Artificial Intelligence

Topic 3

Problem Solving and Search

- ◇ Problem-solving and search
- ◇ Search algorithms
- ◇ Uninformed search algorithms
  - breadth-first search
  - uniform-cost search
  - depth-first search
  - iterative deepening search
  - bidirectional search

Reading: Russell and Norvig, Chapter 3

1. Problem Solving and Search

Seen that an intelligent agent has:

- knowledge of the *state* of the "world"
- a notion of how actions or *operations* change the world
- some *goals*, or states of the world, it would like to bring about

Finding a sequence of operations that changes the state of the world to a desired goal state is a *search problem* (or basic *planning problem*).

Search algorithms are the cornerstone of AI

In this section we see some examples of how the above is encoded, and look at some common *search strategies*.

state — description of the world at a particular time
— impossible to describe the whole world
— need to abstract those attributes or properties that are important.

Examples

Example 1: Say we wish to drive from Arad to Bucharest. First we "discretise" the problem:

states — map of cities + our location

operators — drive from one city to the next

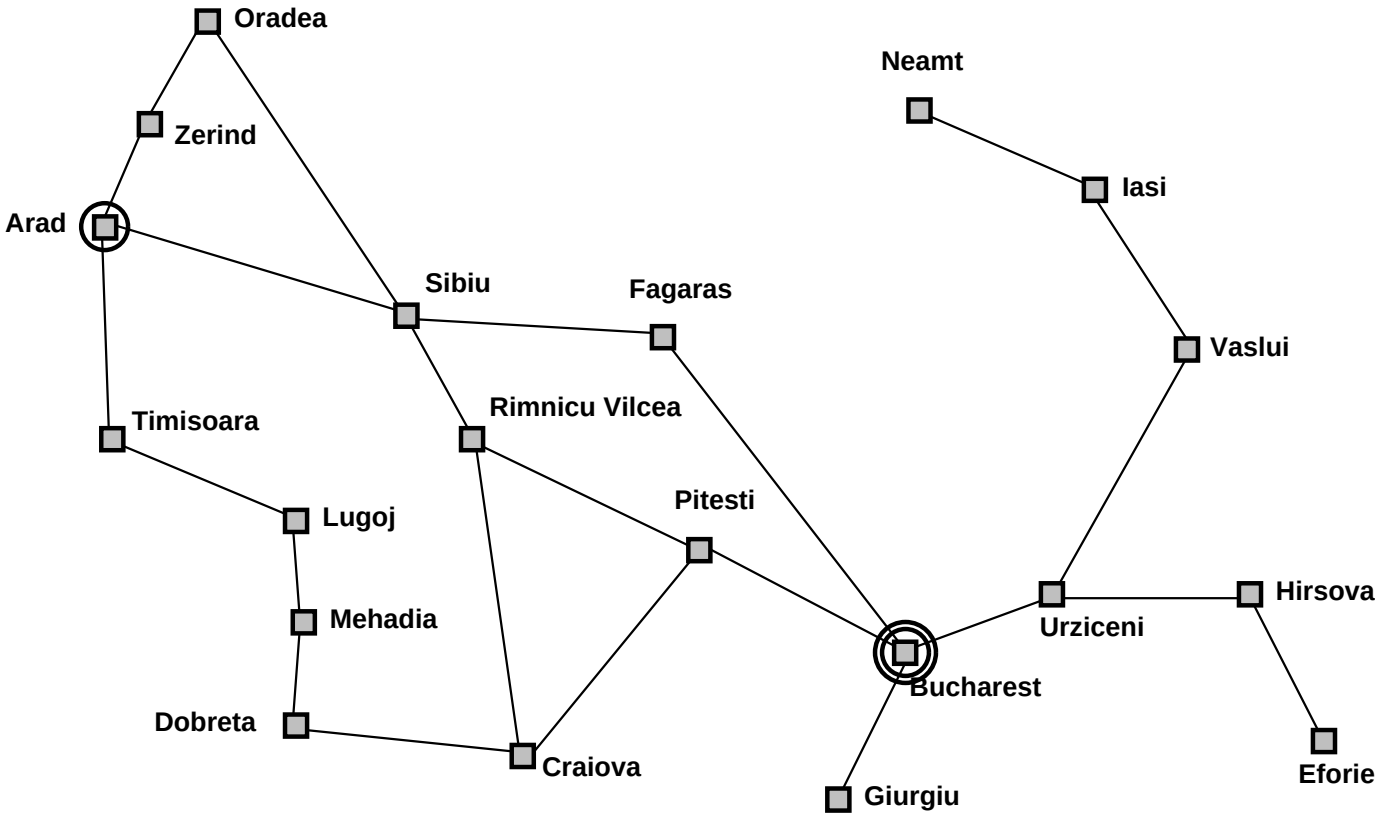
start state — driver located at Arad

goal state — driver located at Bucharest

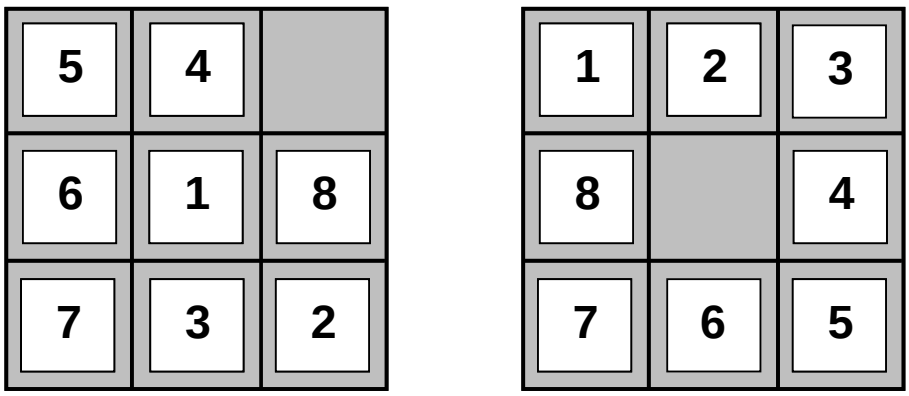
Find solution:
sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

1.1 States, Operators, Graphs and Trees

The states and operators form a *graph* — states form *nodes*, operators form *arcs* or *edges*...



Example 2: The *eight puzzle*



states — description of the positions of the numbered squares

operators — some alternatives. . .

(a) move a numbered square to an adjacent place, or

(b) move blank left, right, up or down — far fewer operators

Example 3: *Missionaries and Cannibals*

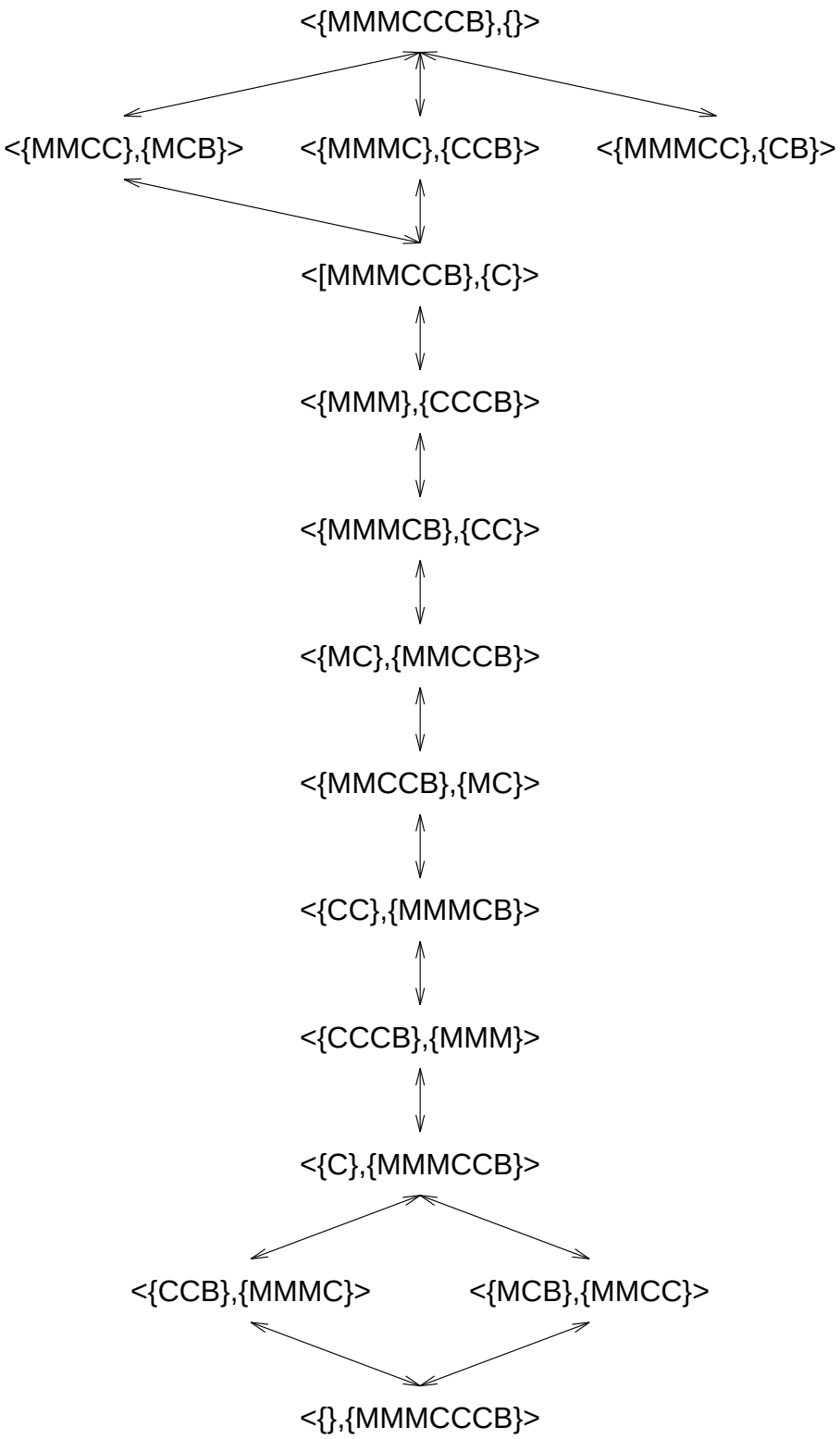
start state — 3 missionaries, 3 cannibals, and a boat that holds two people, on one side of river

goal state — all on other side

states — description of legal configurations (ie. where no-one gets eaten) of where the missionaries, cannibals, and boat are

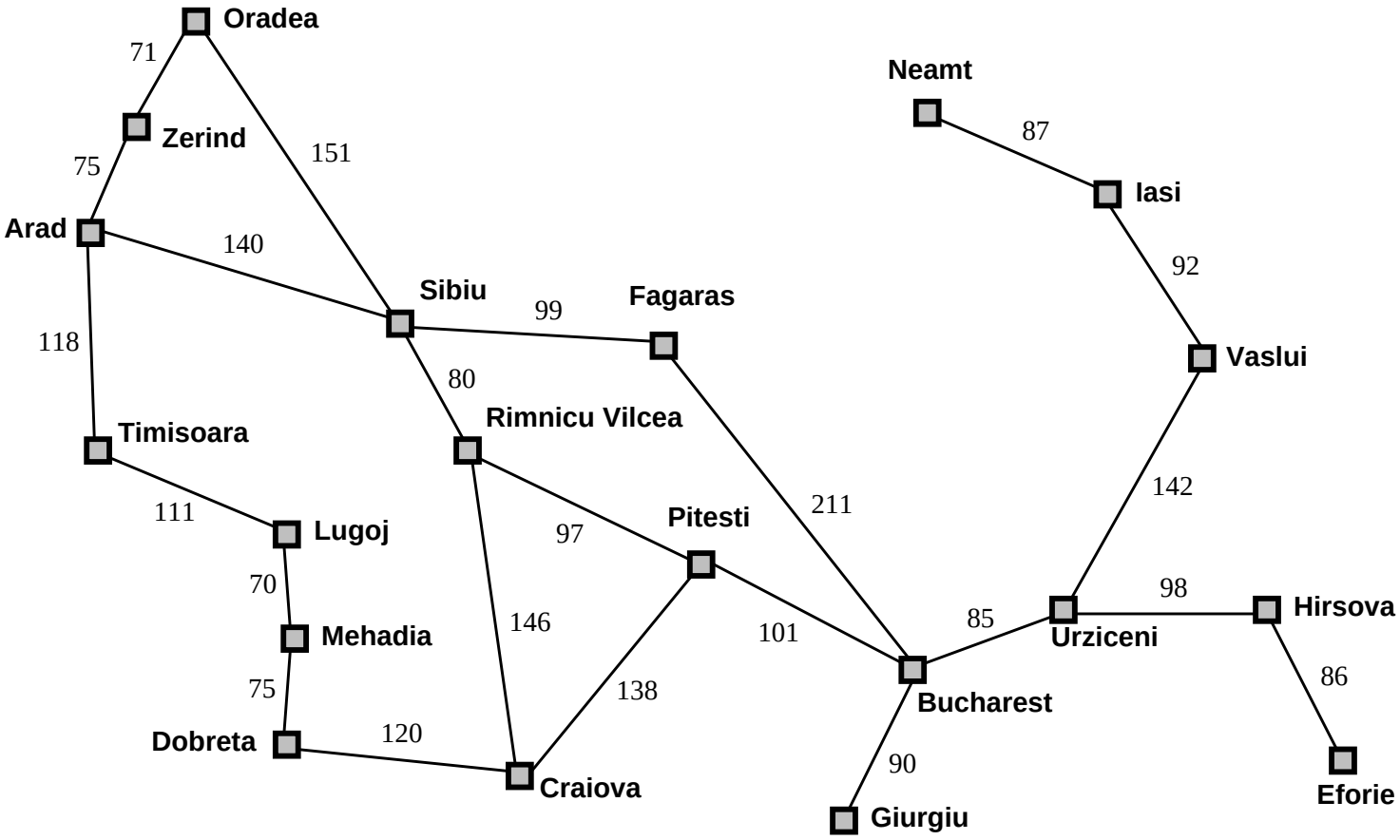
operators — state changes possible using 2-person boat

1.1 States, Operators, Graphs and Trees



1.2 Operator costs

Graphs may also contain the *cost* of getting from one node to another (ie. associated with each operator, or each arc).



cost of path = sum of the costs of arcs making up the path

1.2 Operator costs

Usually concerned not just with finding a path to the goal, but finding a cheap path.

shortest path problem — find the cheapest path from a start state to a goal state

Where operator costs are not given, all operators are assumed to have unit cost.

1.3 Problem formulation

A *problem* is defined by four items:

initial state e.g., "at Arad"

operators (or *successor function* $S(x)$)
e.g., Arad $\rightarrow$ Zerind Arad $\rightarrow$ Sibiu etc.

goal test, can be
explicit, e.g., $x =$ "at Bucharest"
implicit, e.g., "world peace"

path cost (additive)
e.g., sum of distances, number of operators executed, etc.

A *solution* is a sequence of operators leading from the initial state to a goal state.

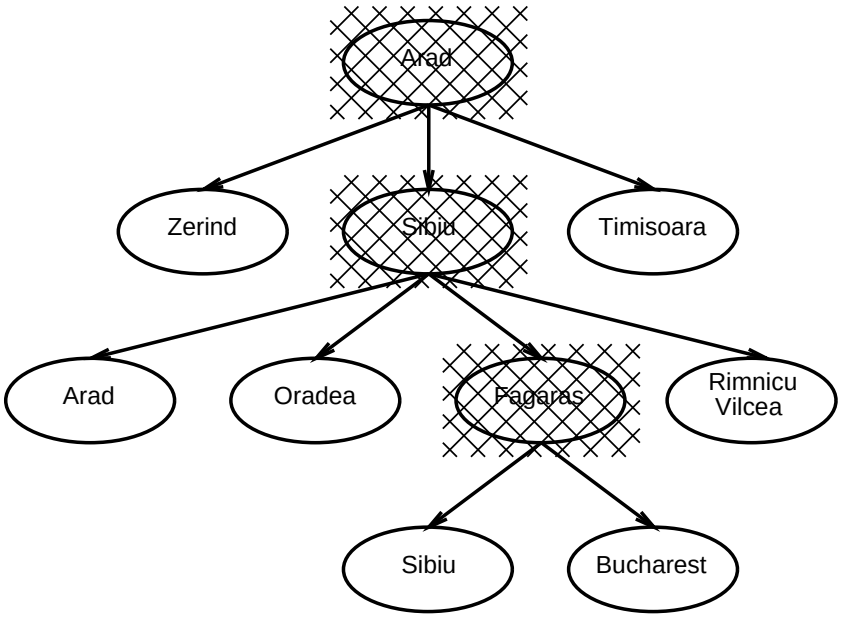
2. Search algorithms

Basic idea:

offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. *expanding* states)

```
function GENERAL-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion
            then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state
            then return the corresponding solution
        else expand node and add resulting nodes to the search tree
    end
```

2.1 General search example



General search algorithm

Given a *start state* $s_0 \in S$, a *goal function* $g(s) \rightarrow \{true, false\}$, and an (optional) *terminal condition* $t(s) \rightarrow \{true, false\}$:

Initialise a set $U = \{s_0\}$ of *unvisited* nodes containing just the start state, and an empty set $V = \{\}$ of *visited* nodes.

1. If U is empty halt and report no goal found.
2. Select, according to some (as yet undefined) *strategy*, a node s from U.
3. (Optional) If $s \in V$ discard s and repeat from 1.
4. If $g(s) = true$ halt and report goal found.
5. (Optional) If $t(s) = true$ discard s and repeat from 1.
6. Otherwise move s to the set V, and add to U all the nodes reachable from s. Repeat from 1.

Step 3 is an *occurs check* for cycles.

- Some search strategies will still work without this
  $\Rightarrow$ trade-off — work to check if visited vs work re-searching same nodes again.
- Others may cycle forever.

With these cycles removed, the graph becomes a *search tree*.

2.3 Comparing search strategies

The *search strategy* is crucial — determines in which order the nodes are expanded. Concerned with:

completeness — does the strategy guarantee finding a solution if there is one
optimality — does it guarantee finding the best solution
time complexity — how long does it take
space complexity — how much memory is needed to store states

Time and space complexity are often 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 ∞)

2.4 Implementation of search algorithms

Many strategies can be implemented by placing unvisited nodes in a *queue* (Step 6) and always selecting the next node to expand from the front of the queue (Step 2)

$\Rightarrow$ the way the children of expanded nodes are placed in the queue determines the search strategy.

Many different strategies have been proposed. We'll look at some of the most common ones. . .

3. Uninformed search strategies

Uninformed 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

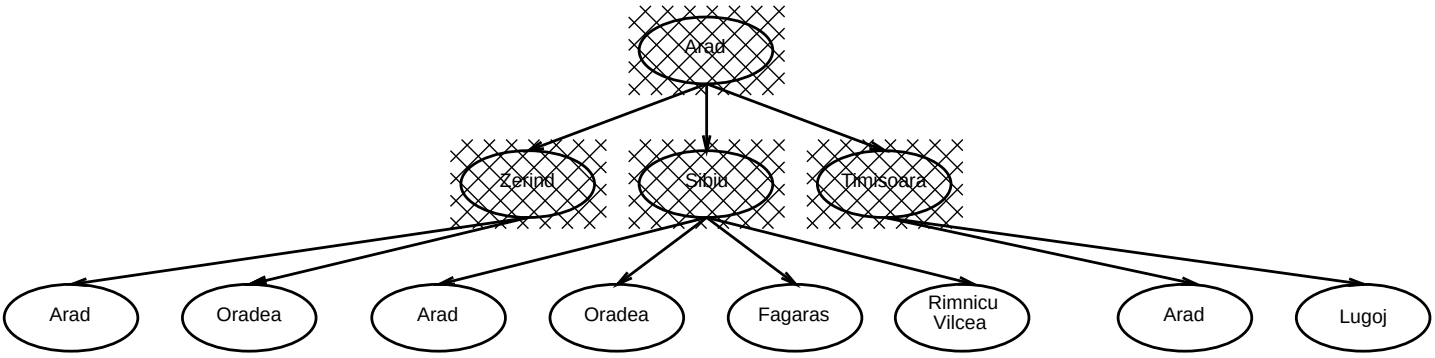
Later we will look at *informed* stategies that use additional information.

3.1 Breadth-first search

Expand shallowest unexpanded node

Implementation:

$\textsc{QueueingFn}$ = put successors at end of queue



$\Rightarrow$ *expand all nodes at one level before moving on to the next*

Complete? Yes (if b is finite)

Time? $O(1 + b + b^2 + b^3 + \ldots + b^d) = O(b^d)$, i.e., exponential in d

Space? $O(b^d)$ (all leaves in memory)

Optimal? *Yes (if cost = 1 per step); not optimal in general*

Space is the big problem; can easily generate nodes at 1MB/sec so 24hrs = 86GB.

Good example of computational explosion...

Assume branching factor of 10, 1000 nodes/sec, 100 bytes/node.

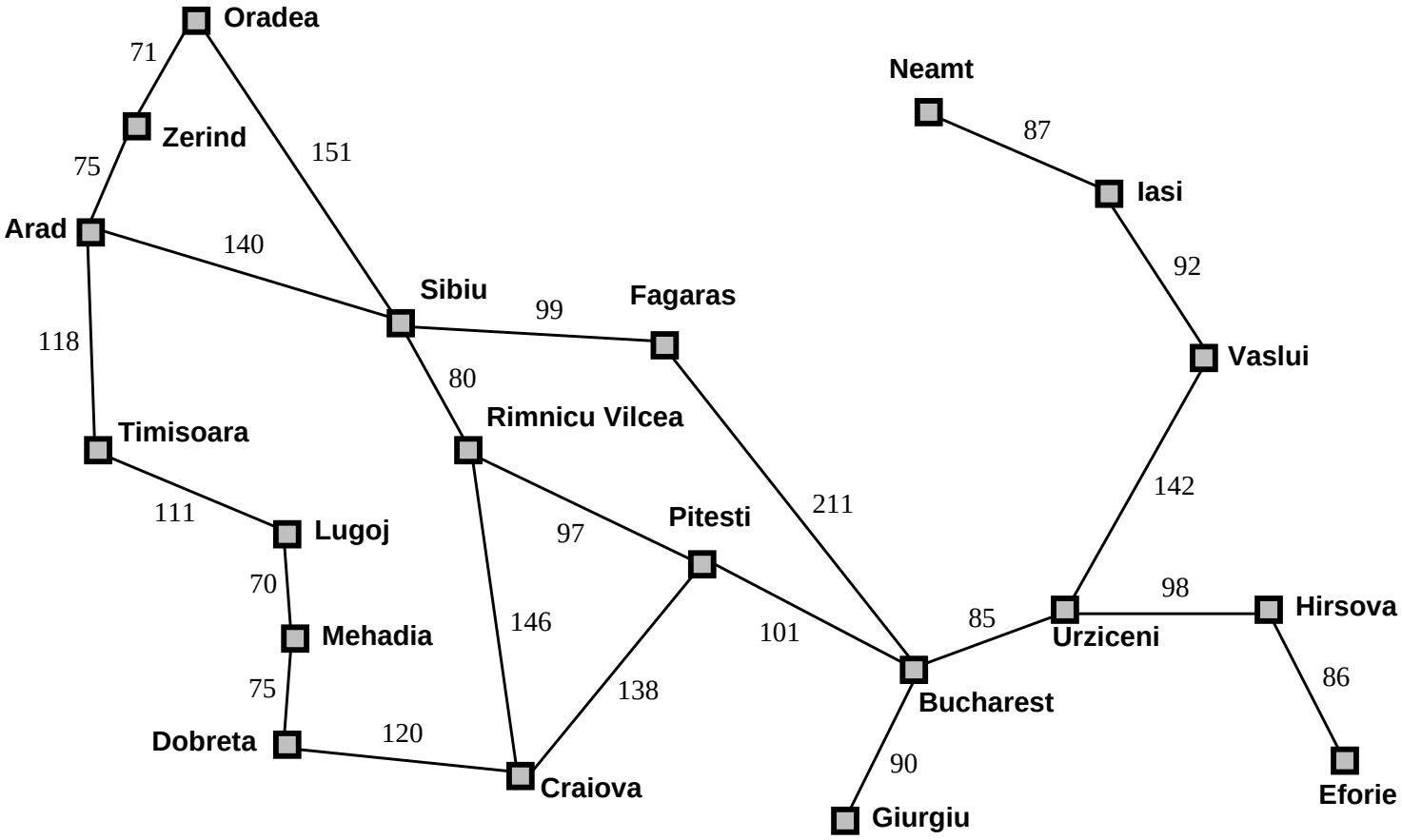
Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Welcome to AI!

3.2 Uniform-cost search

Problem: Varying cost operations

eg. Romania with step costs in km...

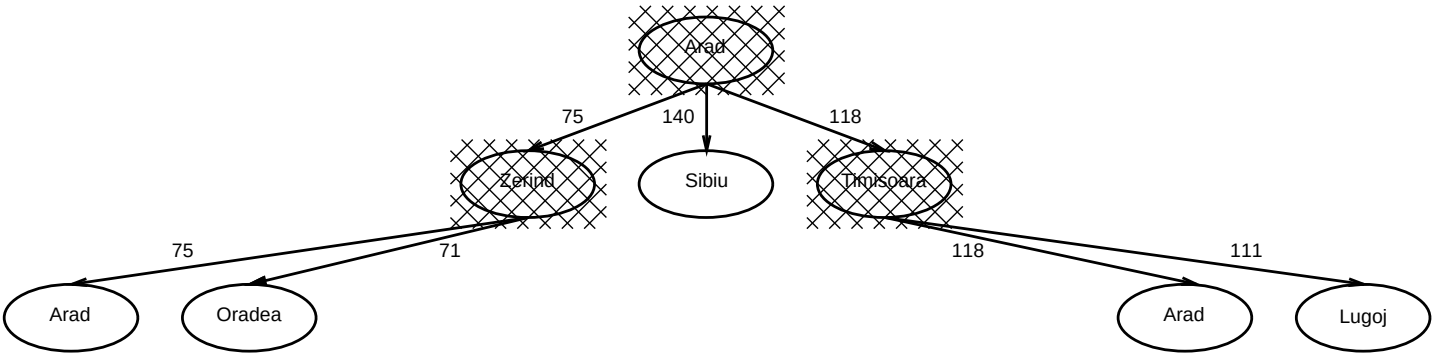


3.2 Uniform-cost search

Expand least total cost unexpanded node

Implementation:

QUEUEINGFN = insert in order of increasing path cost



3.2 Uniform-cost search

Complete? Yes (if step cost ≥ 0)

Time? # of nodes with path cost $g \leq$ cost of optimal solution

Space? # of nodes with $g \leq$ cost of optimal solution

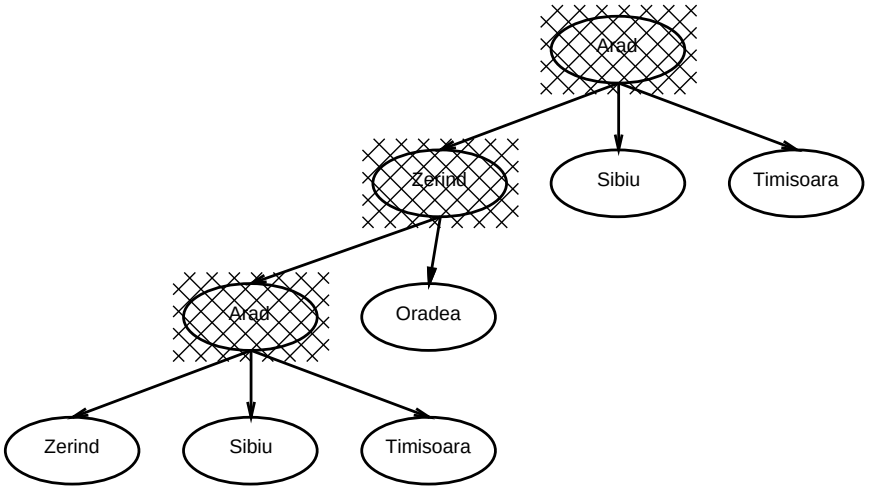
Optimal? *Yes (if step cost ≥ 0)*

3.3 Depth-first search

Follow one path until you can go no further, then backtrack to last choice point and try another alternative.

Implementation:

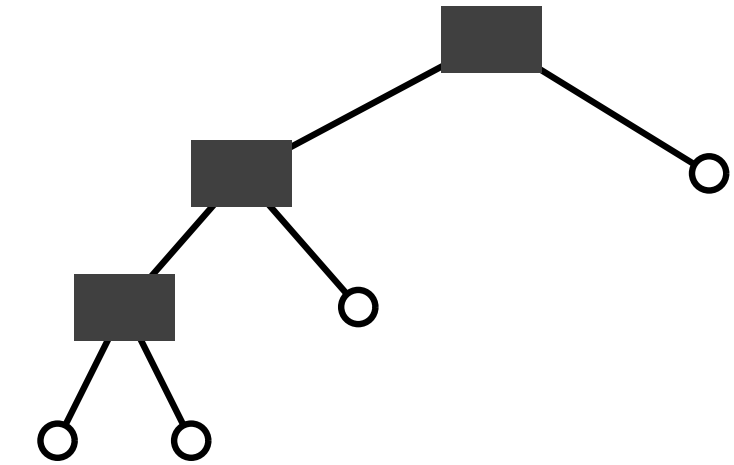
QUEUEINGFN = insert successors at front of queue
(or use *recursion* — "queueing" performed automatically by internal stack)

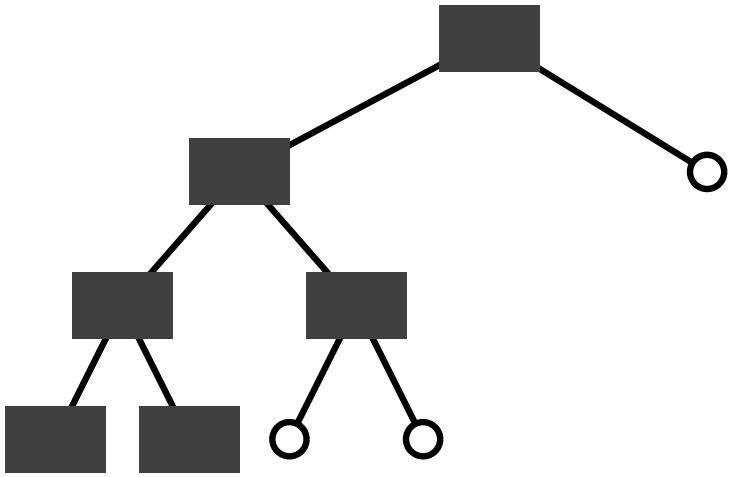


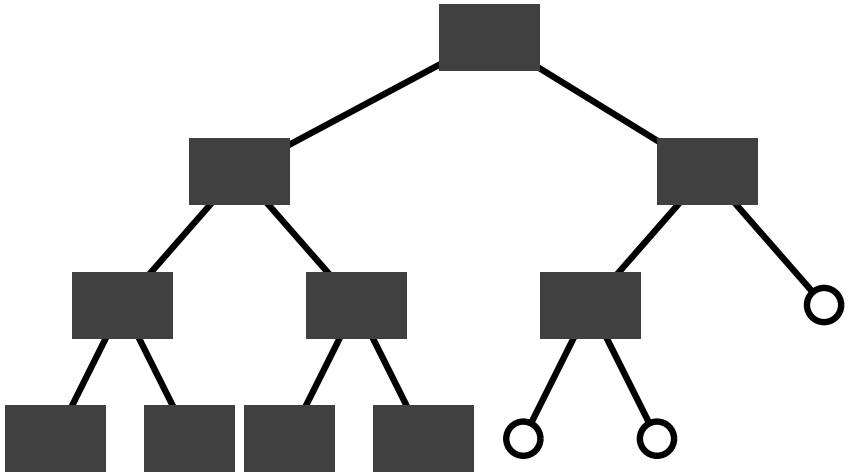
Occurs check or terminal condition needed to prevent infinite cycling.

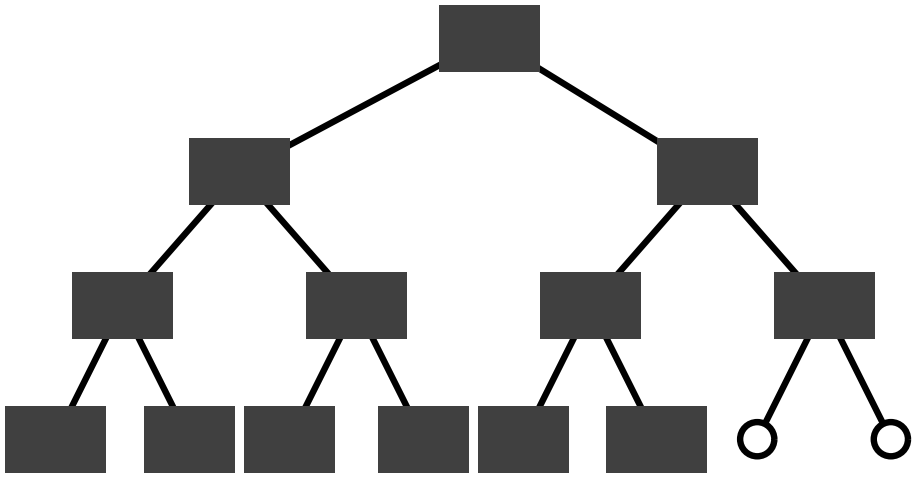
3.3 Depth-first search

Finite tree example...









3.3 Depth-first search

Complete? No: fails in infinite-depth spaces. — complete for finite tree (in particular, require cycle check).

Time? $O(b^m)$.

— may do very badly if m is much larger than d

— may be much faster than breadth-first if solutions are dense

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

Optimal? No

Space performance is big advantage.

Time, completeness and optimality can be big disadvantages.

3.4 Depth-limited search

= depth-first search with depth limit l

Implementation:

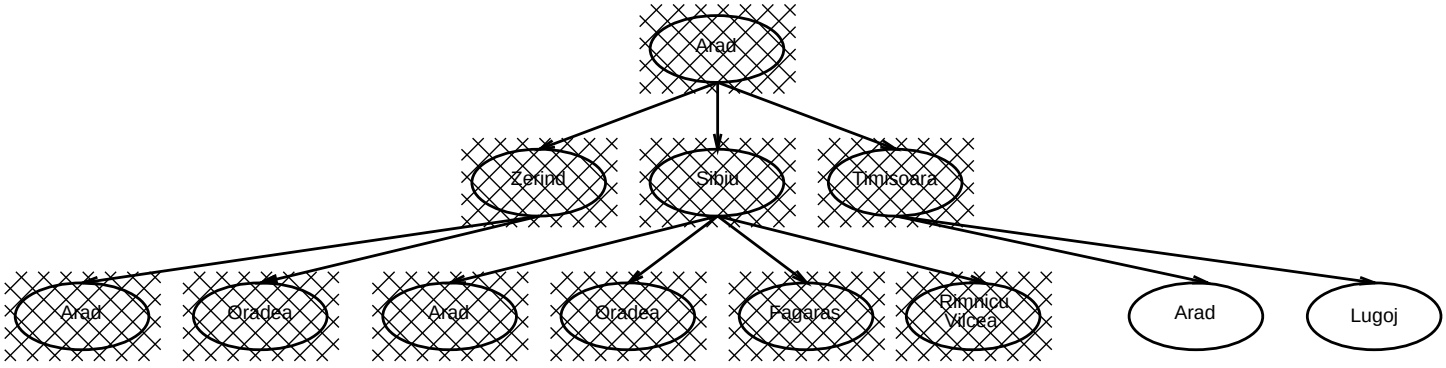
Nodes at depth l have no successors.
Can be implemented by our *terminal condition*.

Sometimes used to apply depth-first to infinite (or effectively infinite) search spaces. Take "best" solution found with limited resources. (See Game Playing...)

Also in...

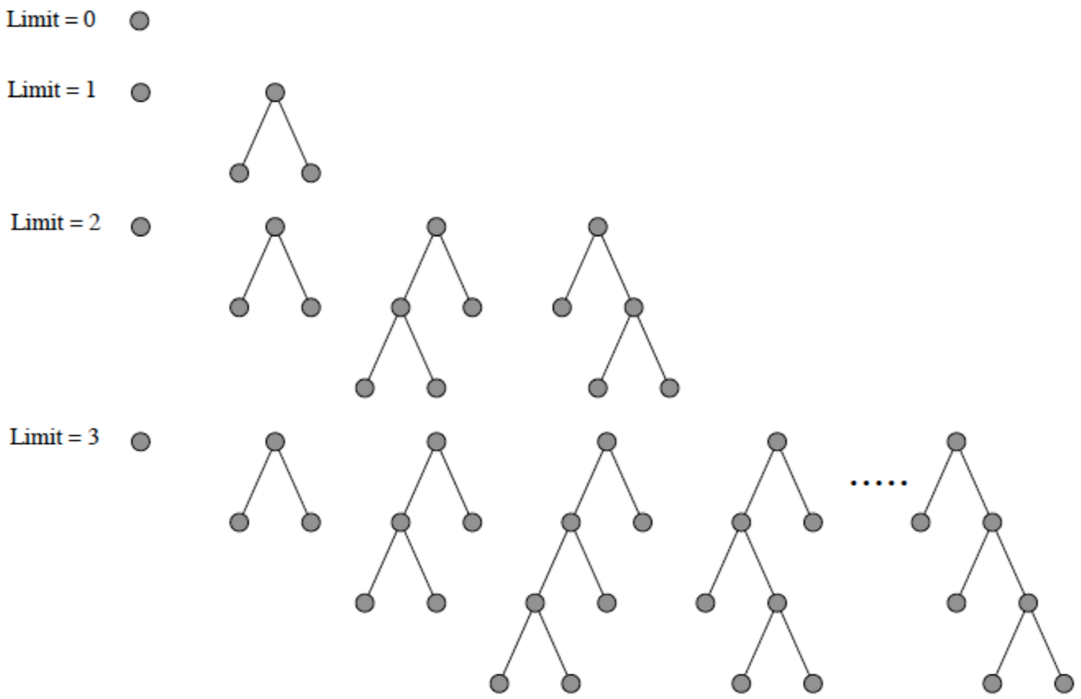
3.5 Iterative deepening search

"Probe" deeper and deeper (often bounded by available resources).



3.5 Iterative deepening search

Summary view. . .



3.5 Iterative deepening search

Complete? Yes

Time? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d \rightarrow O(b^d)$

Space? $O(bd)$

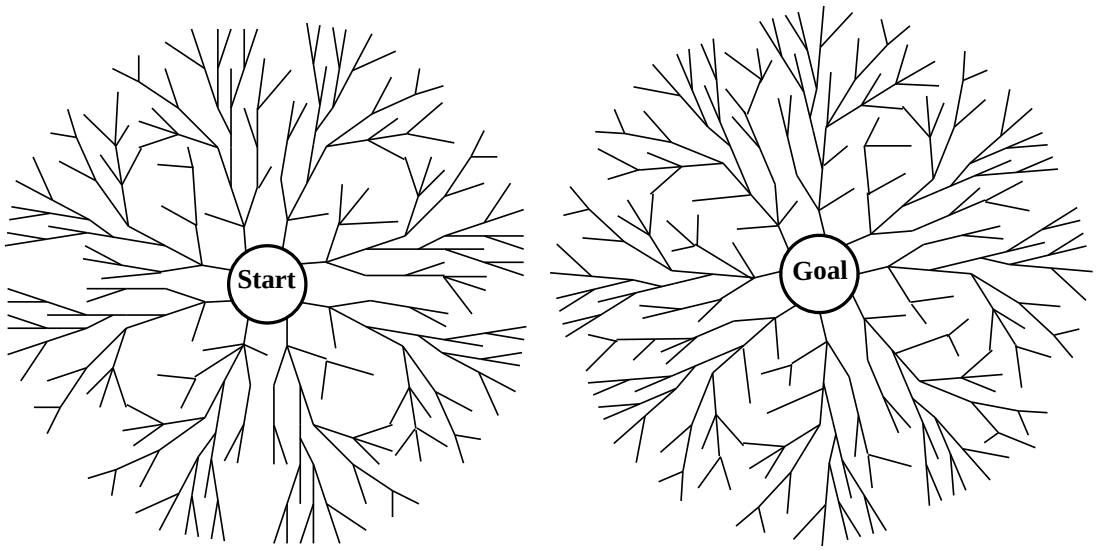
Optimal? *Yes*, *if step cost = 1*

Can also be modified to explore uniform-cost tree

How do the above compare with

- breadth-first?
- depth-first?

3.6 Bidirectional search



Tends to expand fewer nodes than unidirectional, but raises other difficulties — eg. how long does it take to check if a node has been visited by other half of search?

4. Summary

Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored.

- Abstraction reveals states and operators.
- Evaluation by goal or utility function.
- Strategy implemented by queuing function (or similar).

Variety of uninformed search strategies...

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Iterative deepening search uses only linear space and not much more time than other uninformed algorithms.

The End