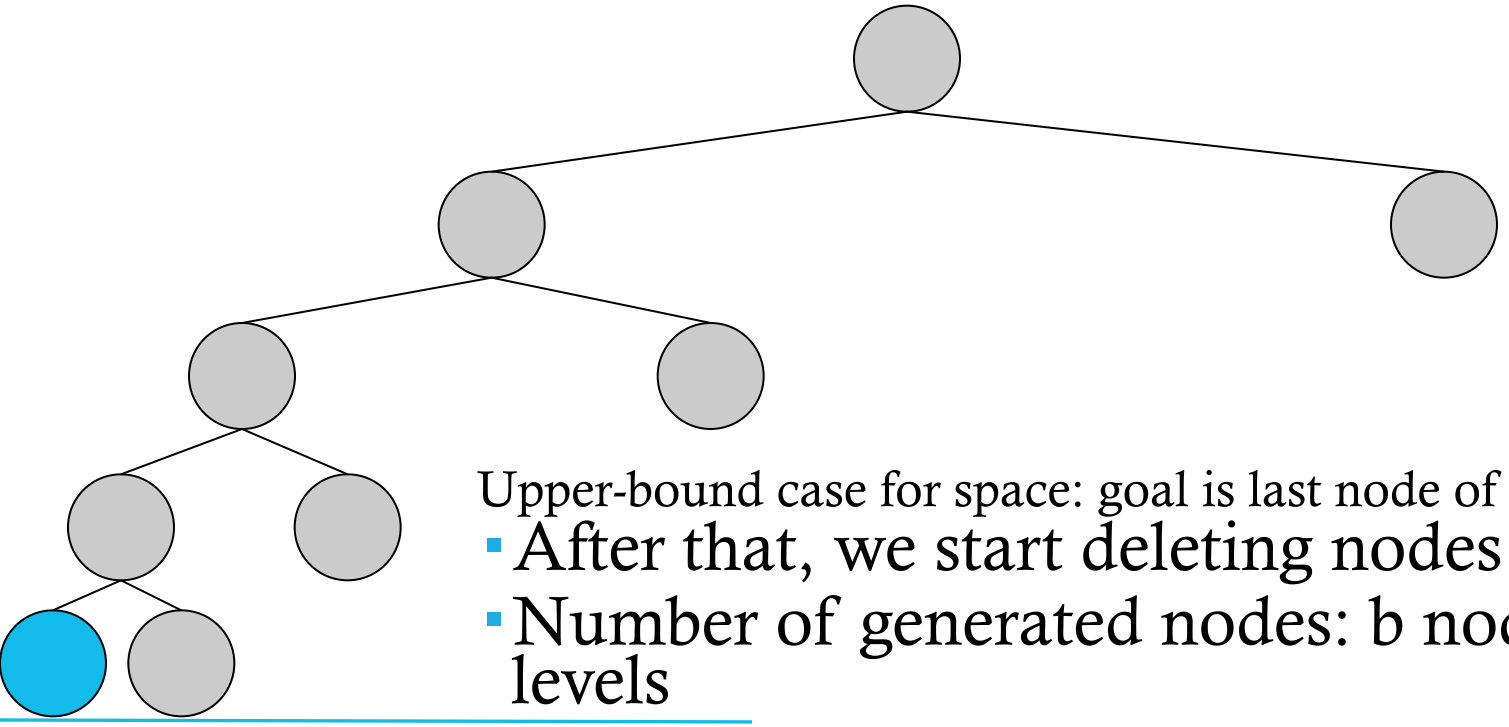
Like the normal depth-first search, depth-limited search is an uninformed search.

It works exactly like depth-first search, but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search.

Even if the search could still expand a vertex beyond that depth, it will not do so and thereby it will not follow infinitely deep paths or get stuck in cycles.

Therefore depth-limited search will find a solution if it is within the depth limit, which guarantees at least completeness on all graphs

# DEPTH-LIMITED SEARCH



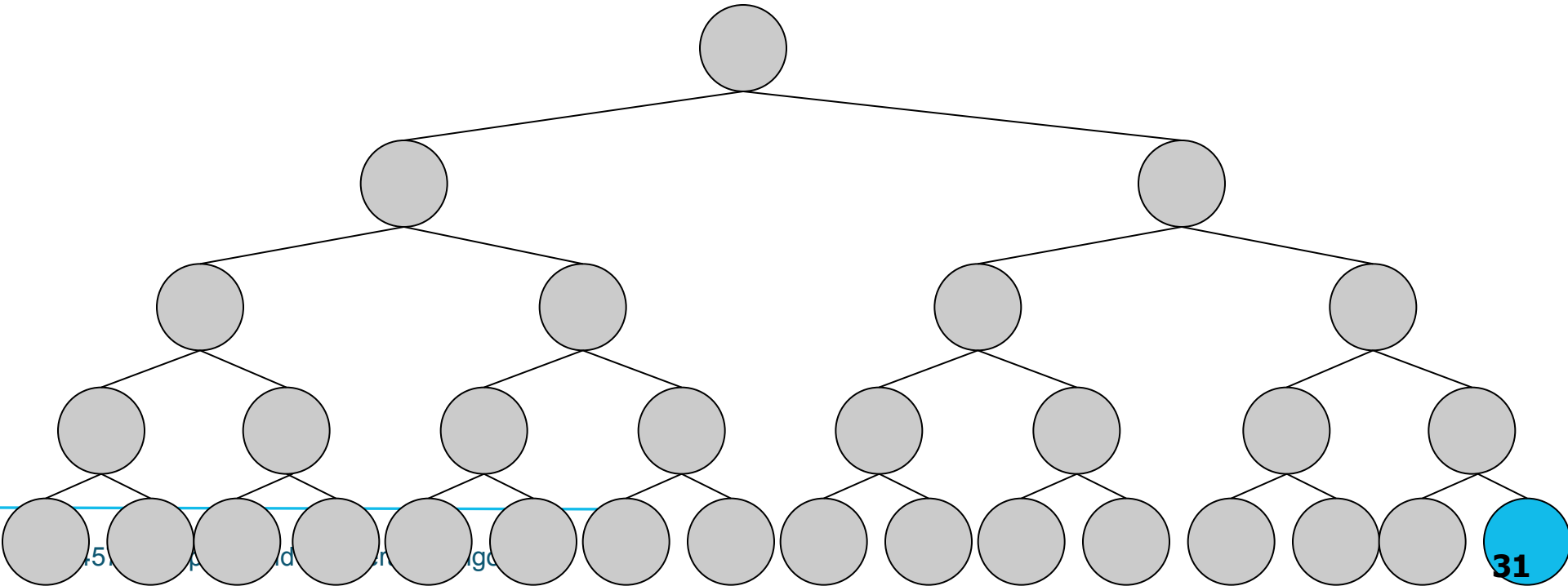
Upper-bound case for space: goal is last node of first branch

- After that, we start deleting nodes
- Number of generated nodes: b nodes at each of levels
- Space complexity: all generated nodes = O(bl)

# DEPTH-LIMITED SEARCH

Upper-bound case for time: goal is last node of last branch

- Number of nodes generated:
  b nodes for each node of l levels (entire tree to depth l)
- Time complexity: all generated nodes $O(b^l)$



# DEPTH-LIMITED SEARCH

## Properties of depth limited search

- complete if there is a solution within the depth bound $d \leq l$
- if $l < d$, i.e., the shallowest solution is deeper than the bound, DLS will return 'no' even though there is a solution
- space complexity is $O(bl)$ where $b$ is the branching factor and $l$ is the depth bound
- time complexity is $O(b^l)$
- if $l >> d$ this can significantly increase the cost of the search compared to, e.g., breadth-first search
- always terminates

# ITERATIVE DEEPENING SEARCH

The technique of iterative deepening is based on limited/ bounded search idea.

Iterative deepening is depth-first search to a fixed depth in the tree being searched.

If no solution is found up to this depth then the depth to be searched is increased and the whole `bounded' depth-first search begun again.

# ITERATIVE DEEPENING SEARCH

Complete, if b is finite

Optimal, if path cost is equal to depth

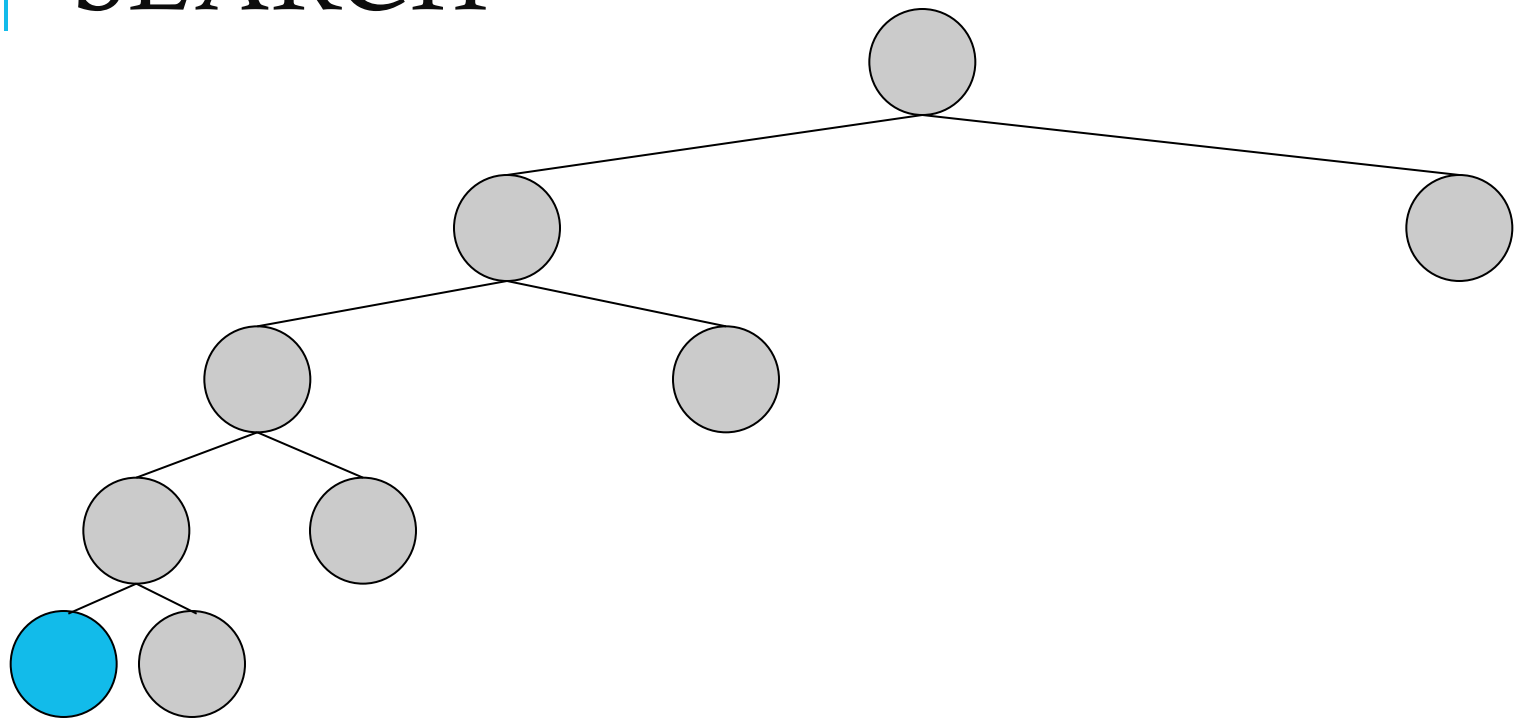
- Guaranteed to return the shallowest goal

Time complexity = $O(b^d)$

Space complexity = O(bd)

Nodes on levels above d are generated multiple times

# ITERATIVE DEEPENING SEARCH



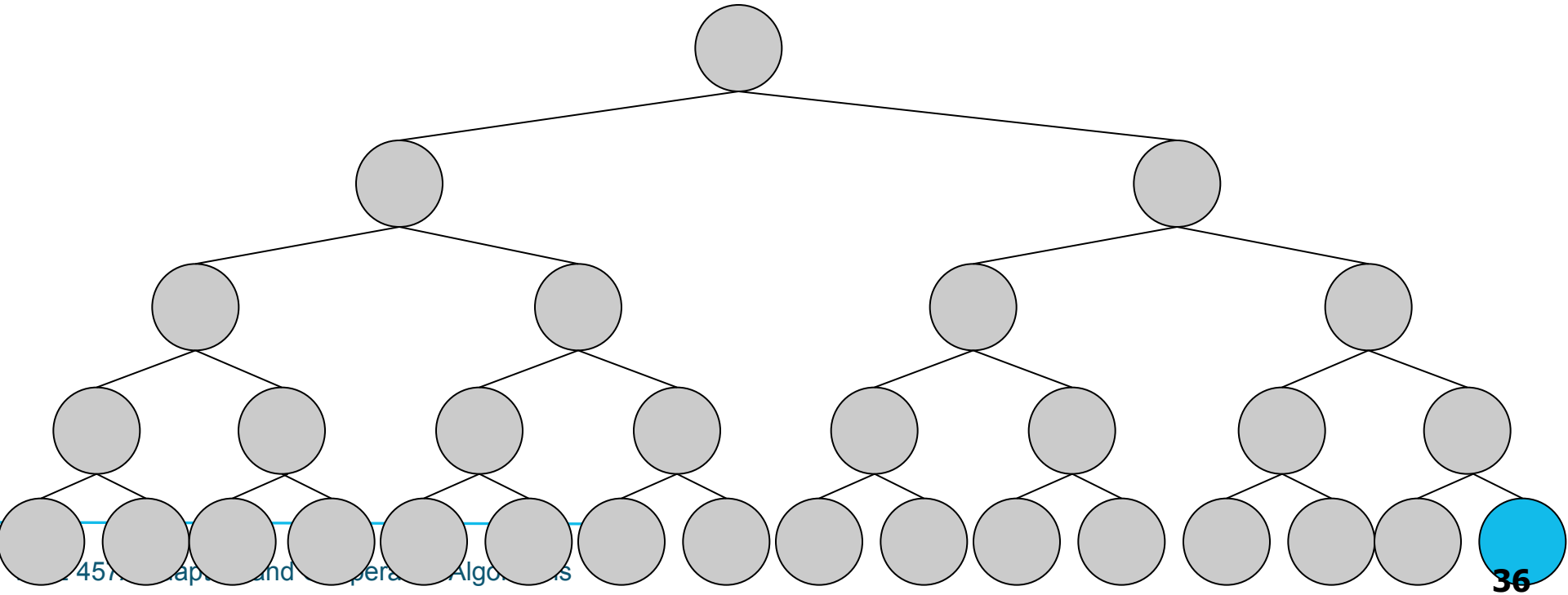
Upper-bound case for space: goal is last node of first branch

- After that, we start deleting nodes
- Number of generated nodes: b nodes at each of d levels
- Space complexity: all generated nodes = O(bd)

# ITERATIVE DEEPENING SEARCH

Upper-bound case for time: goal is last node of last branch

- Number of nodes generated:
  b nodes for each node of d levels (entire tree to depth d)
- Time complexity: all generated nodes $O(b^d)$



# COMMENTS ON DFS AND BFS

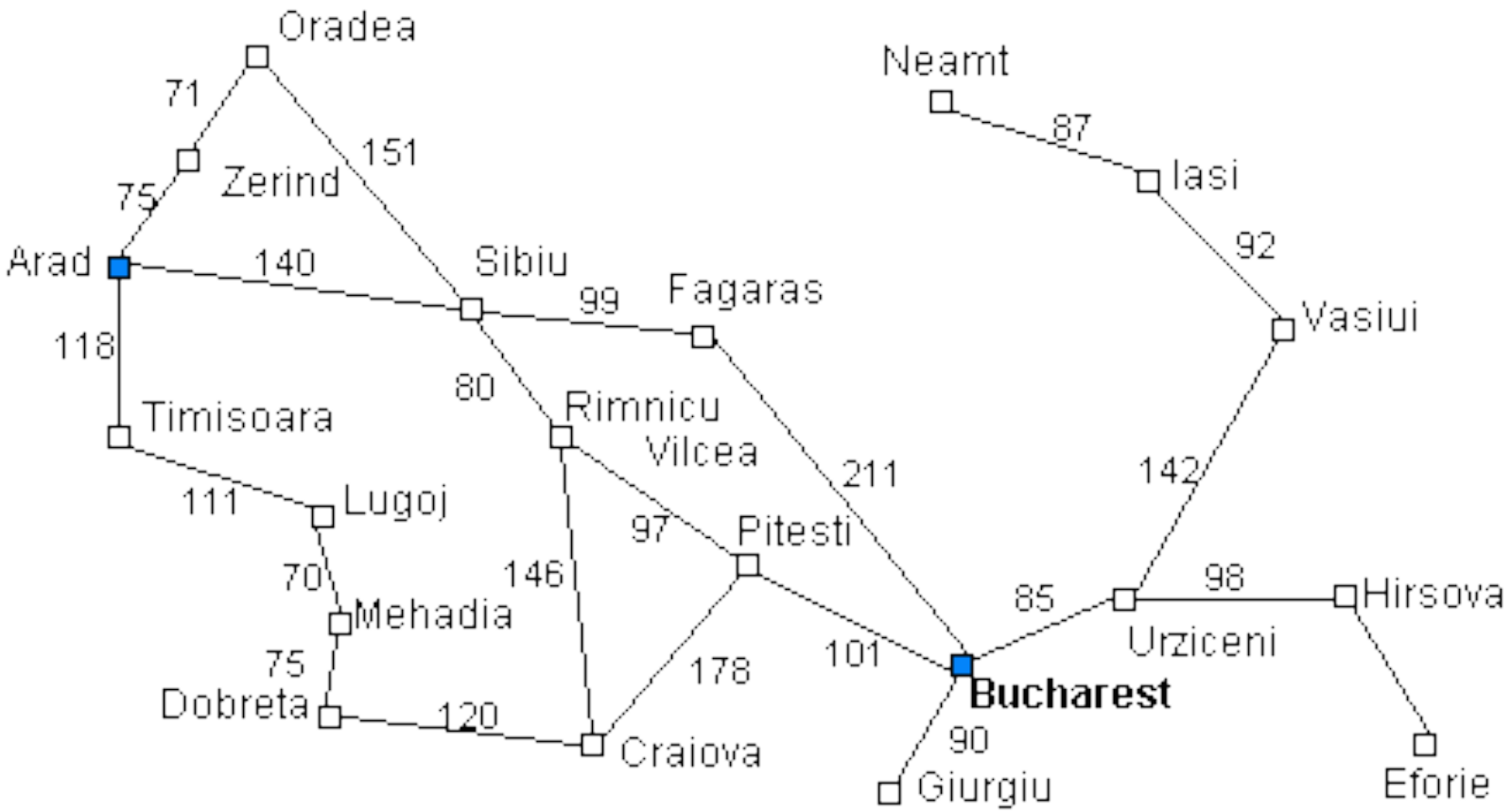
## Depth-first

- Low memory requirement.
- Could get stuck exploring infinite paths.
- Used if there are many solutions and you need only one.

## Breadth-first

- High memory requirement.
- Doesn't get stuck.
- Finds the shortest path (minimum number of steps).

# SUMMARY / EXAMPLE GOING FROM ARAD TO BUCHAREST



# SUMMARY / EXAMPLE

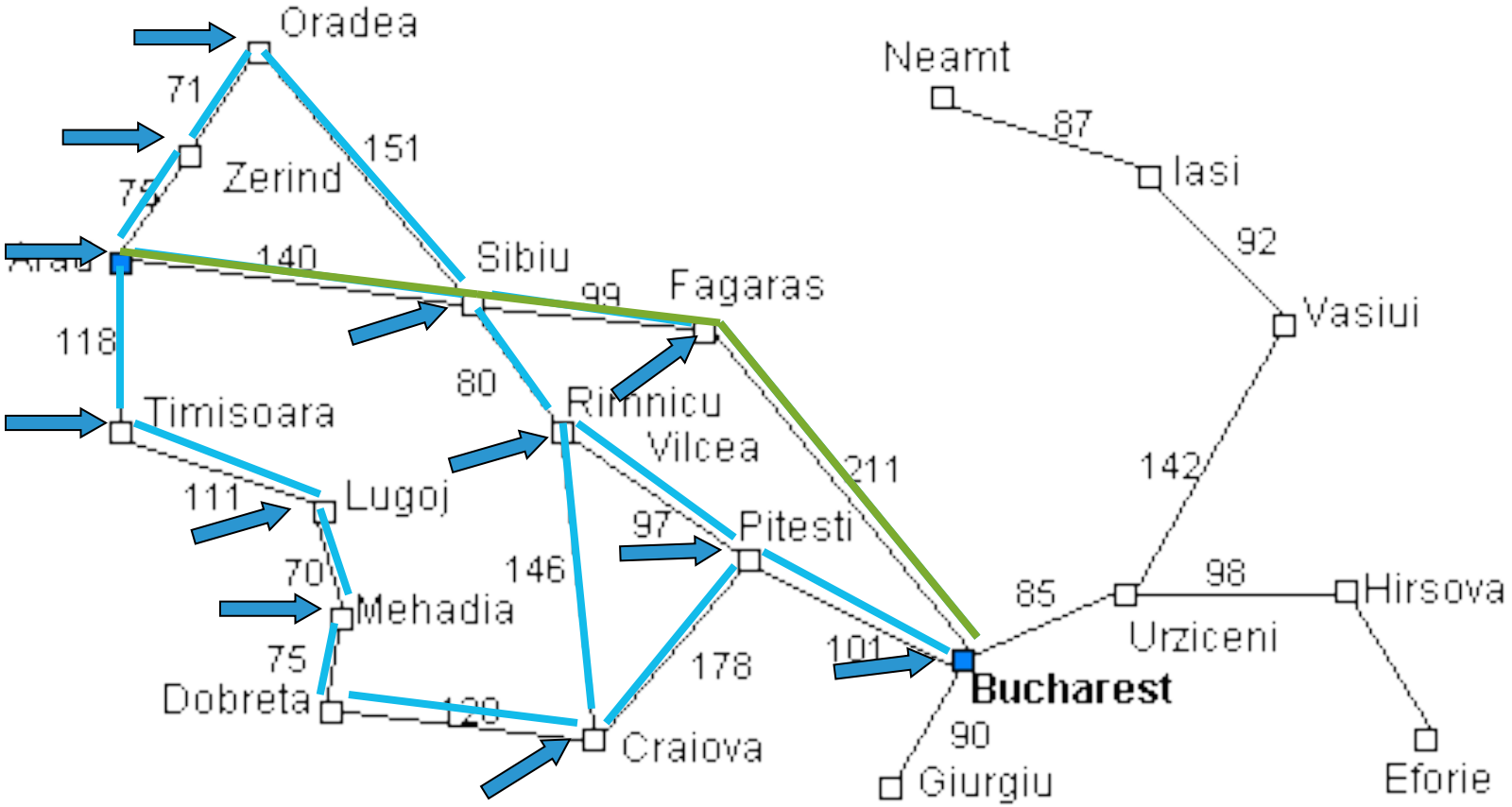
Initial state
- Being in Arad

Action
- Move to a neighbouring city, if a road exists.

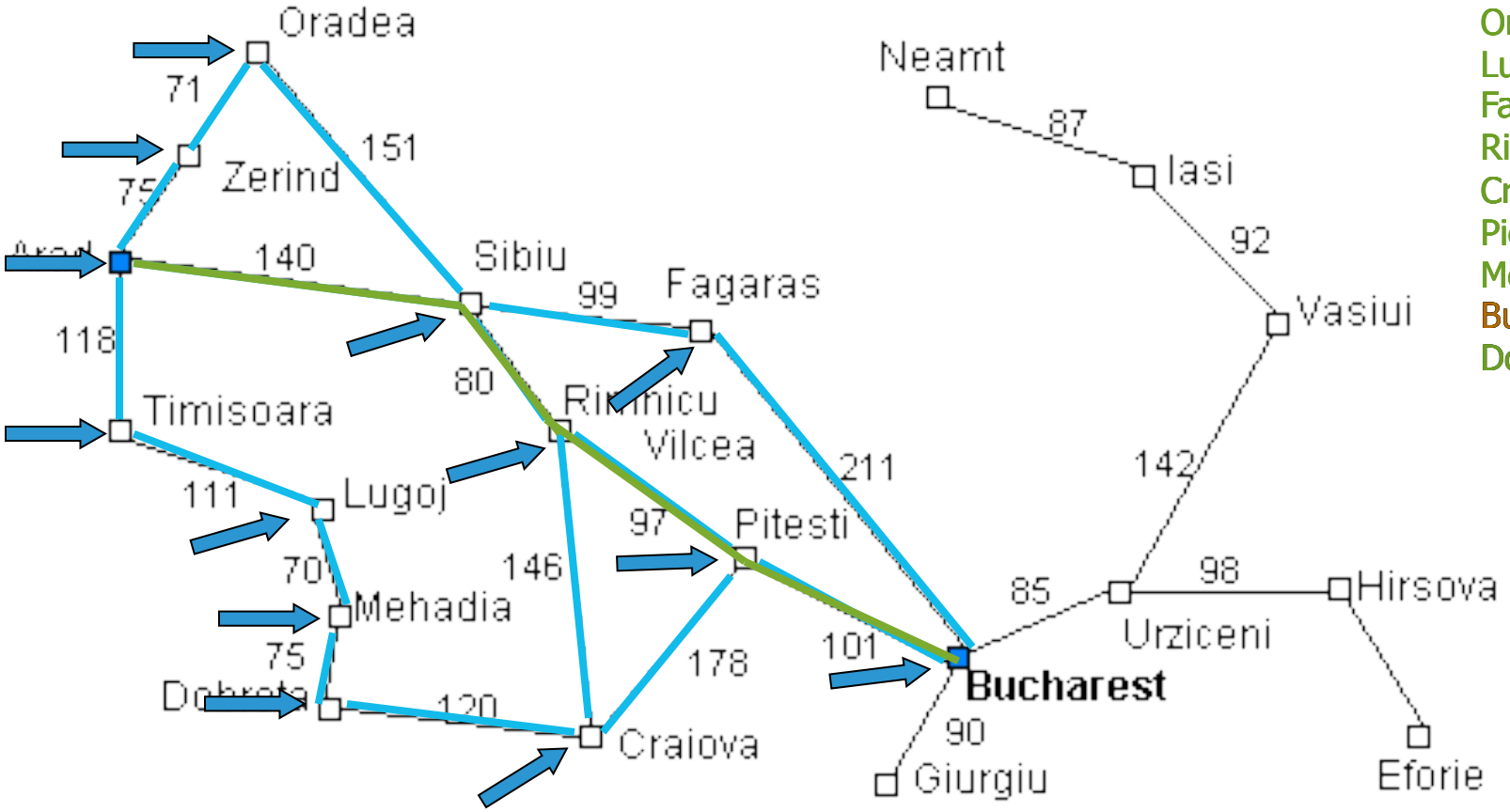
Goal test
- Are we in Bucharest?

Cost
- Move cost = distance between cities
- Path cost = distance travelled since Arad

# SUMMARY / EXAMPLE BREATH-FIRST SEARCH

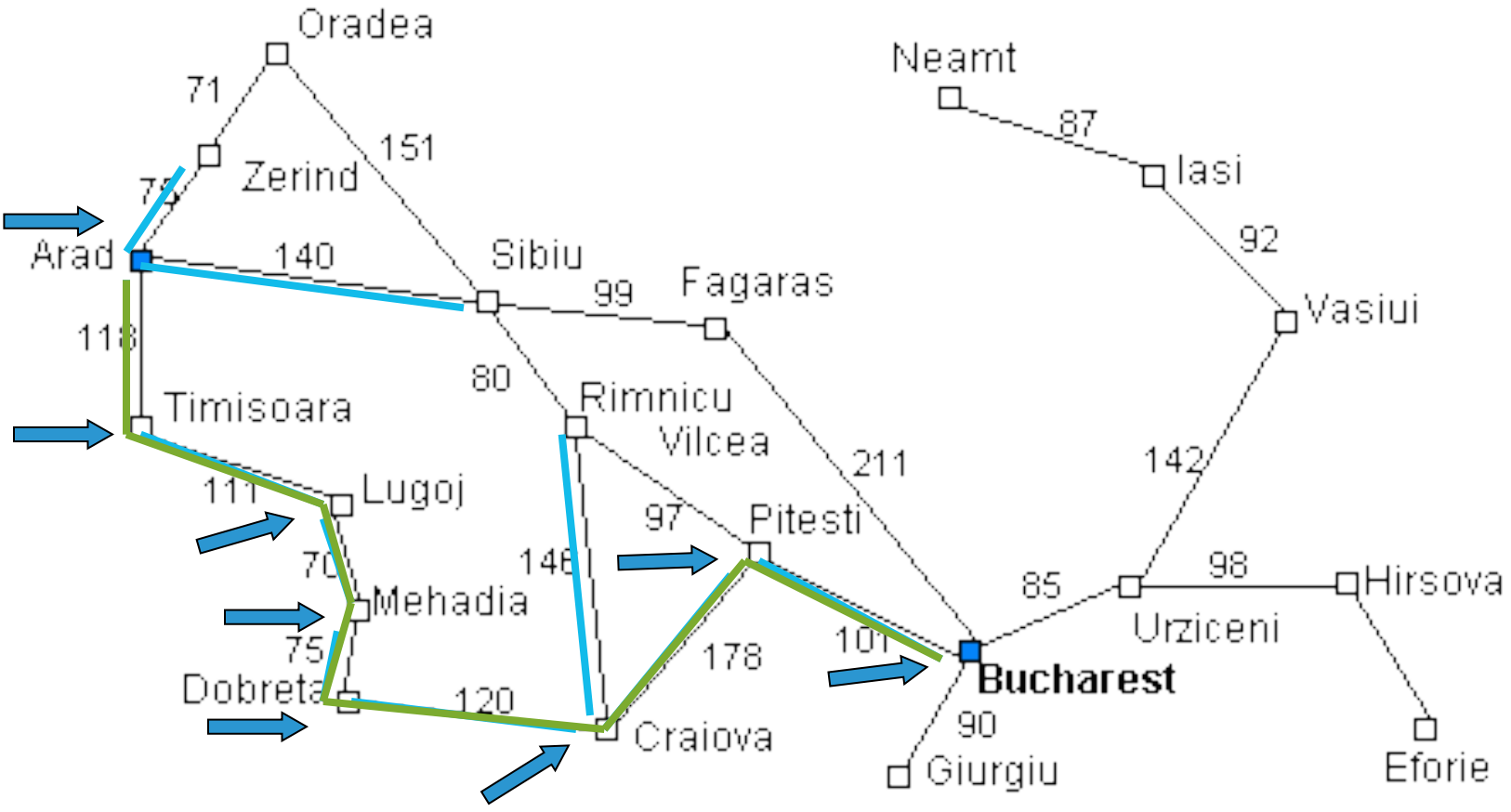


# SUMMARY / EXAMPLE

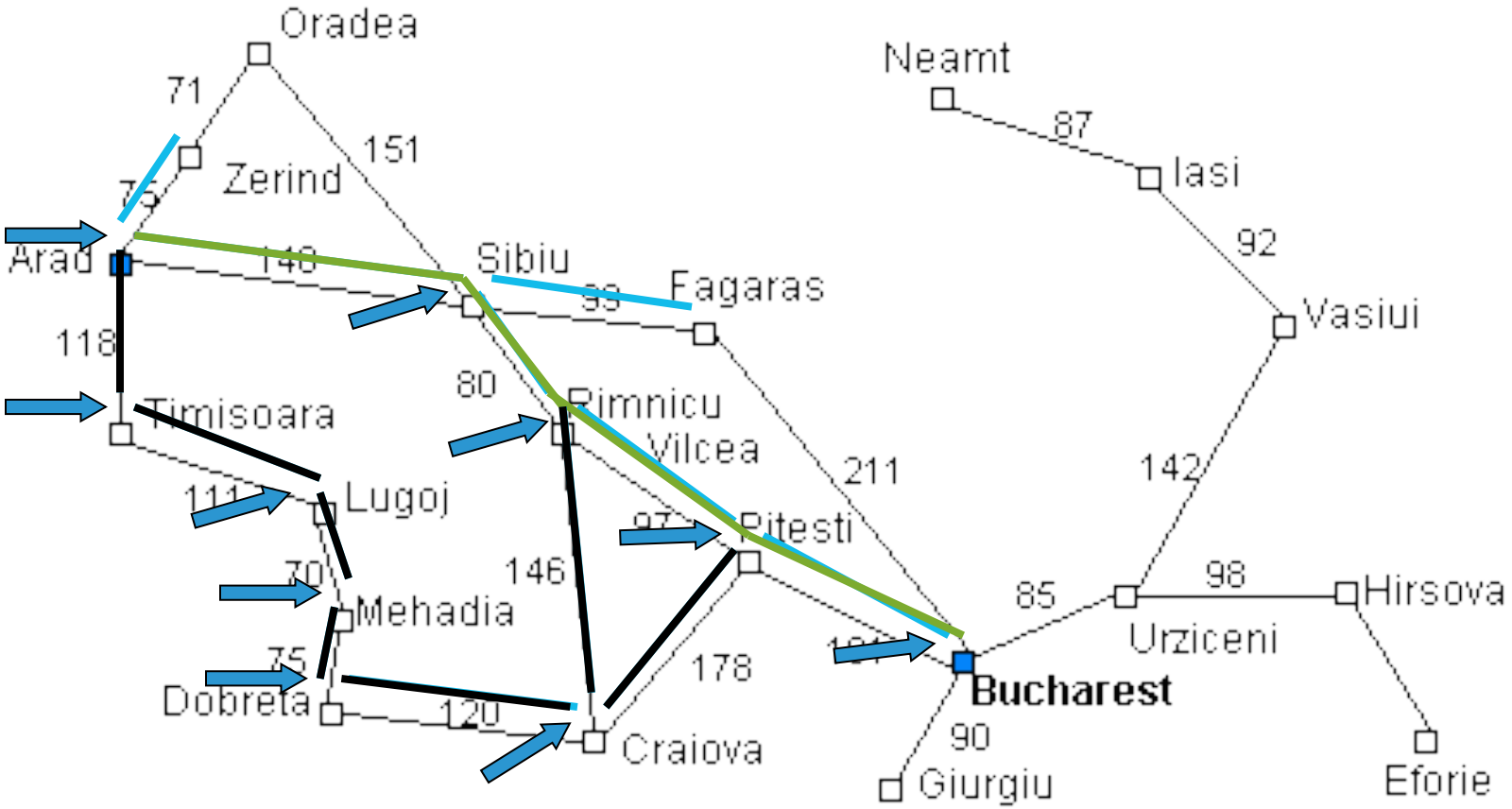



Arad 0
Zerind 75
Sibiu 140
Timisoara 118
Oradea 146
Lugoj 229
Fagaras 239
Rimnicu 220
Craiova 366
Pietsti 317
Mehadia 299
Bucharest 418
Dobreta 374

# SUMMARY / EXAMPLE DEPTH-FIRST



# SUMMARY / EXAMPLE



# SUMMARY / EXAMPLE ITERATIVE DEEPENING SEARCH