Problem Solving by Search

Uwe Egly

Vienna University of Technology
Institute of Information Systems
Knowledge-Based Systems Group

Outline

Overview

Search: very important technique in CS and AI

Different kinds of search:

- Deterministic search
  - Uninformed (“blind”) search strategies
  - Informed or heuristic search strategies: use information about problem structure
- Local search
- Search in game trees (not covered in this course)

In this lecture: A quick recapitulation of deterministic search

Example for a Search Problem

What is the current situation?

- On holiday in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

What is the desired (or goal) situation?

- To be in Bucharest in order to catch the flight

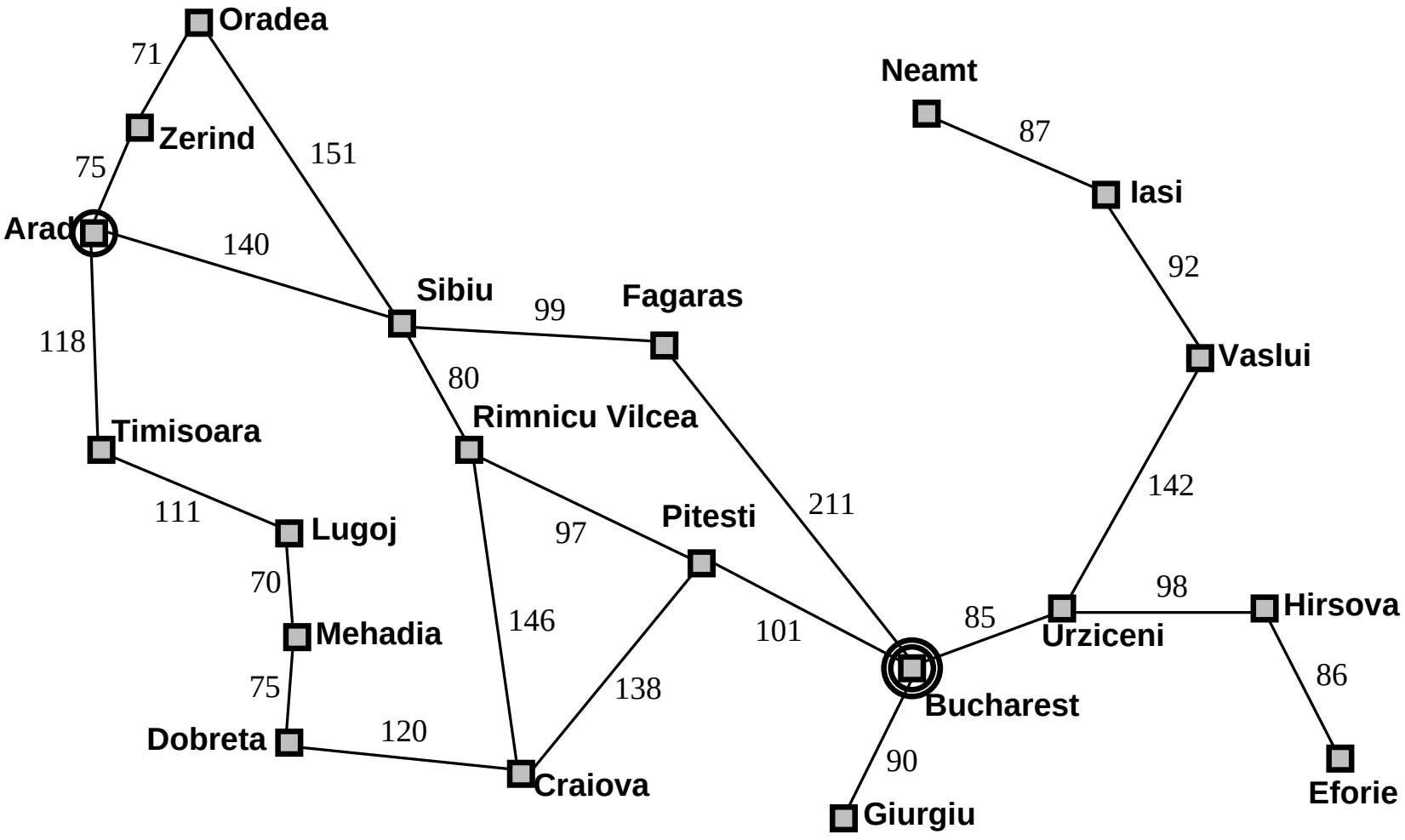
Formulate the problem!

- Various cities
- Drive between cities

Find a solution!

- Sequence of cities from Arad to Bucharest
  e.g., Arad, Sibiu, Fagaras, Bucharest

Example for a Search Problem: The Map



How do we Formulate a Search Problem?

A search problem consists of the following 4 components

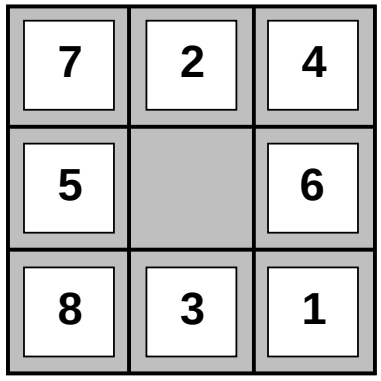
1. An initial state or start state, e.g., be at Arad
2. A non-empty set of goal states, either
   implicitly given as specific states, e.g., be at Bucharest, or
   defined as all states satisfying the goal test
3. A non-empty set of operators (action-state pairs)

$$Z_i \xrightarrow{\text{costs}} Z_{i+1}$$

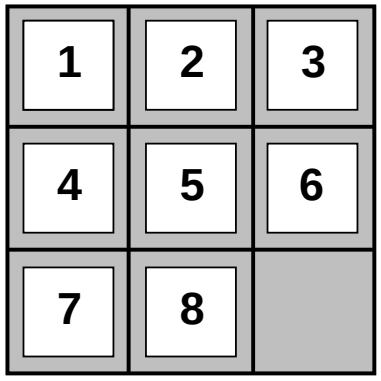
   - E.g., drive from Arad to Sibiu
   - Transforms Z_i into the successor state Z_{i+1}
   - Associated to each operator are positive costs
   - No costs given $\rightsquigarrow$ unit costs

4. Function to compute the path costs from operator costs
   e.g., $\sum$ of operator costs of operator applications on a path

Example of a Search Problem: 8-Puzzle



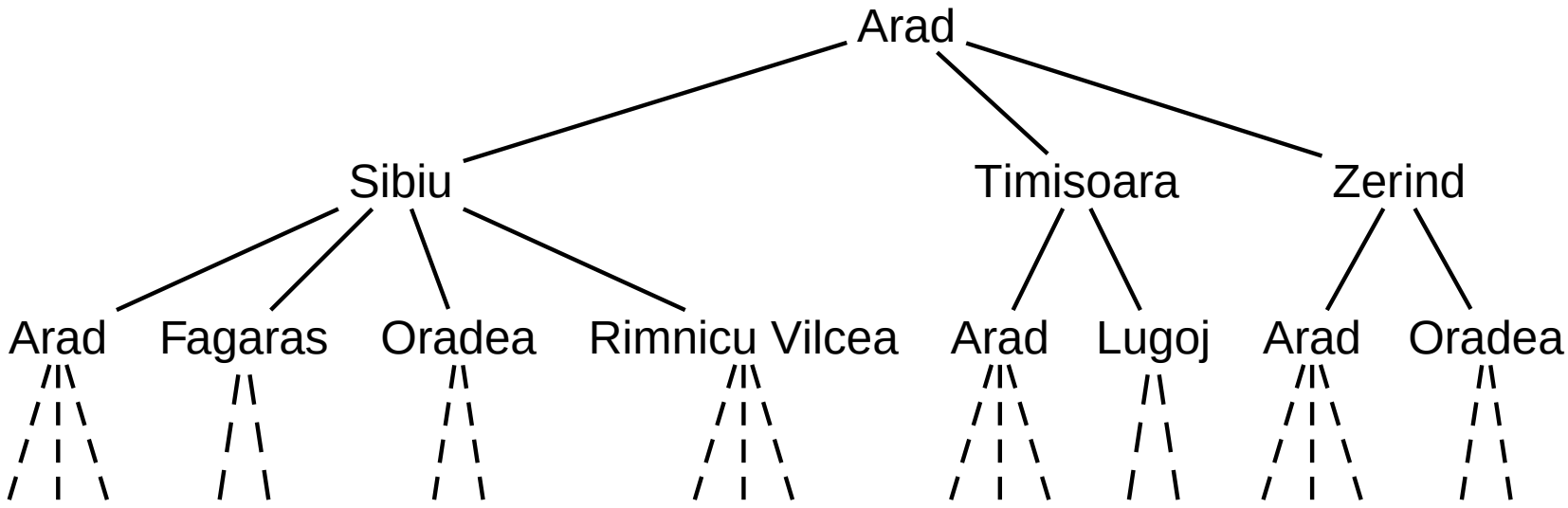
Start State



Goal State

- States: integer locations of tiles (ignore intermediate positions)
- Operators: move blank left, right, up, down (ignore unjamming)
- Operator costs: unit cost, i.e., 1 per move
- Goal test: = goal state (given)
- Path costs: summation over operator costs on the path
- Solution: sequence of rules transforming start into a goal state
- State space dynamically generated by possible moves

The State Space for the Holiday Example



Find solution = search the state space for a path

- from the start state
- to a goal state

Search Strategies

A strategy is defined by picking the order of node expansion

Four important properties of a strategy:

1. Completeness: Does it always find a solution if one exists?
2. Space complexity: Maximum number of nodes in memory
3. Time complexity: Number of nodes generated/expanded
4. Optimality: Does it always find a least-cost solution?

Degree of complexity (poly, exp, etc.) always wrt problem size!

Time and space complexity are measured in terms of

b : maximum branching factor of the search tree

d : depth of the least-cost solution

m : maximum depth of the state space (may be ∞)

Recall: Big O Notation

Definition

$g(n)$ is in $O(f(n))$, if there exist two positive constants c and n_0, such that

$$g(n) \leq c \cdot f(n)$$

holds for all $n > n_0$.

Uninformed Search Strategies (USS)

USSs use only the info available in the problem definition

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

Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

Example: Search 3 levels to go from Arad to Bucharest

Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

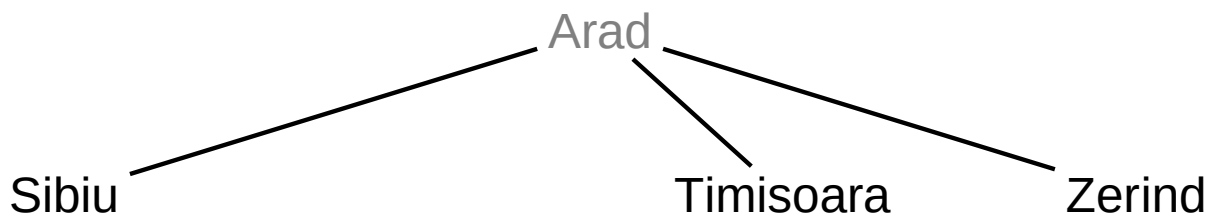
Example: Search 3 levels to go from Arad to Bucharest

Arad

Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

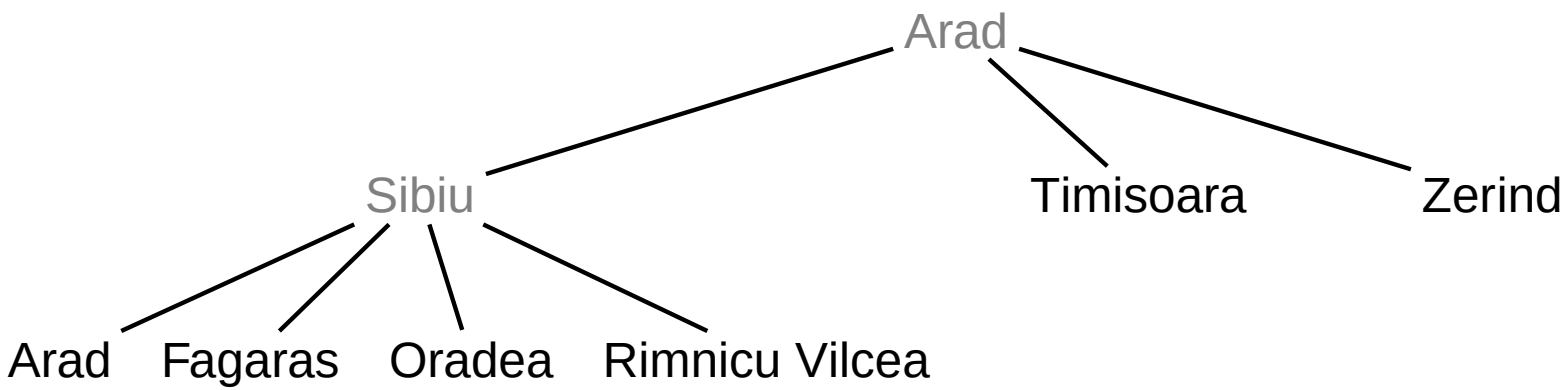
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

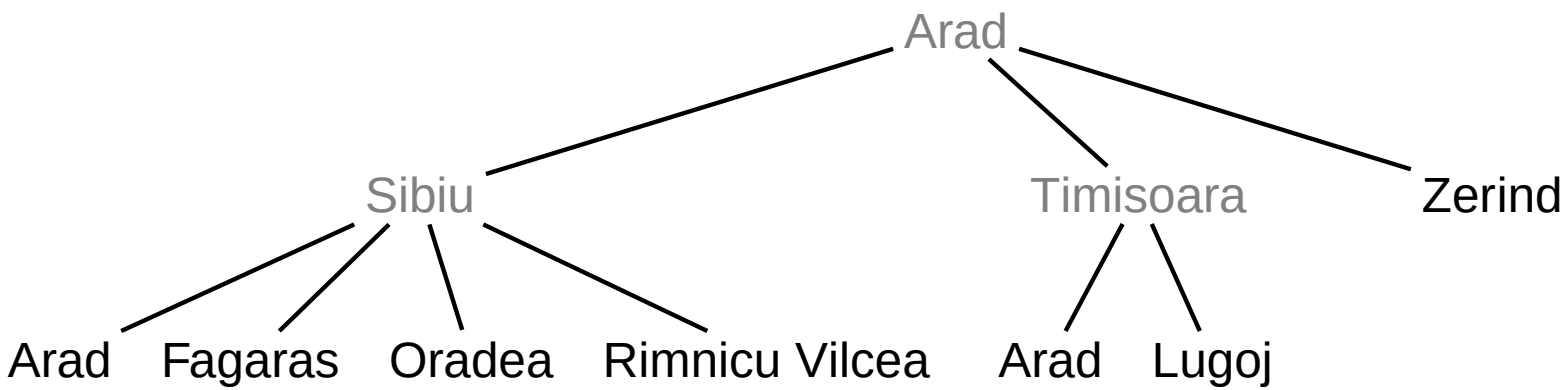
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

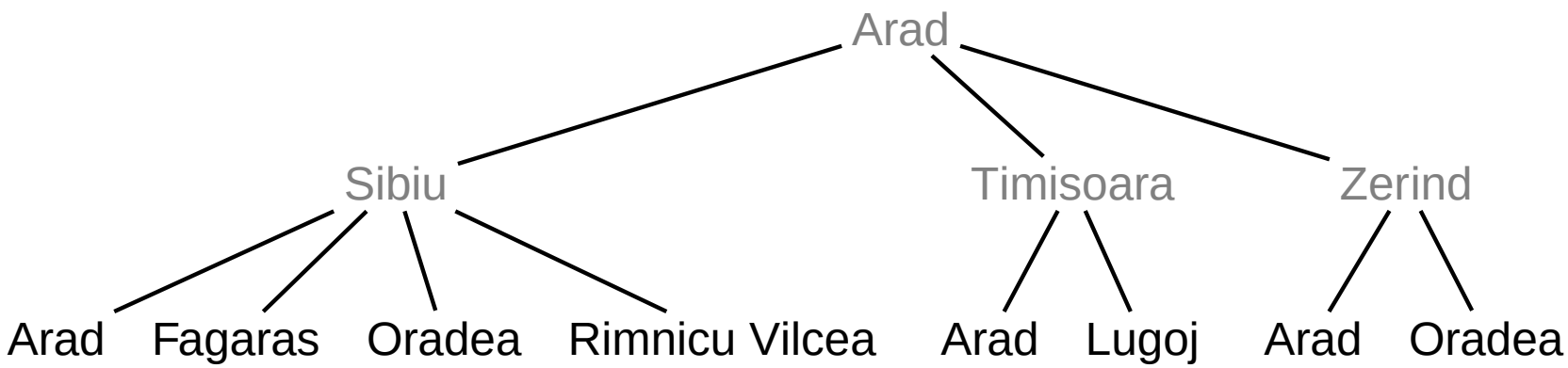
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

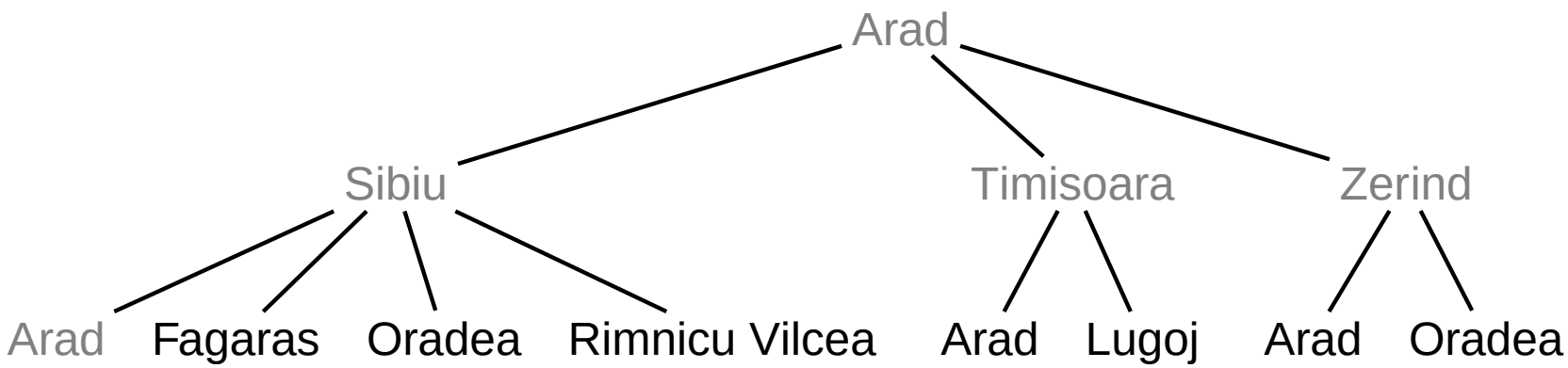
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

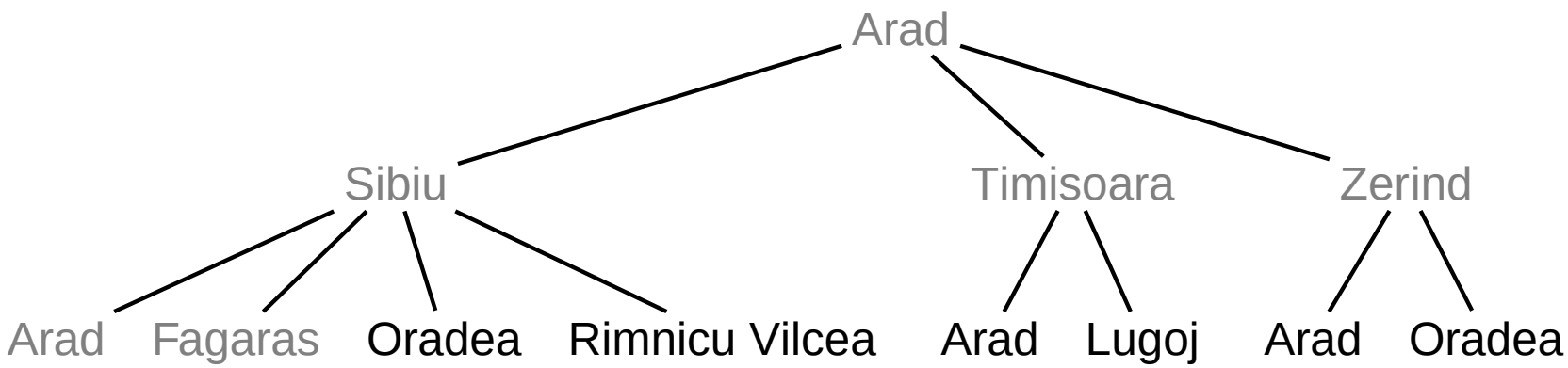
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

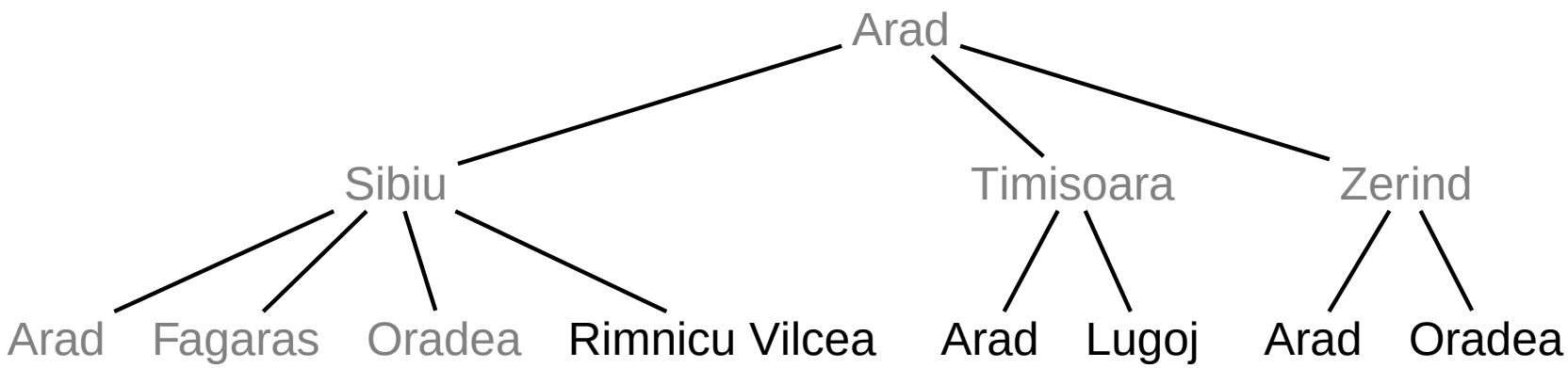
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

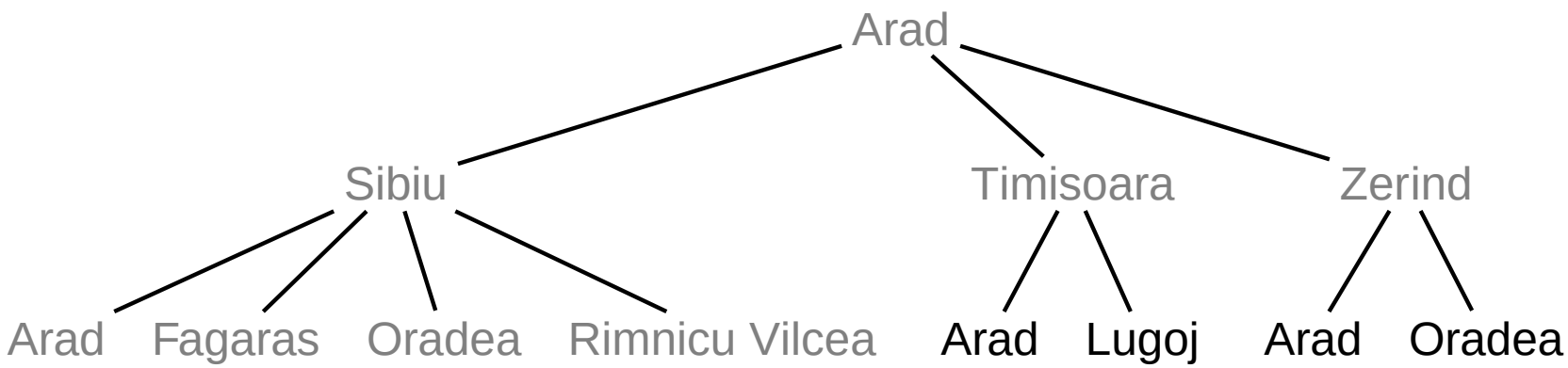
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

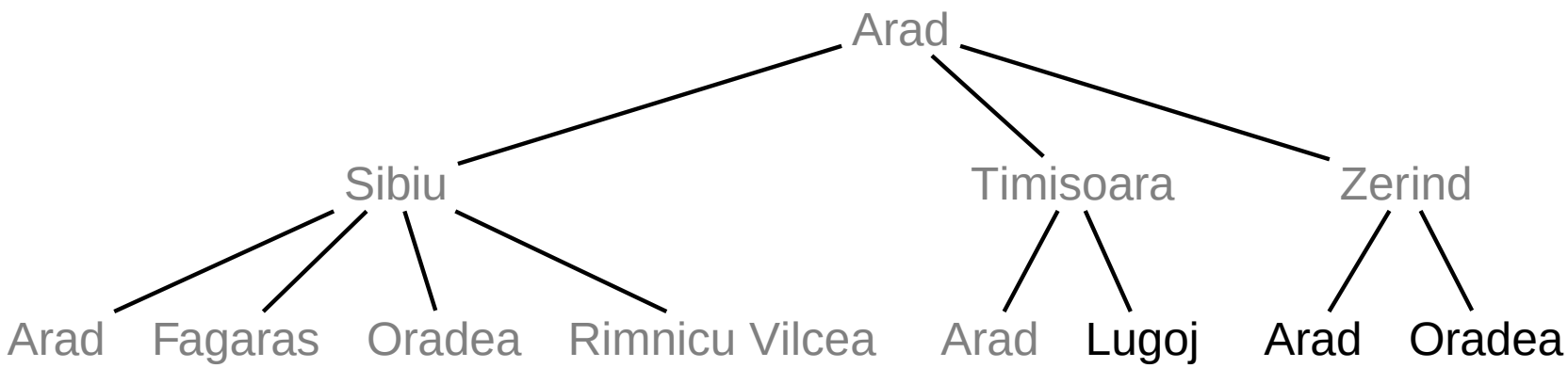
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

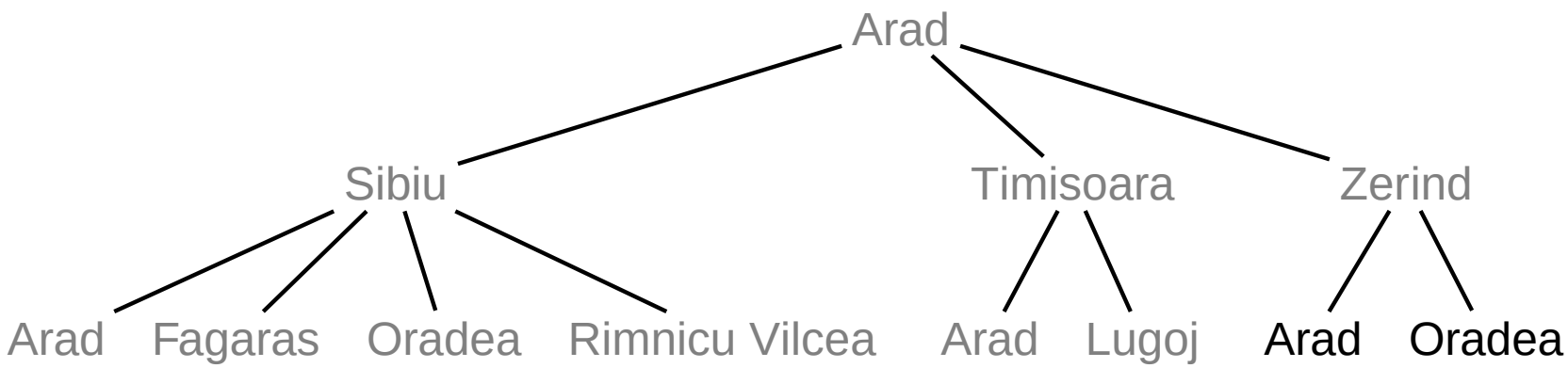
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

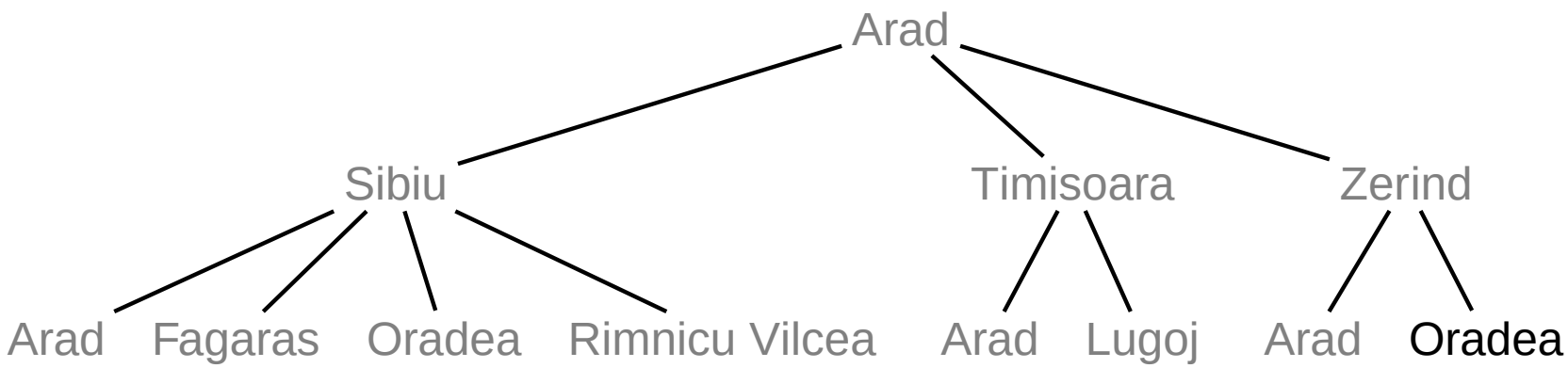
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

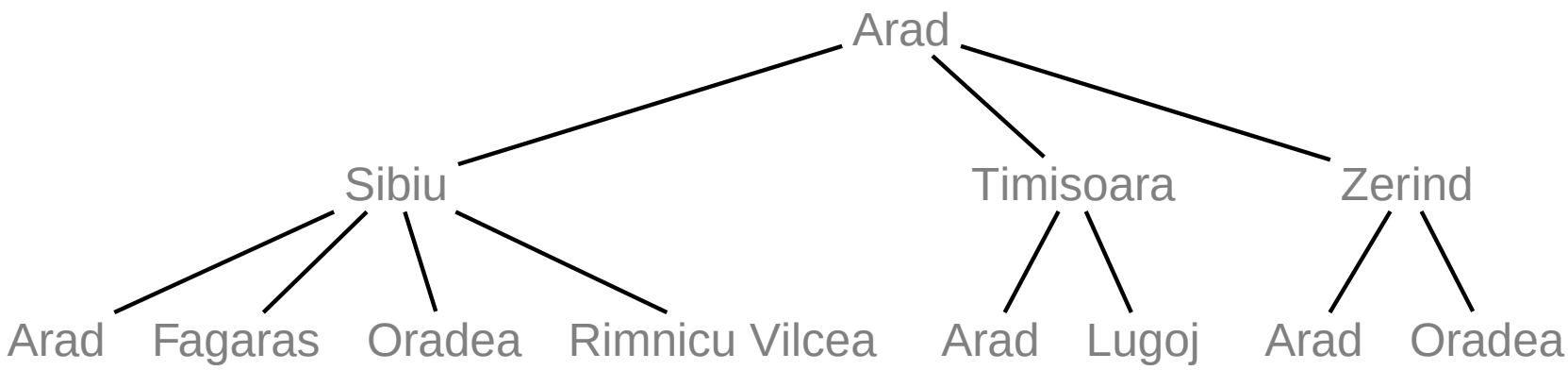
Example: Search 3 levels to go from Arad to Bucharest



Breadth-first Search (BFS)

Idea: Expand a shallowest unexpanded node

Example: Search 3 levels to go from Arad to Bucharest



Properties of BFS

Completeness:	Yes (if b is finite)
Space complexity:	$O(b^d)$, i.e., exponential in d
(keep any node)	$(1 + b + b^2 + \ldots + b^d = \frac{(b^{d+1}-1)}{(b-1)})$
Time complexity:	$O(b^d)$, i.e., exponential in d
Optimality:	Yes (when using unit costs)

Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest

Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

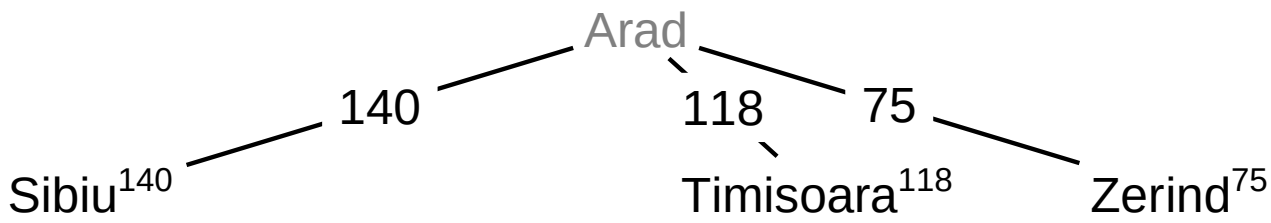
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest

Arad

Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

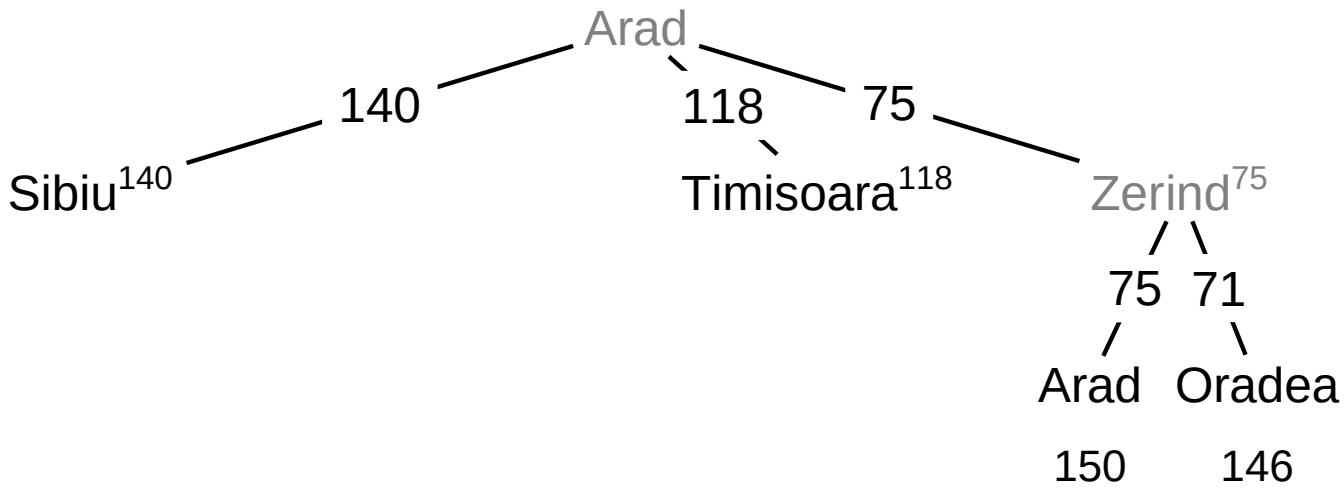
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

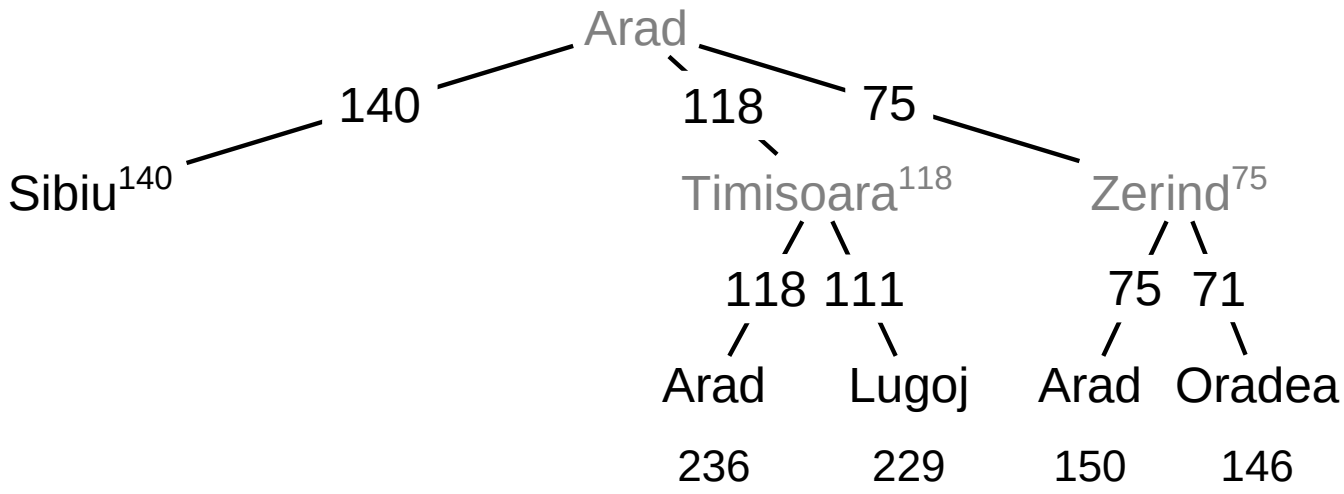
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

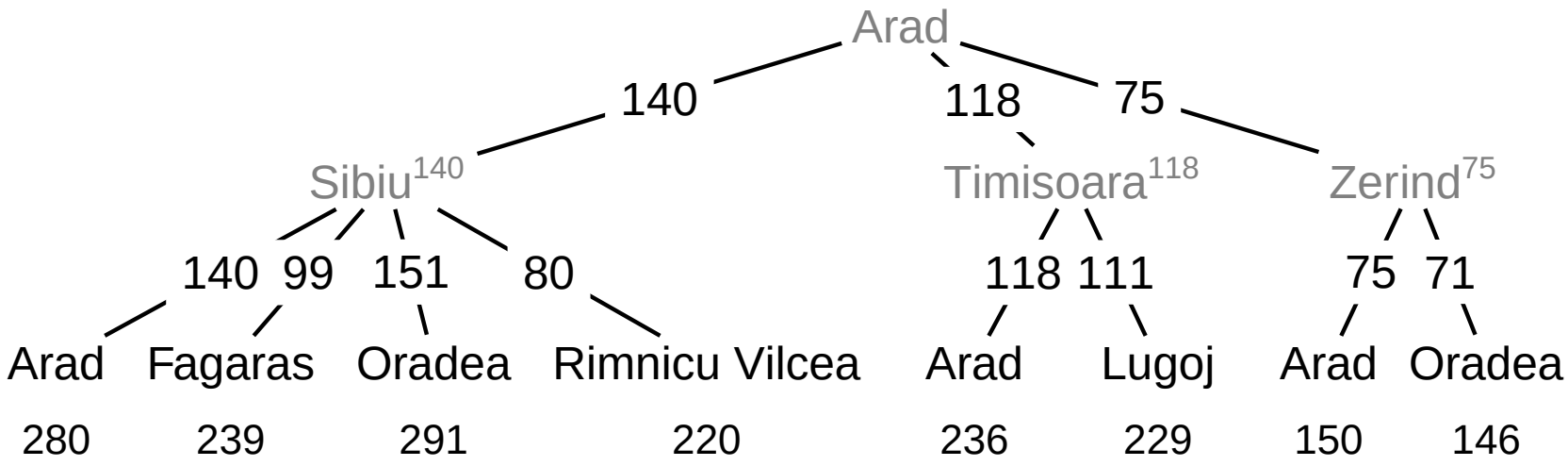
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

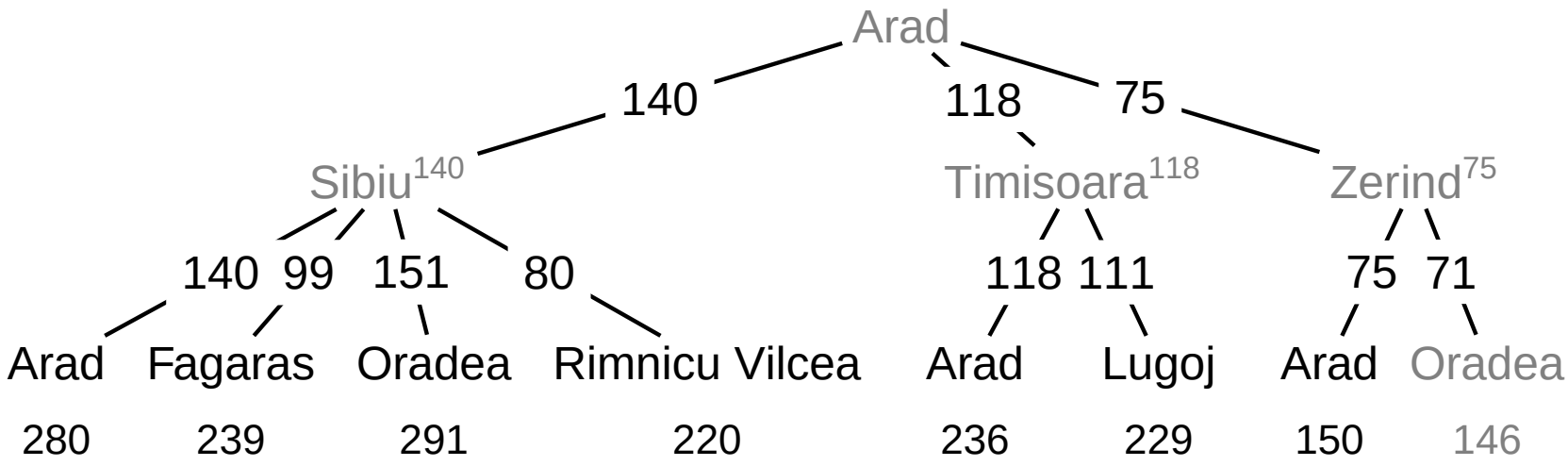
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

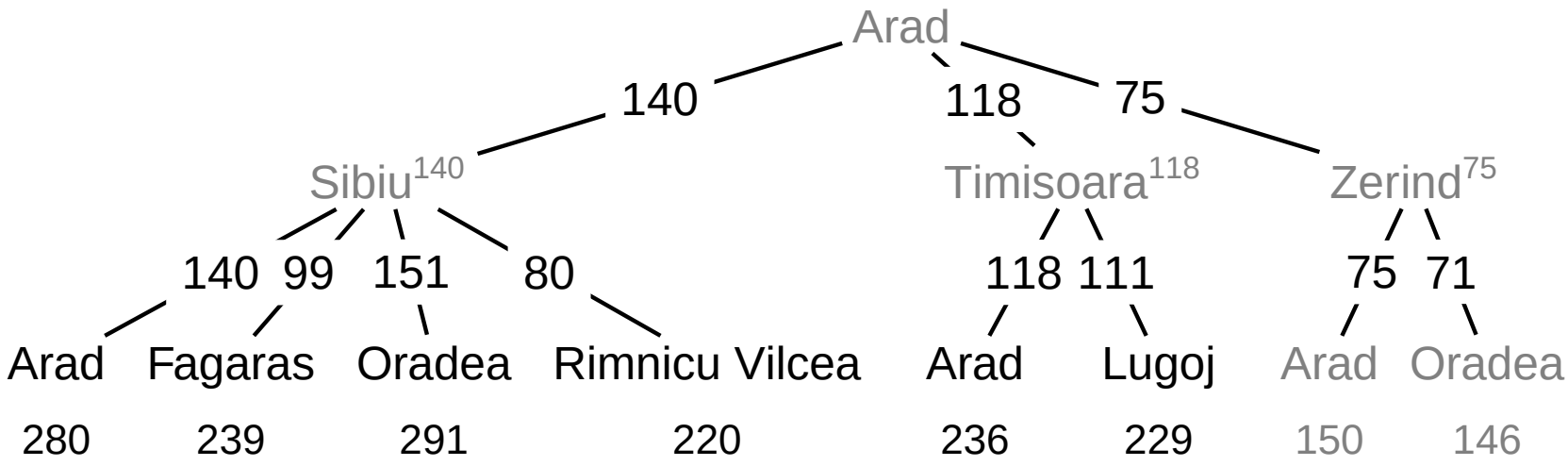
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

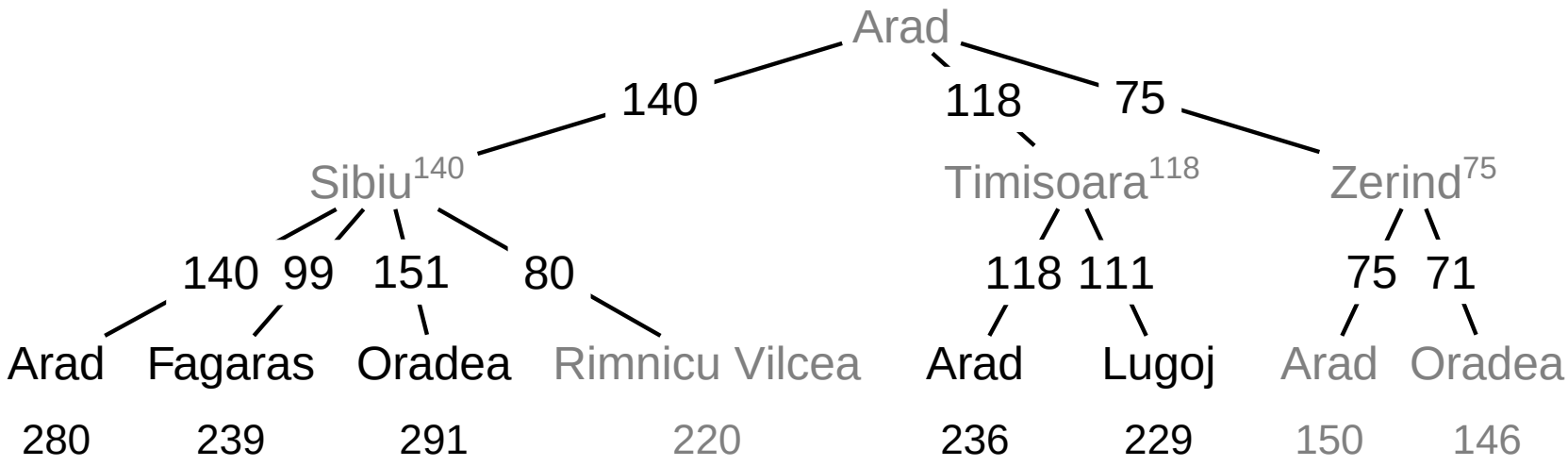
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

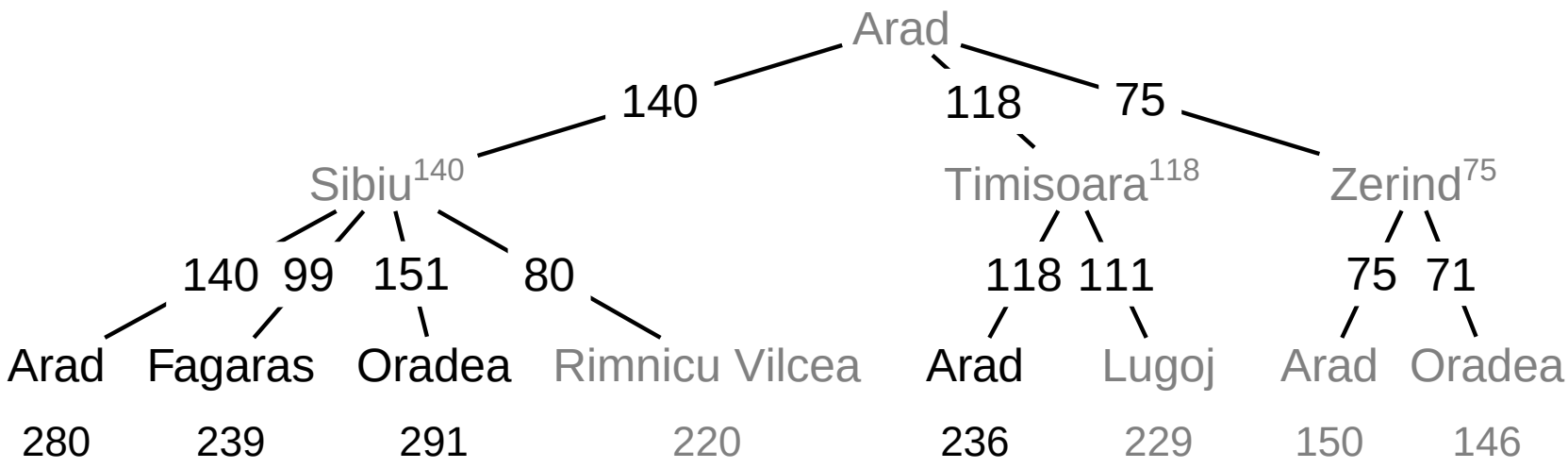
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

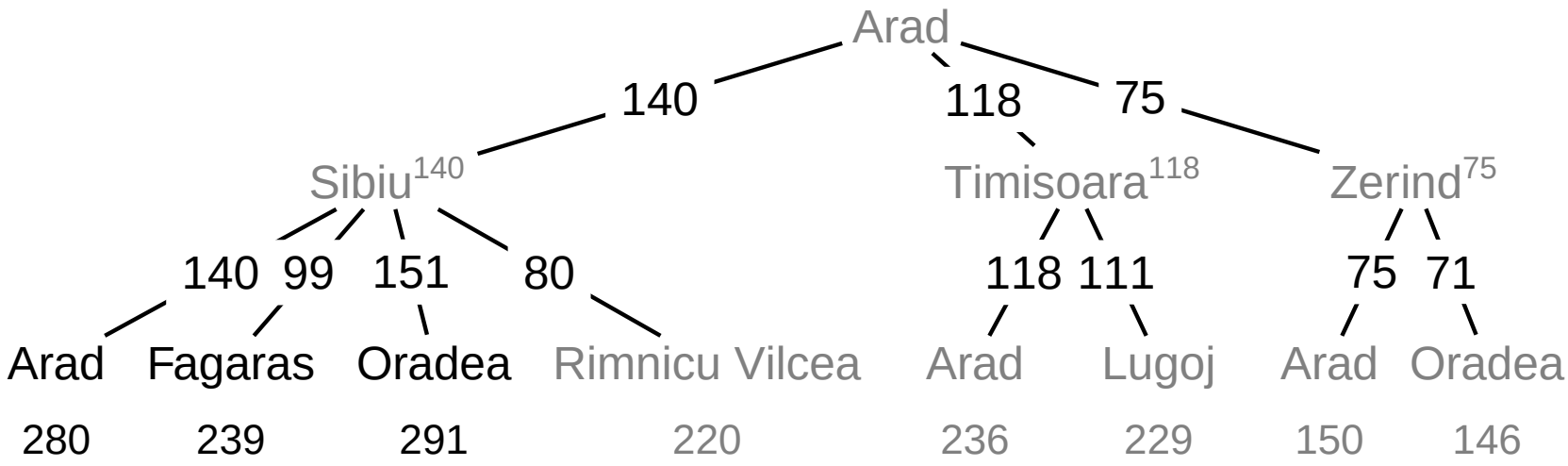
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

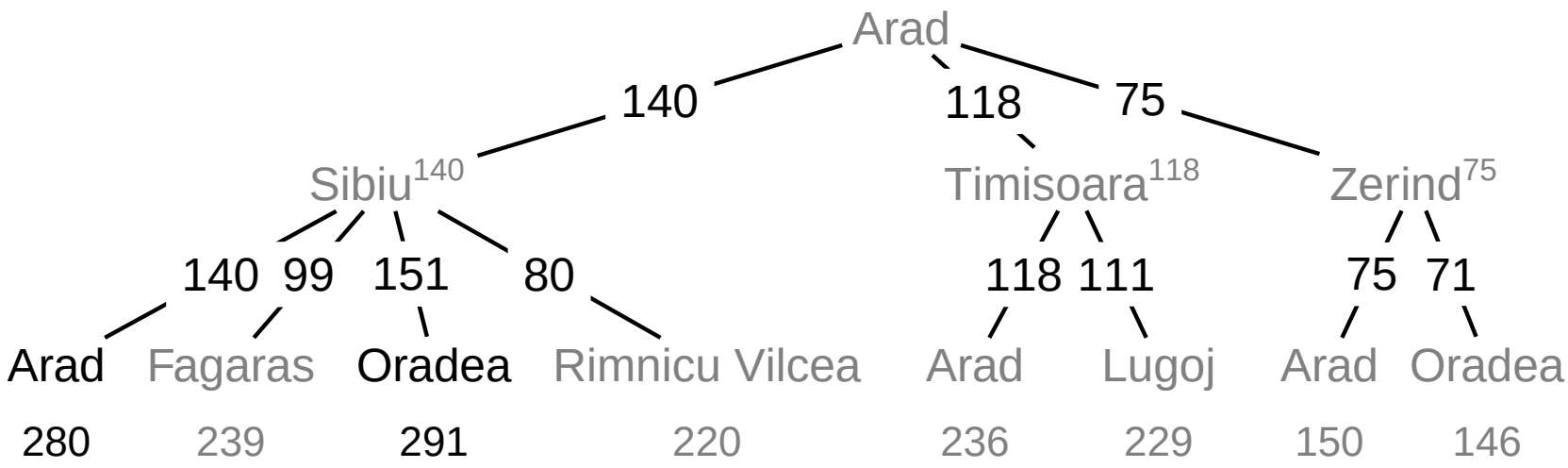
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

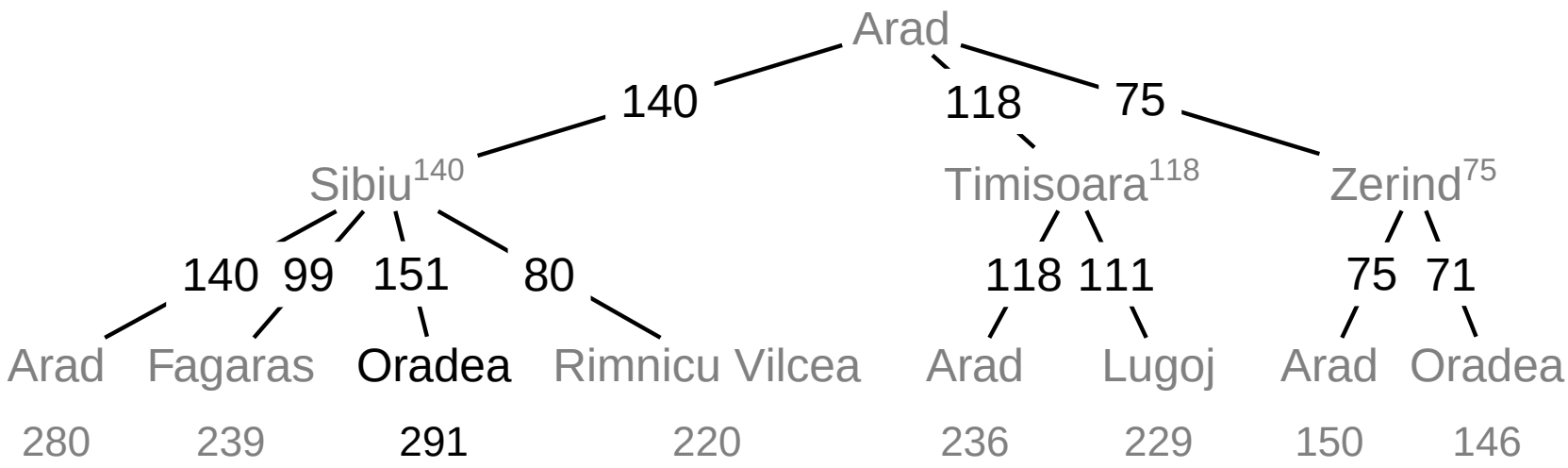
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

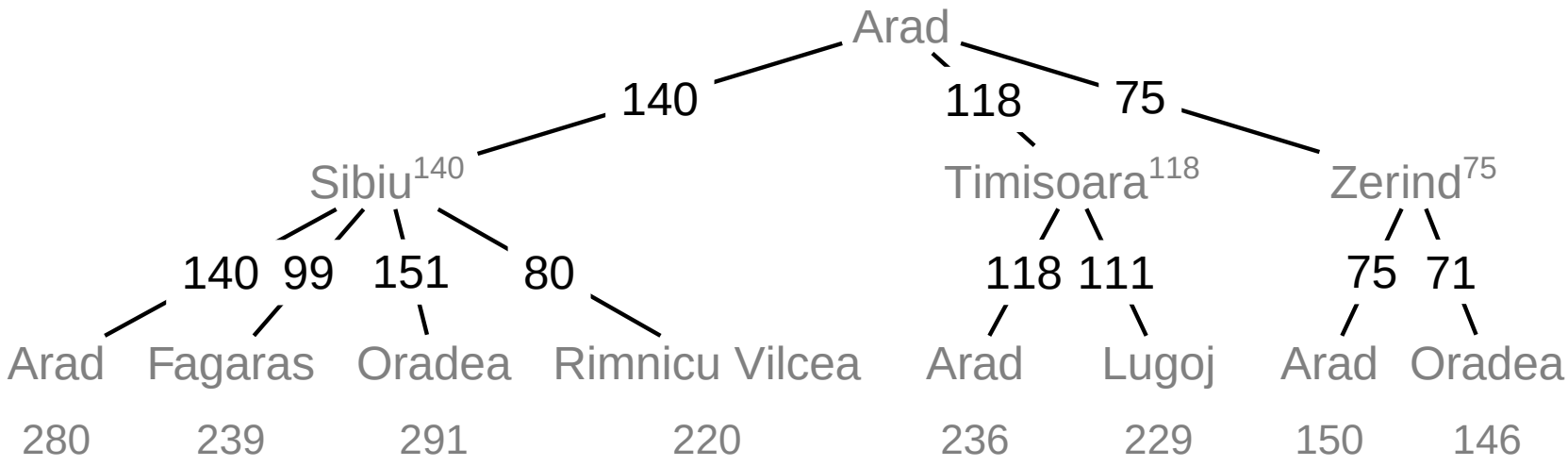
Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Uniform-cost Search (UCS)

Idea: Expand an unexpanded node with minimal costs

Exa: Use $\sum$ and search 3 levels to go from Arad to Bucharest



Properties of UCS

Special case: BFS (= UCS with unit costs)

Let C^* be the costs of an optimal solution, ϵ positive

Completeness: Yes, if b is finite and operator cost $\geq \epsilon$

Space complexity: # of nodes with $g \leq C^*$ is $O(b^{\lceil C^*/\epsilon \rceil})$

Time complexity: # of nodes with $g \leq C^*$ is $O(b^{\lceil C^*/\epsilon \rceil})$

Optimality: Yes, because nodes expanded in increasing order of $g(n)$

Attention: $\lceil C^*/\epsilon \rceil$ is an approximation for the depth

Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Arad

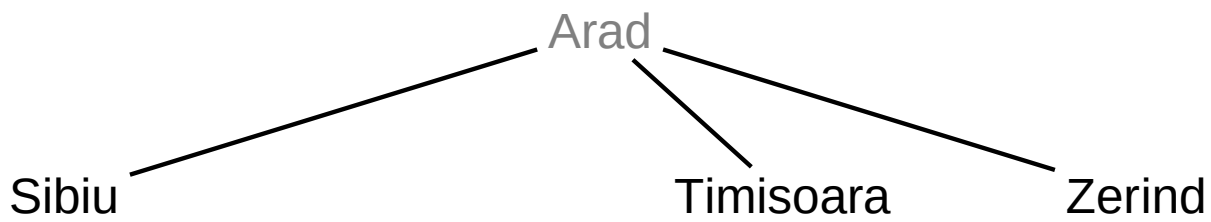
Arad

Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Sibiu

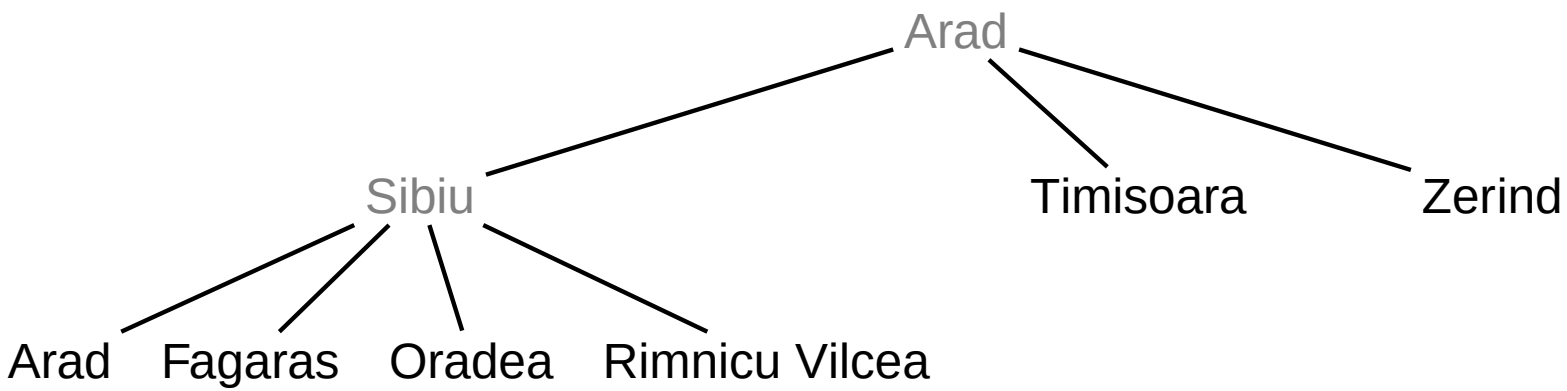


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Arad

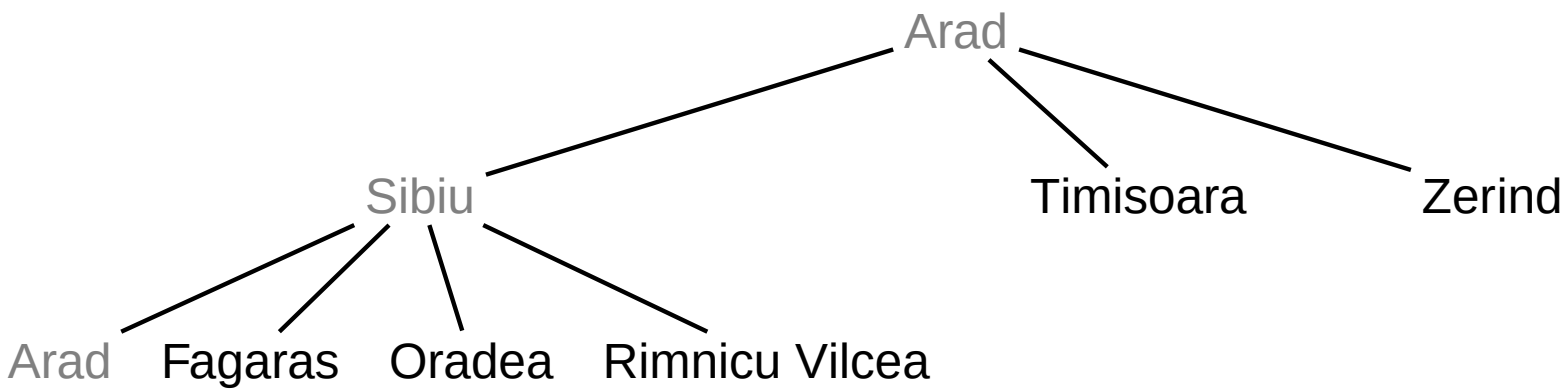


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Arad

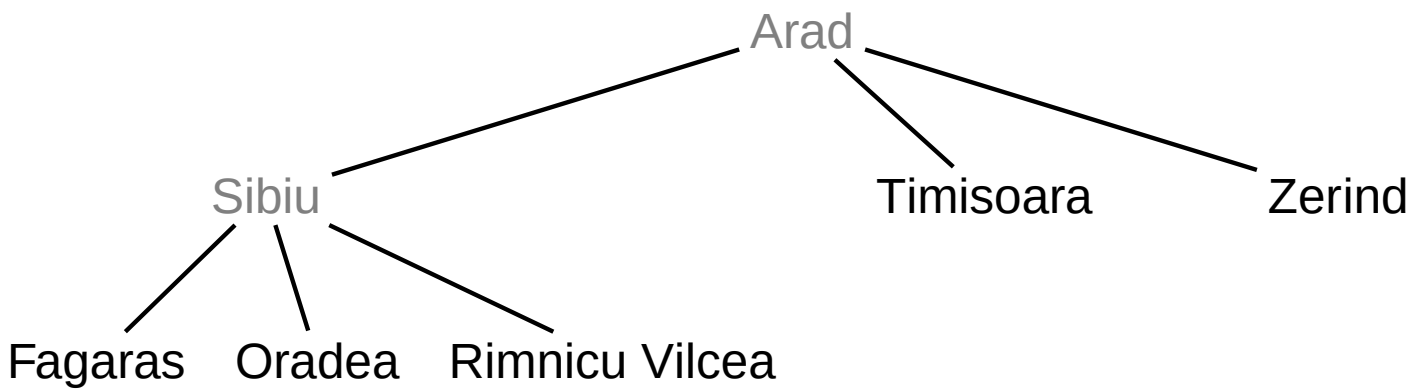


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Fagaras

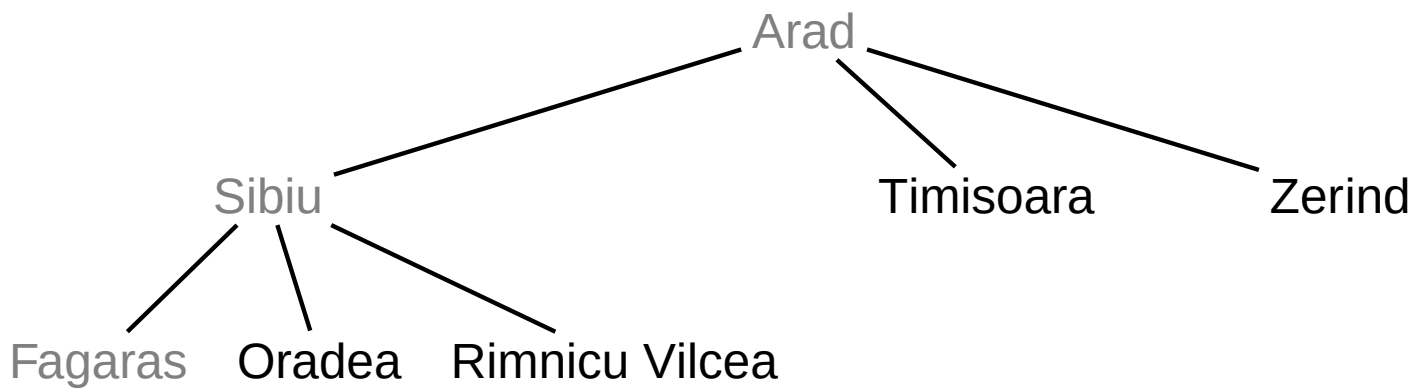


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Fagaras

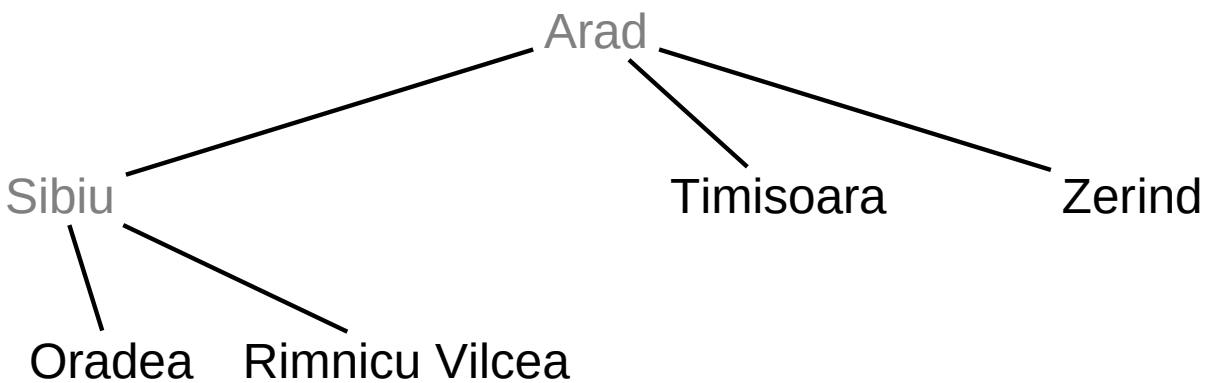


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Oradea

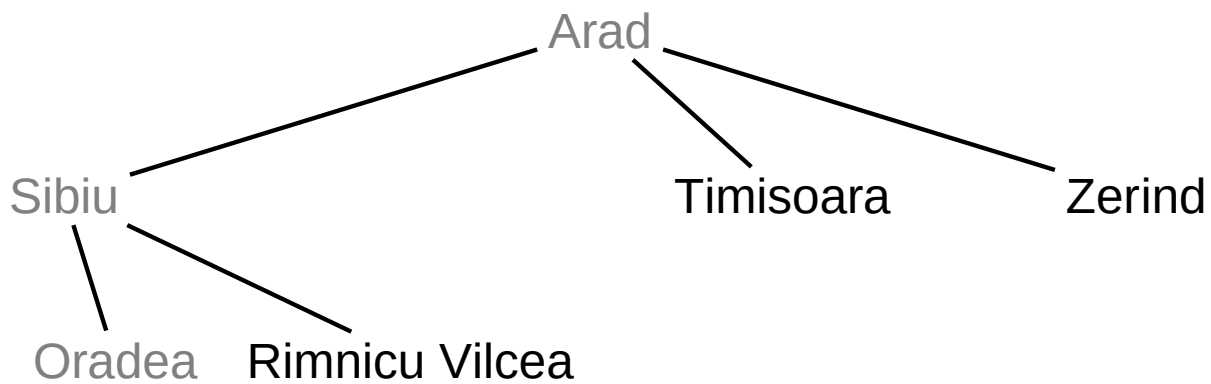


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Oradea

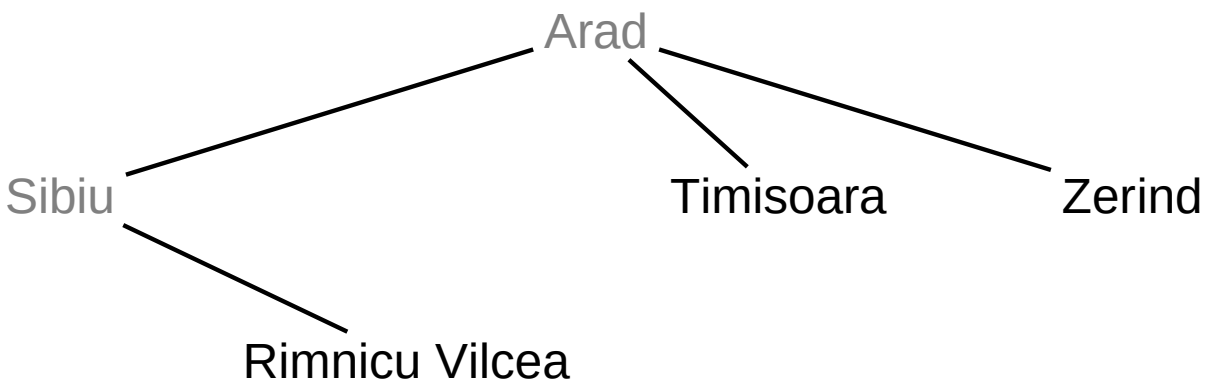


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Rimnicu Vilcea

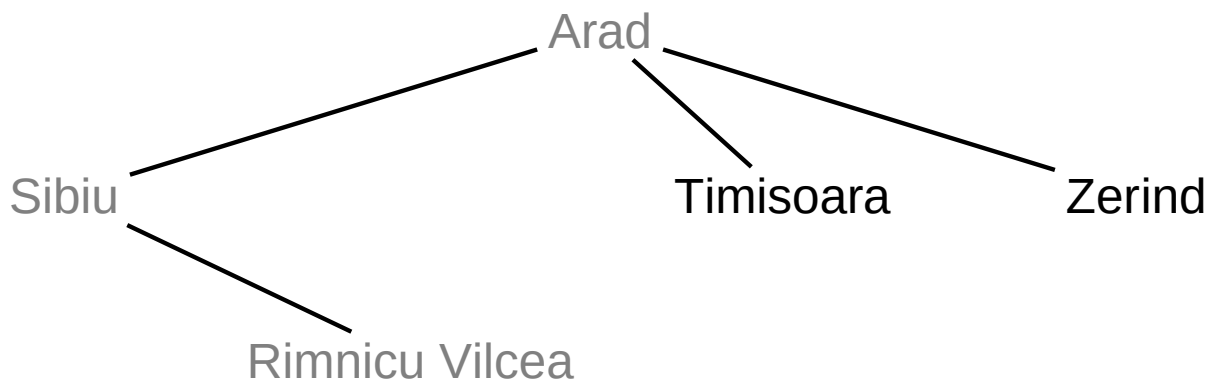


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Rimnicu Vilcea

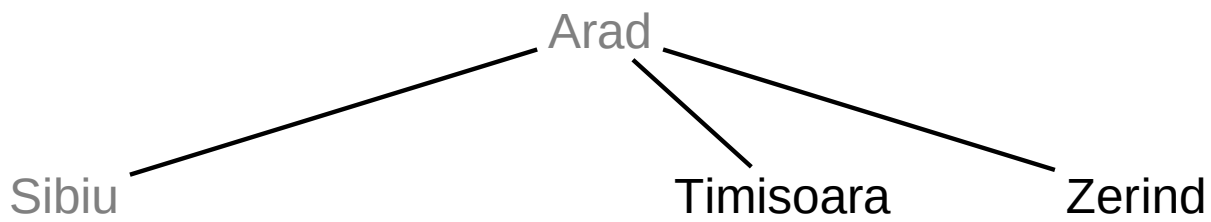


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Sibiu

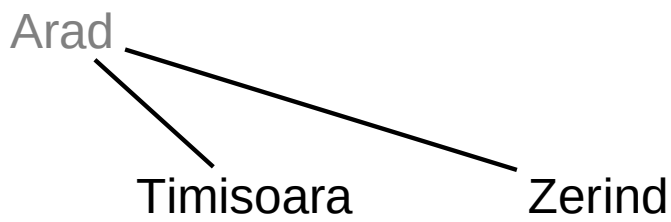


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Timisoara

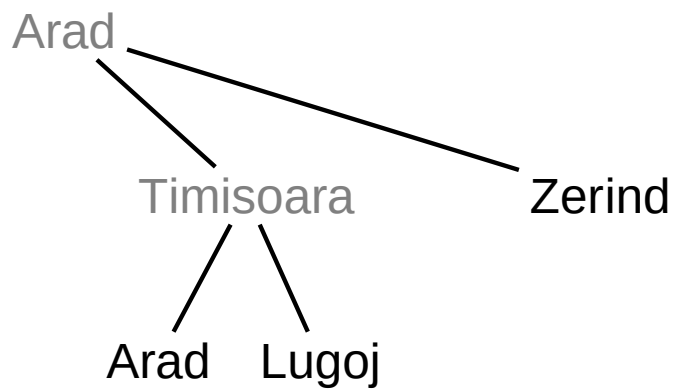


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Arad

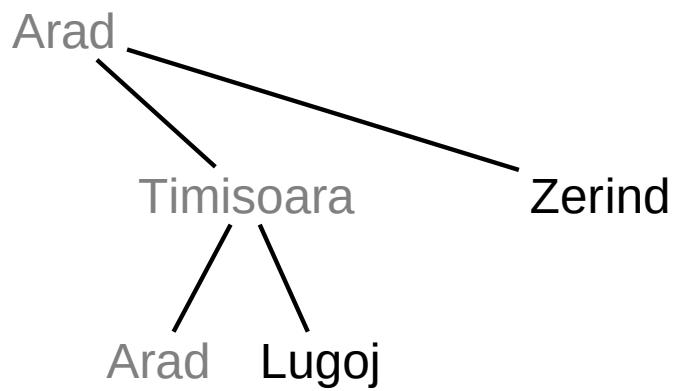


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Arad

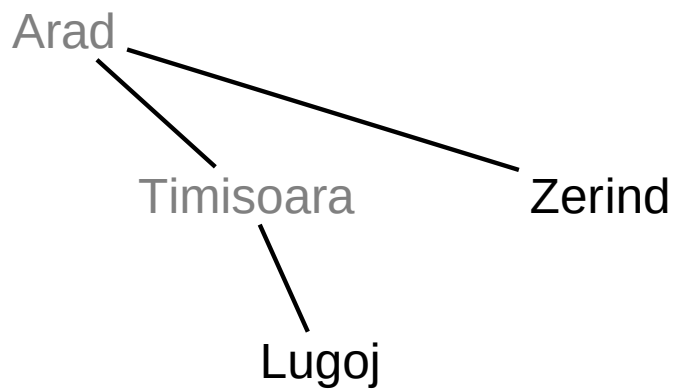


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Lugoj

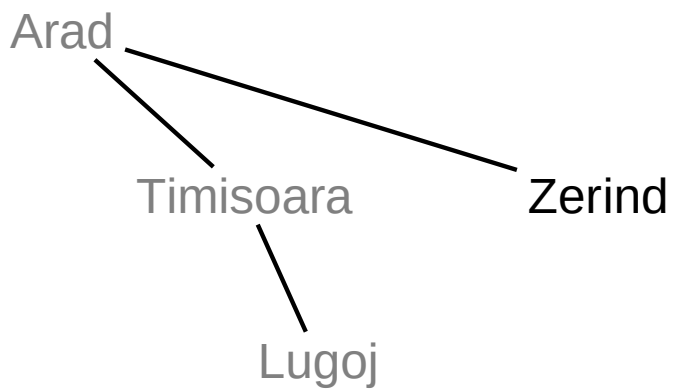


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Lugoj

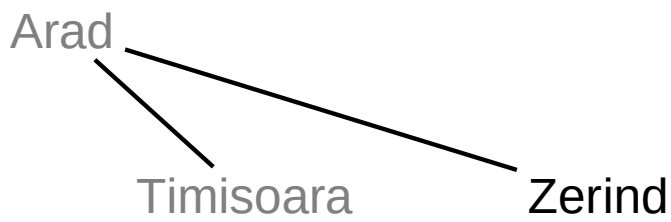


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Timisoara

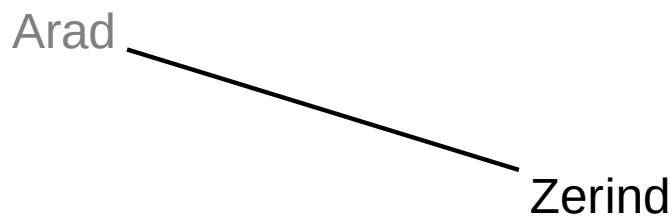


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Zerind

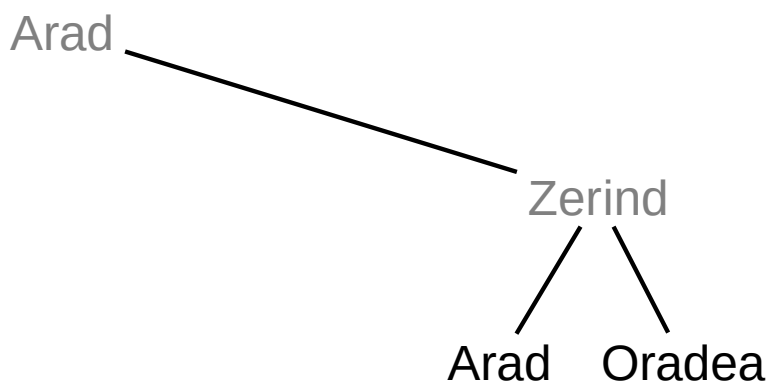


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Arad

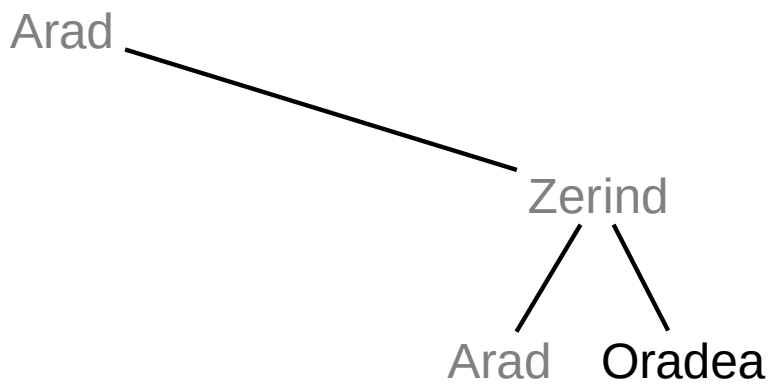


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Arad

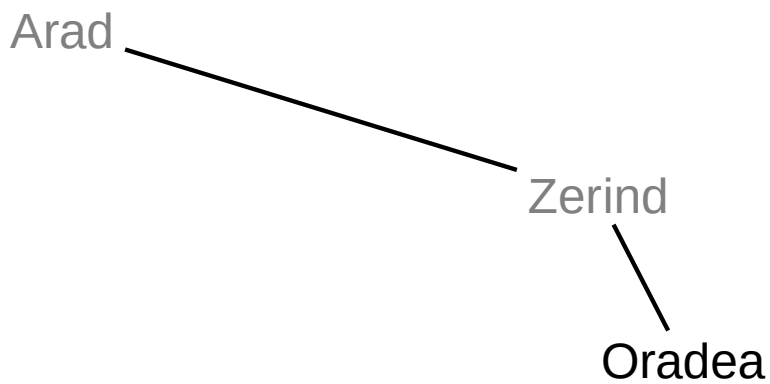


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Expand node Oradea

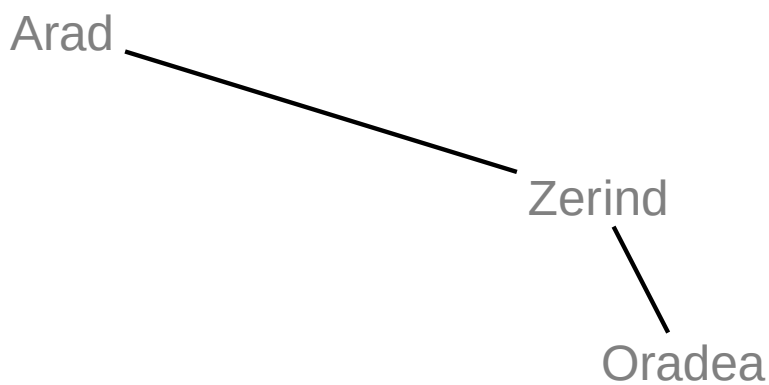


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Oradea

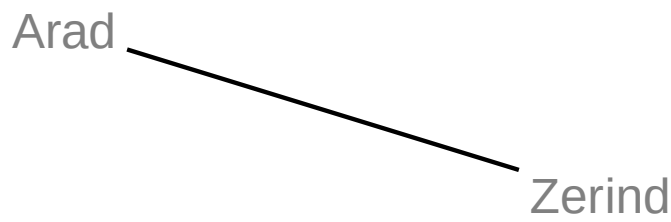


Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Zerind



Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Backtrack node Arad

Arad

Depth-first Search (DFS)

Idea: Expand an unexpanded node of maximal depth

Example: Search the space given by BFS (i.e., fix it to 3 levels)

Search completed!

Properties of DFS

Problem: DFS can be non-terminating, i.e., $m \rightarrow \infty$
(even if a solution is in the search space)

Completeness:	No
Space complexity:	$O(bm)$, i.e., space is linear in m!
Time complexity:	$O(b^m)$ terrible if m is much larger than d
Optimality:	No

Depth-limited Search (DLS)

DLS: DFS with depth limit l (nodes at depth l have no successors)

Completeness:	Yes, if $l > d$
Space complexity:	$O(bl)$
Time complexity:	$O(b^l)$
Optimality:	No

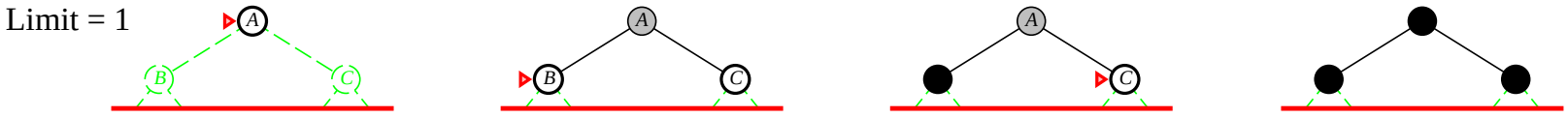
Depth-first Iterative Deepening Search (DFIDS)

Idea: Set $l = 0, 1, 2, \ldots$ and use DLS with limit l as a subroutine

Limit = 0

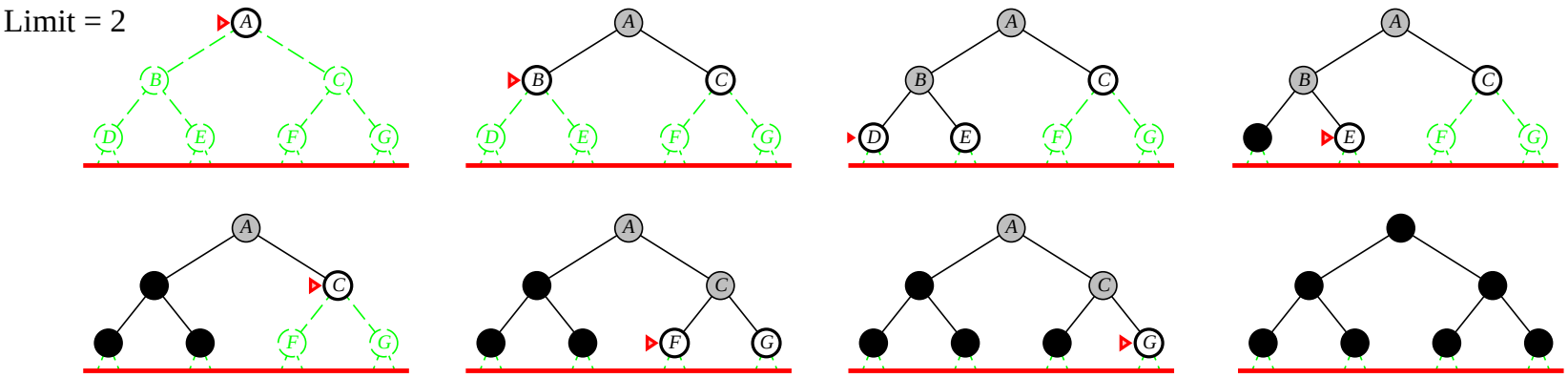
Depth-first Iterative Deepening Search (DFIDS)

Idea: Set $l = 0, 1, 2, \ldots$ and use DLS with limit l as a subroutine



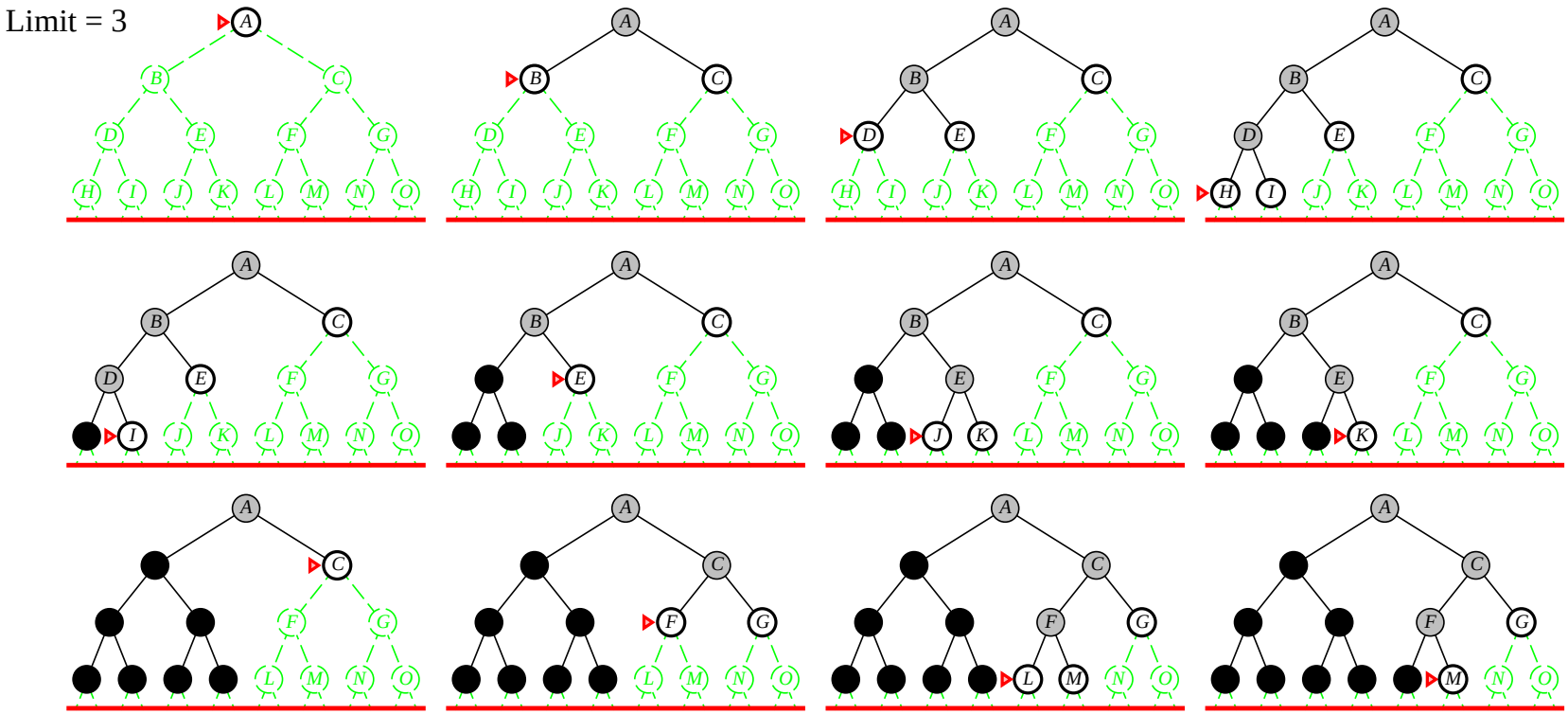
Depth-first Iterative Deepening Search (DFIDS)

Idea: Set $l = 0, 1, 2, \ldots$ and use DLS with limit l as a subroutine



Depth-first Iterative Deepening Search (DFIDS)

Idea: Set $l = 0, 1, 2, \ldots$ and use DLS with limit l as a subroutine



Properties of DFIDS

Completeness: Yes

Space complexity: $O(bd)$

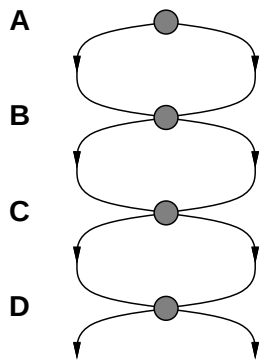
Time complexity: $O(b^d)$

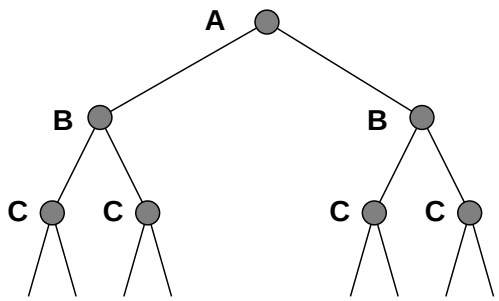
$(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d$

Optimality: Yes (when using unit costs)

Tree Search vs Graph Search

- So far, the constructed search space was a tree
- In travel example, paths like Arad–Sibiu–Arad occurred
- Moreover, e.g., Arad occurred on several such paths
- Expanding the same node more than once is computationally expensive and should be avoided
- Search is graph-based instead of tree-based





Benefits and Problems of Graph Search

- GS can be "exponentially more efficient" than tree search
- With graph search, it is often harder to prove optimality
- No problem for UCS with unit or constant step costs
- In general: If $>$ 1 paths to the same state exist, take care that you choose the "good one" for the expansion
- More problematic for heuristic search like A^*
  (see next lecture)

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms