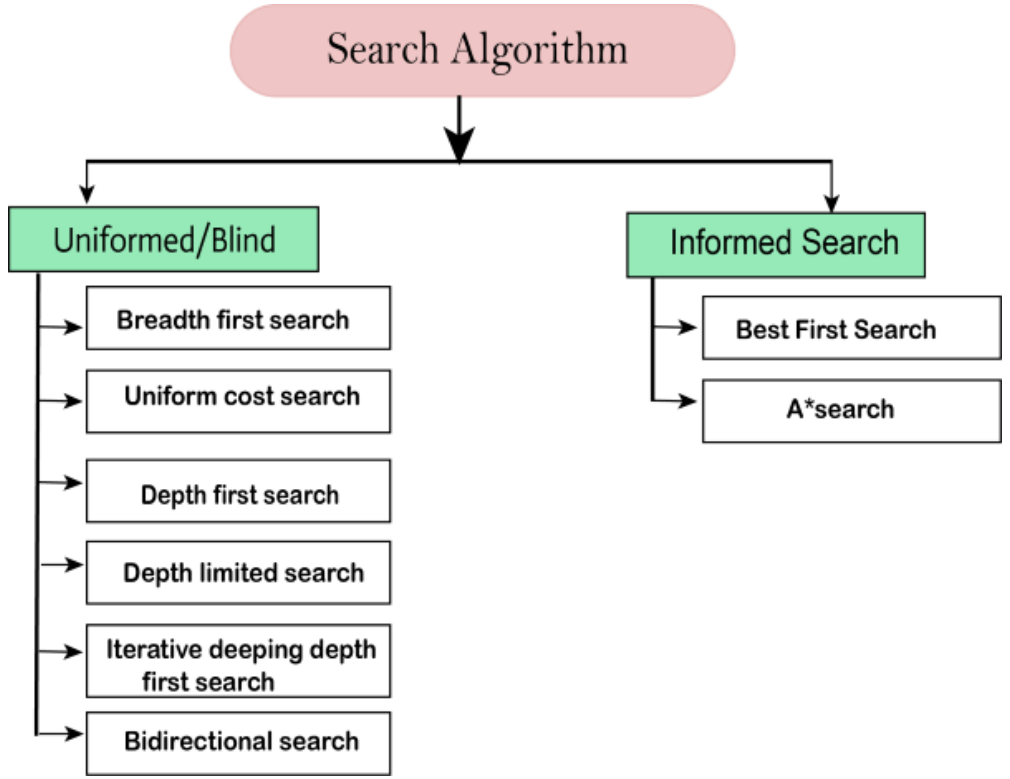
AI101

Lecture 3: Uninformed and Informed Search



Recap

Last Week: AI Systems

- An AI system can be defined as the study of rational agents and their environments.
- An environment is the surrounding of an agent and defines where the agent operates.
- Environments are defined by their characteristics
- Rational agents sense, decide and act in an environment
- They try to optimize a specific measure
- There are different types of agents
- There are multiple ways of how to make agents intelligent

See the AI timeline and more at www.aaai.org/AIlandscape

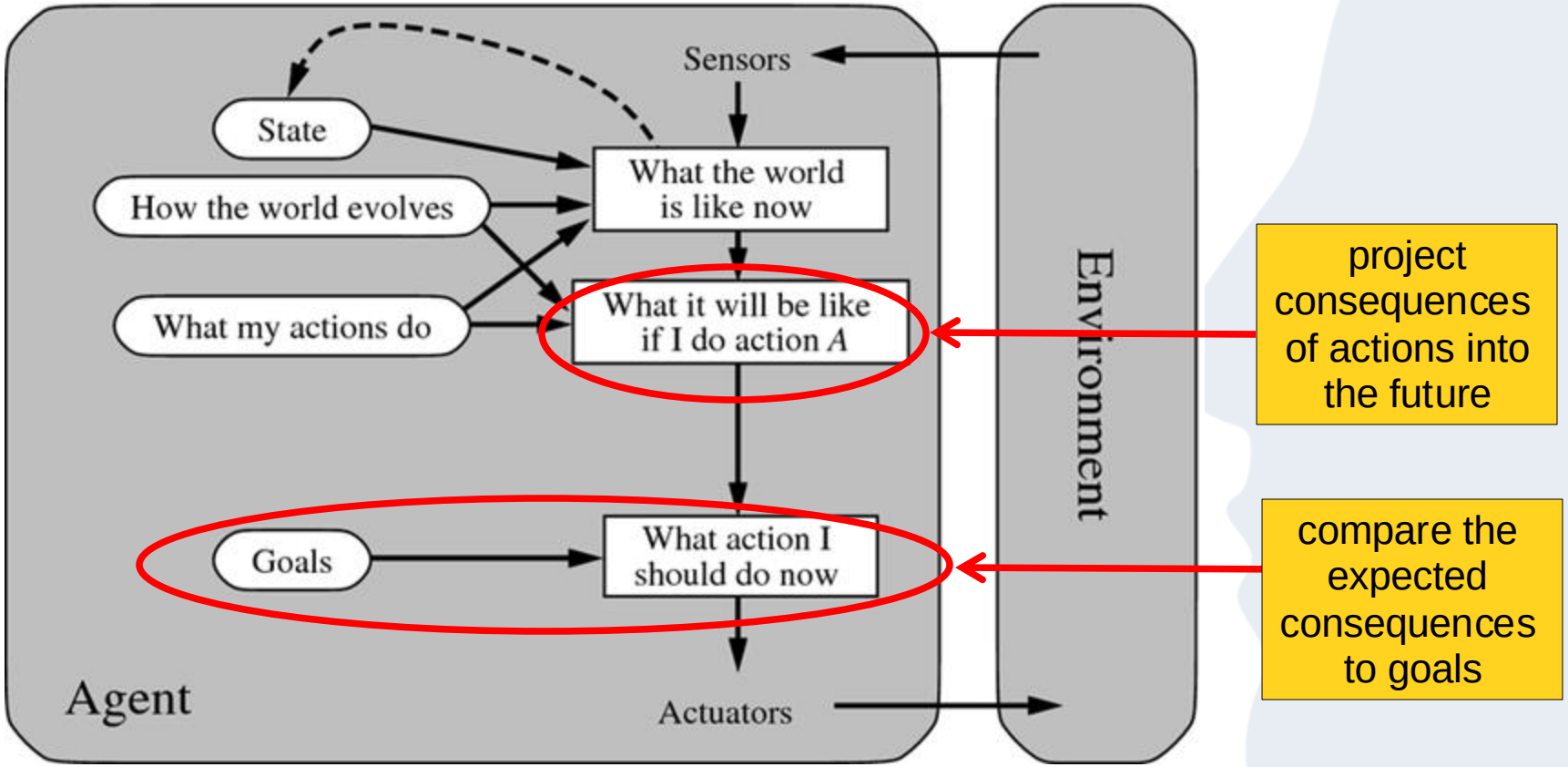
Today

“Imagine you are in a foreign city and want to get from your hotel to the beach with public transport. (without a smartphone but a map)

What do you do?”

- Problem Definitions
- Tree Search Algorithms
- Search Strategies
- Uninformed Search Algorithm
- Heuristics
- Informed Search Algorithm

You have a goal, so your are a goal-based agent



OK, but how do we solve an problem (in general)?

How to solve an problem
All about Problem-Solving Agents

- Problem-solving agents are result-driven
  - Always focus on satisfying its goals, i.e. solve the problem
- Problems are often given in a human understandable way,
  - We need to reformulate the problem for our agent.
- Problem-solving agents employ algorithms to develop/find solutions

Steps to formulate a solvable problem:

1. **Formulate the goal**
2. **Formulate the problem given the goal** (see next slide)

Some key terminology to describe problems

The State Space / States

A state describes a possible situation in our environment. The state space is a set of all possible situations (states).

Transition / Action

Transitions describe possible actions to take between one state and another. We only count direct transitions between two states (single actions).

Costs

Often transitions aren’t alike and differ. We express this by adding a “cost” to each action. Often the goal in search algorithms is to minimize the coast to reach the goal.

How to solve a problem
Components to formulate a problem

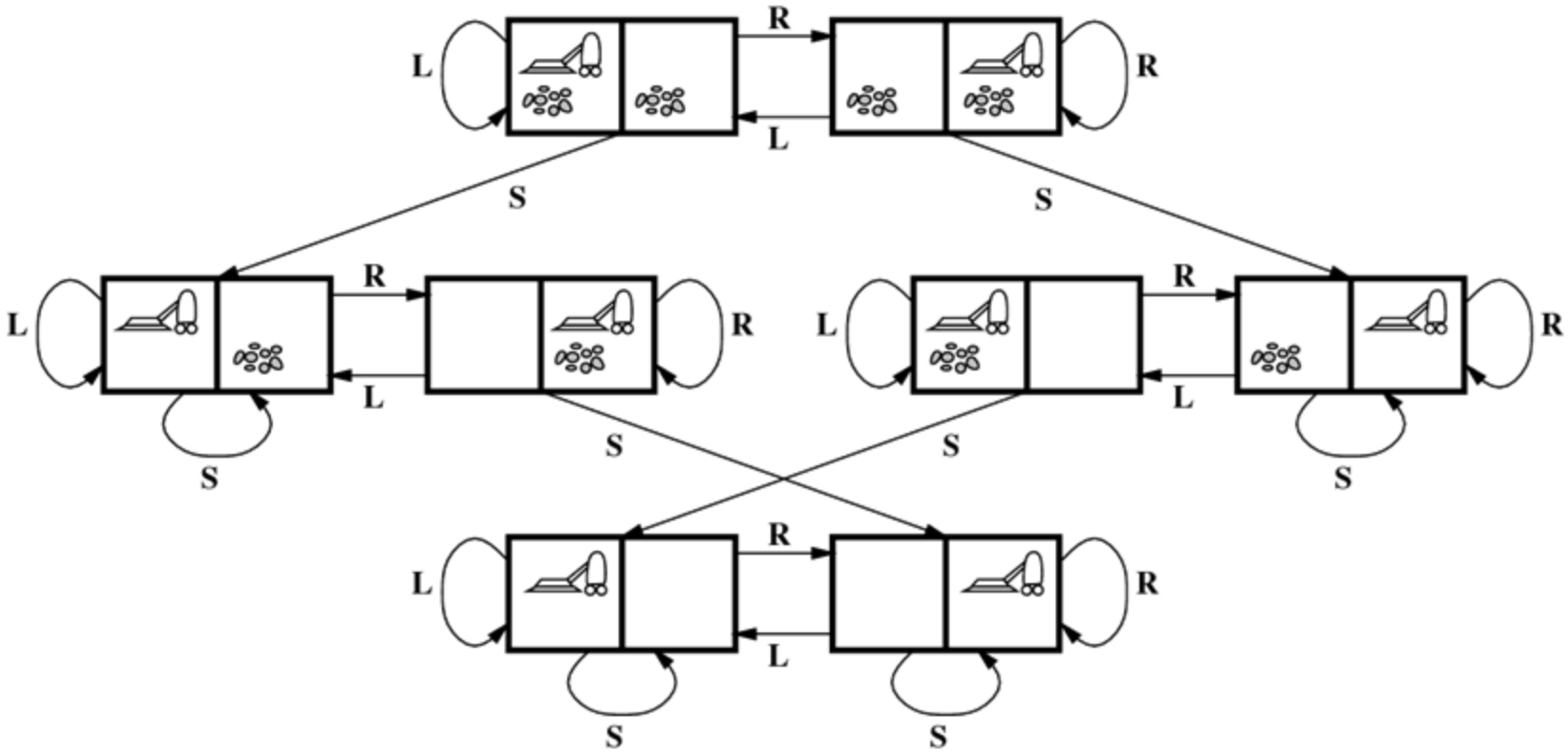
A single state problem is defined in 4 items:

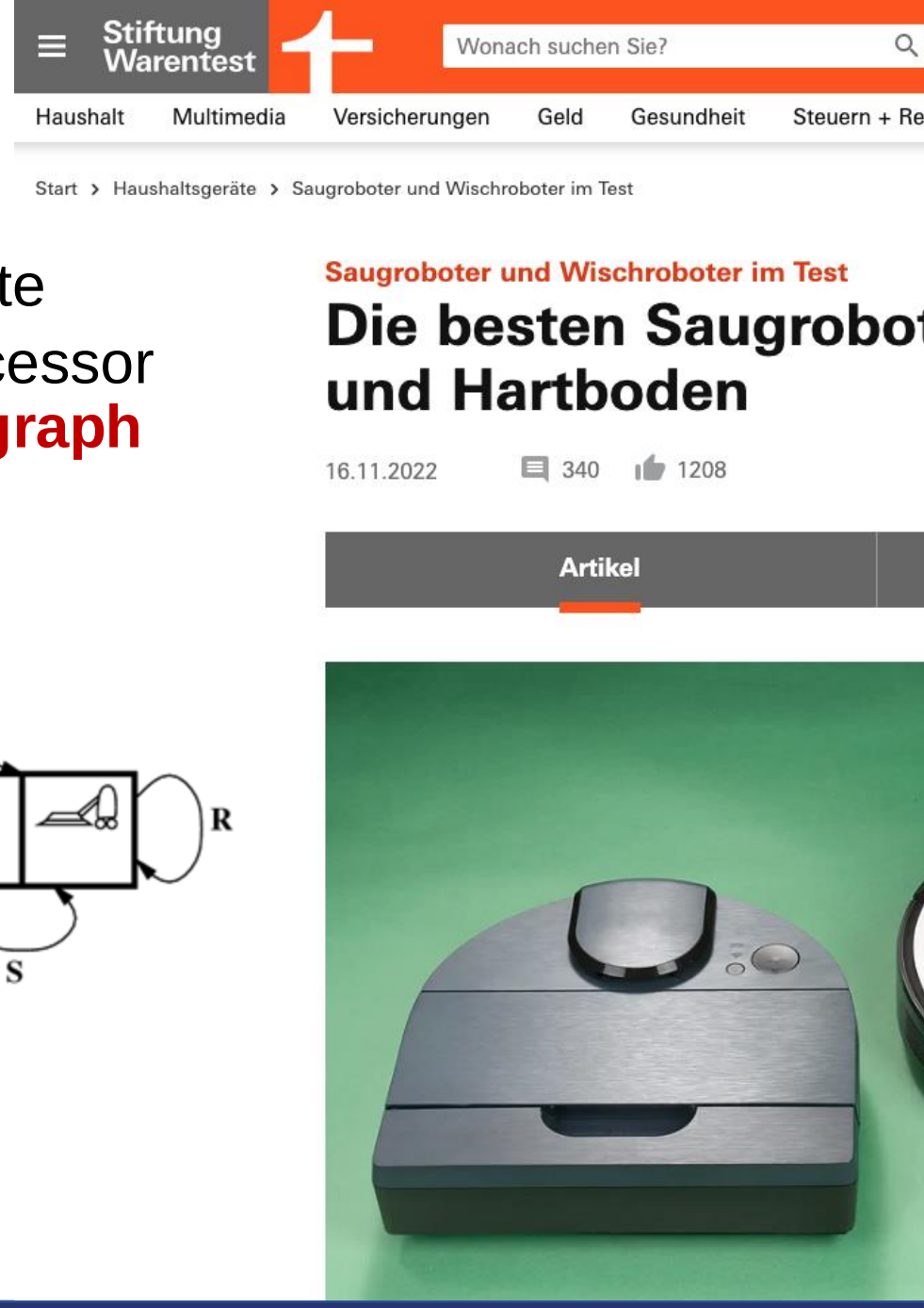
1. **State Space and Initial State**
   - Description of all possible states and the initial environment as state
2. **Descriptions of Actions**
   - Typically a function that maps a state to a set of possible actions in this state
3. **Goal Test**
   - Typically a function to test if the current state fulfils the goal definition
4. **Costs**
   - A cost function that maps actions to its cost
   - An easy way is to have additive costs (sum of costs for all actions taken)

The State Space of a Problem

State Space

- the set of all states reachable from the initial state
- implicitly defined by the initial state and the successor function, so we have a graph, the **state-space graph**





The State Space of a Problem (some more terminology)

State Space

- the set of all states reachable from the initial state
- implicitly defined by the initial state and the successor function, so we have a graph, the **state-space graph**

Path

- a sequence of states connected by a sequence of actions

Solution

- a path that leads from the initial state to a goal state

Optimal Solution

- solution with the minimum path cost

Another Example: Finding a Route through Romania

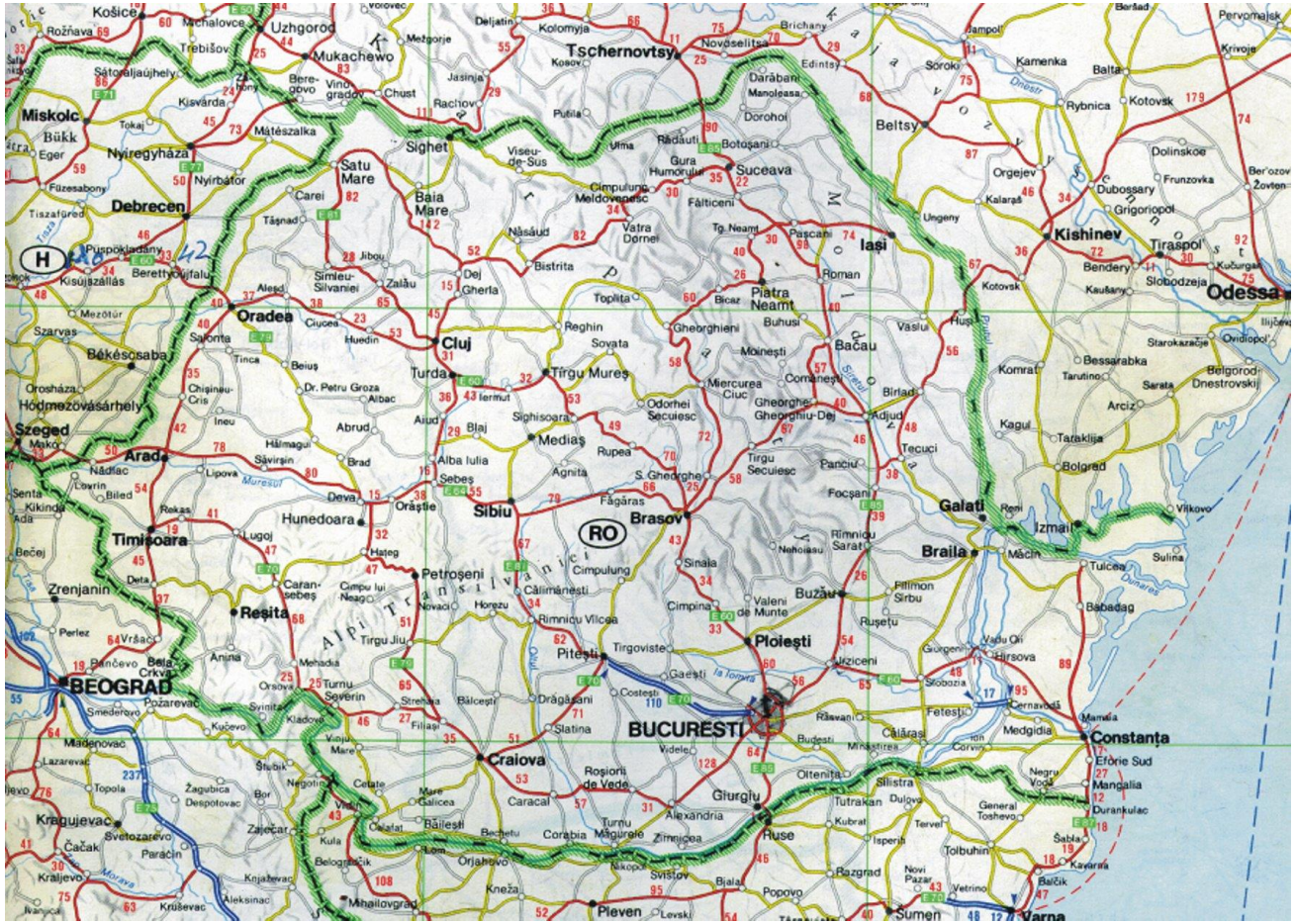
- **Problem**: You are currently in Arad and need to find the fastest way to Bucharest for your flight tomorrow.
- **Environment**: see next slides

- **Initial State**: Arad
- **Description of Actions**: A function defining the cities that can be reached from the given city
- **Goal**: be in Bucharest
- **Costs**: A function that gives you the costs of traveling from one city to another, based on Speed and/or other criteria.

- **Solution**: Sequence of cities/actions to get to Bucharest from Arad

Another Example:
Finding a Route through Romania
Define the State Space/Environment

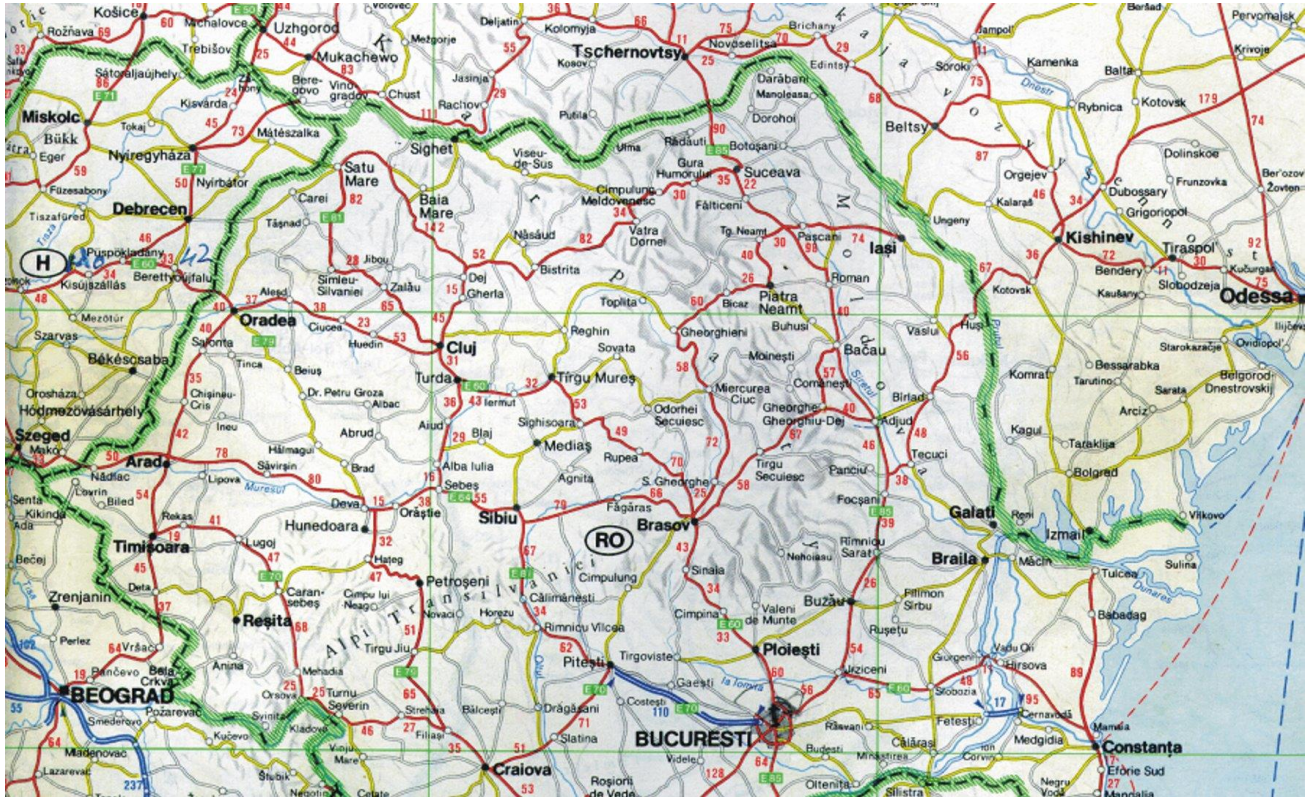
Real world is absurdly complex!



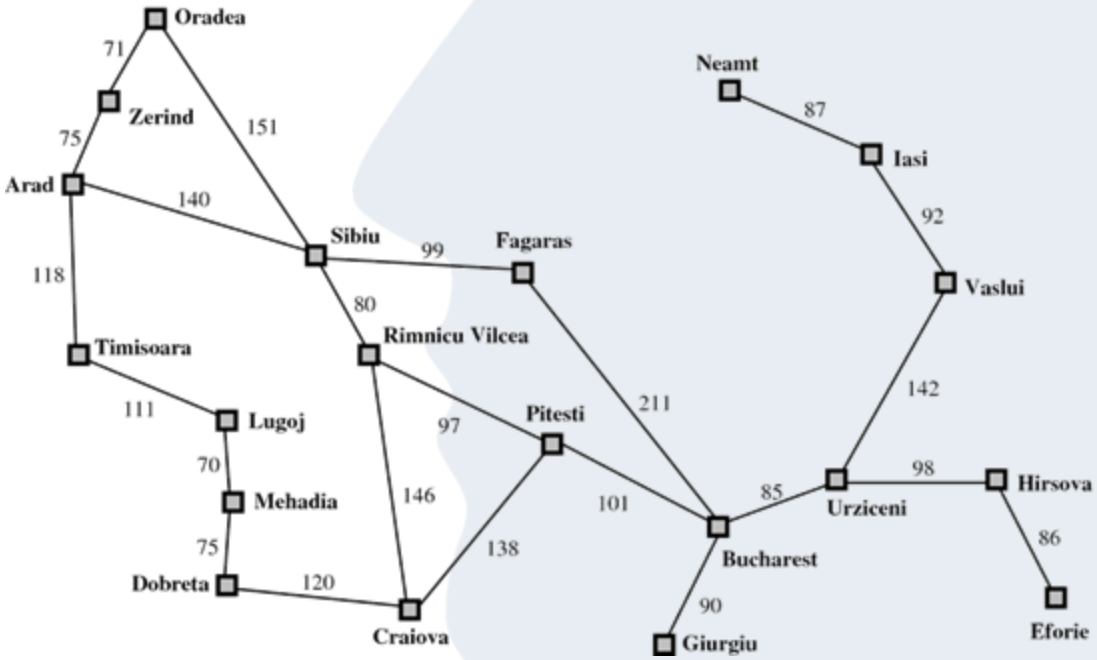
Another Example: Finding a Route through Romania

Define the State Space/Environment

A way to solve real world complexity is **abstraction** in order to gain problem understanding

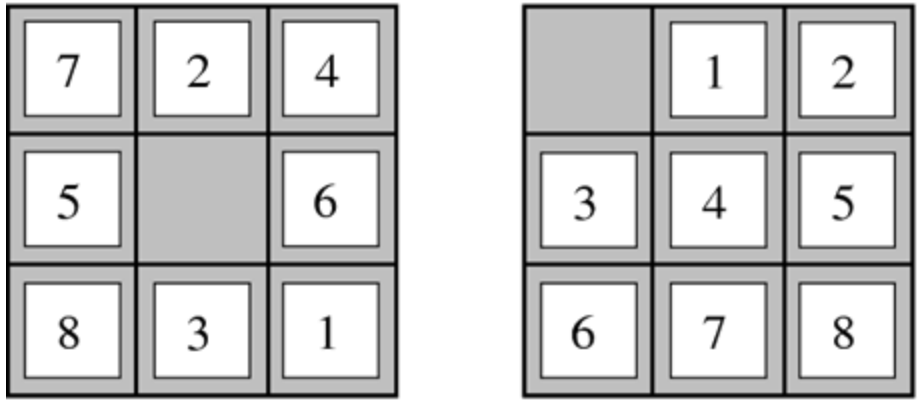


Real world complexity



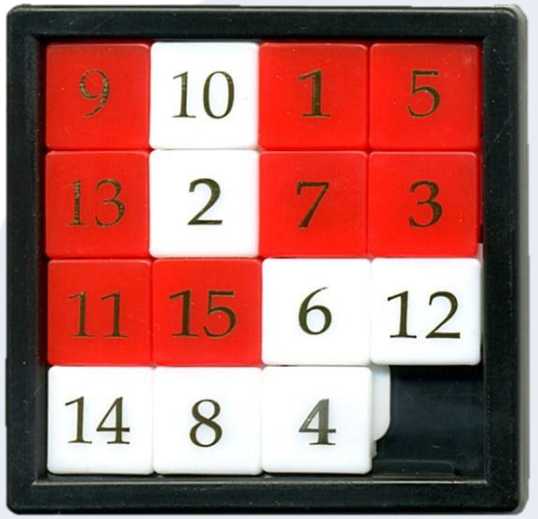
Abstraction

Yet Another Example: The 8-puzzle



Start State

Goal State



States?

- Location of tiles
  - ignore intermediate positions during sliding

Goal test?

- Situation corresponds to goal state

Path cost?

- Number of steps in path (each step costs 1)

Actions?

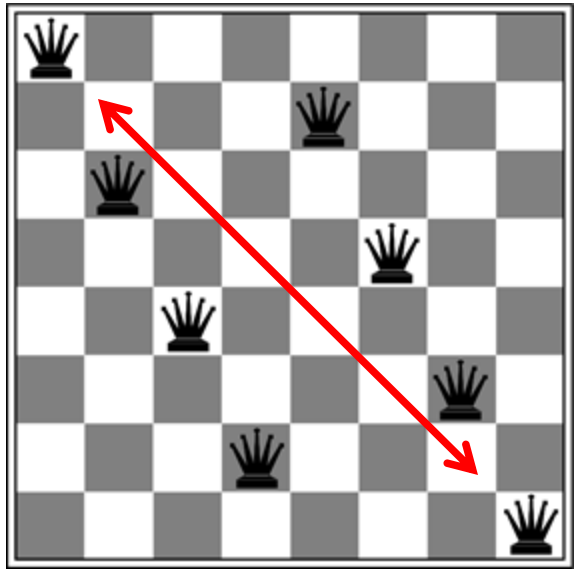
- Move blank tile (left, right, up, down)
  - easier than having separate moves for each tile
  - ignore actions like unjamming slides if they get stuck

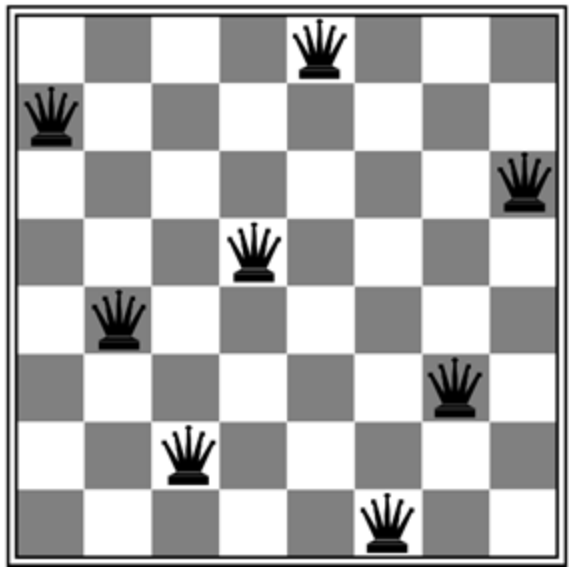
Yet Another Example: The 8-Queens Problem

the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other

Yet Another Example: The 8-Queens Problem

conflict





no conflict

the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other

States?

- any configuration of 8 queens on the board

Goal test?

- no pair of queens can capture each other

Actions?

- move one of the queens to another square

Path cost?

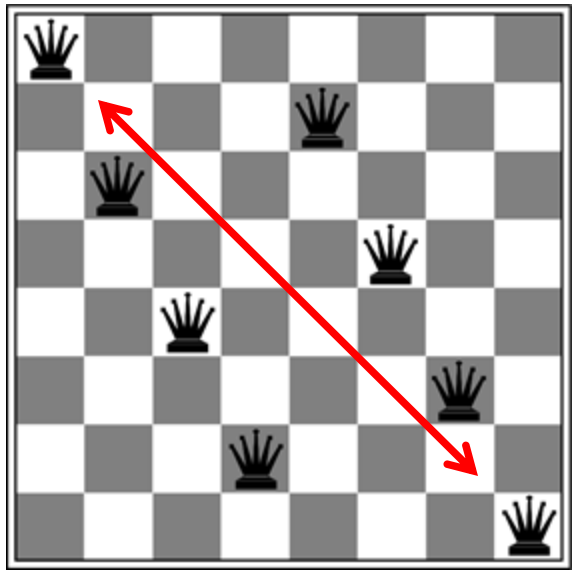
- not of interest here

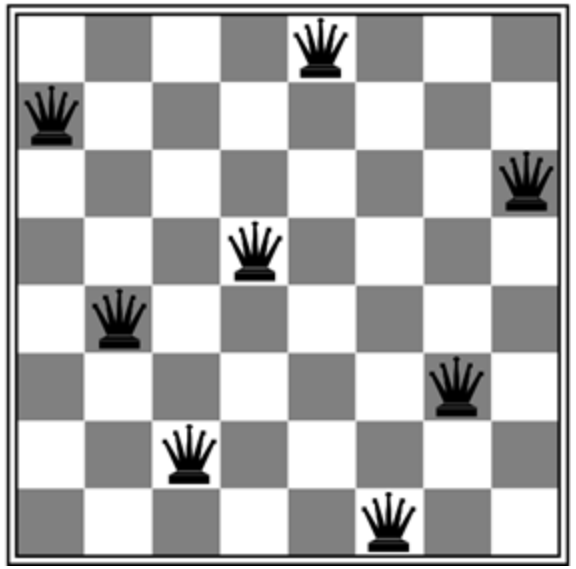
inefficient complete-state formulation

$\rightarrow 64 \cdot 63 \cdot ... \cdot 57 \approx 3 \cdot 10^{14}$ states

Yet Another Example: The 8-Queens Problem

conflict





no conflict

the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other

States?

- *n* non-attacking queens in the left *n* columns

Goal test?

- no pair of queens can capture each other

Actions?

- add queen in column *n* + 1
- without attacking the others

Path cost?

- not of interest here

more efficient incremental formulation
→ only 2057 states

How to solve a problem

Now that we know how to describe problems,
we can think about how to search for a solution

Planning Problem

A planning problem is one in which we have an initial state and want to transform it into a desired goal considering future actions and outcomes of them.

Search

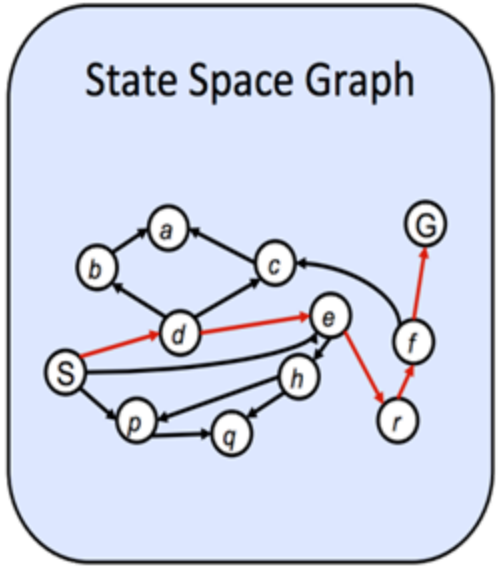
The process of finding the (optimal) solution for such a problem in form of a sequence of actions.

Tree Search Algorithms

https://inst.eecs.berkeley.edu/~cs188/fa18/assets/notes/n1.pdf

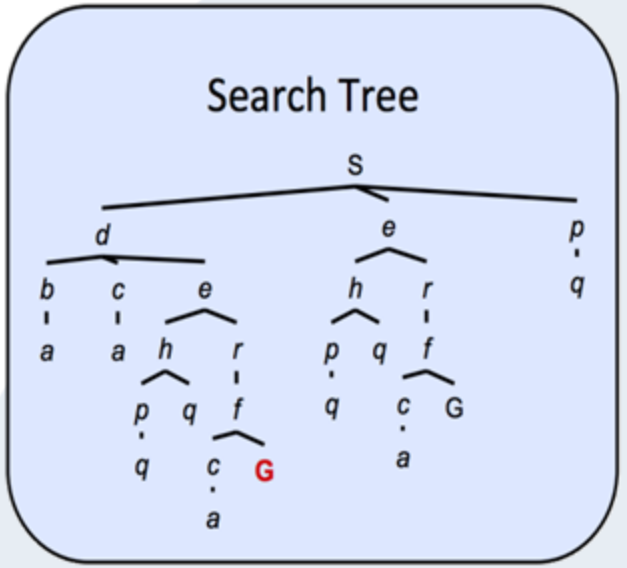
Idea: Treat the state-space graph as a tree

- Can use simulated iterative exploration of the state space
- Needs a strategy to determine which node is expanded next



Each NODE in in the search tree is an entire PATH in the state space graph.

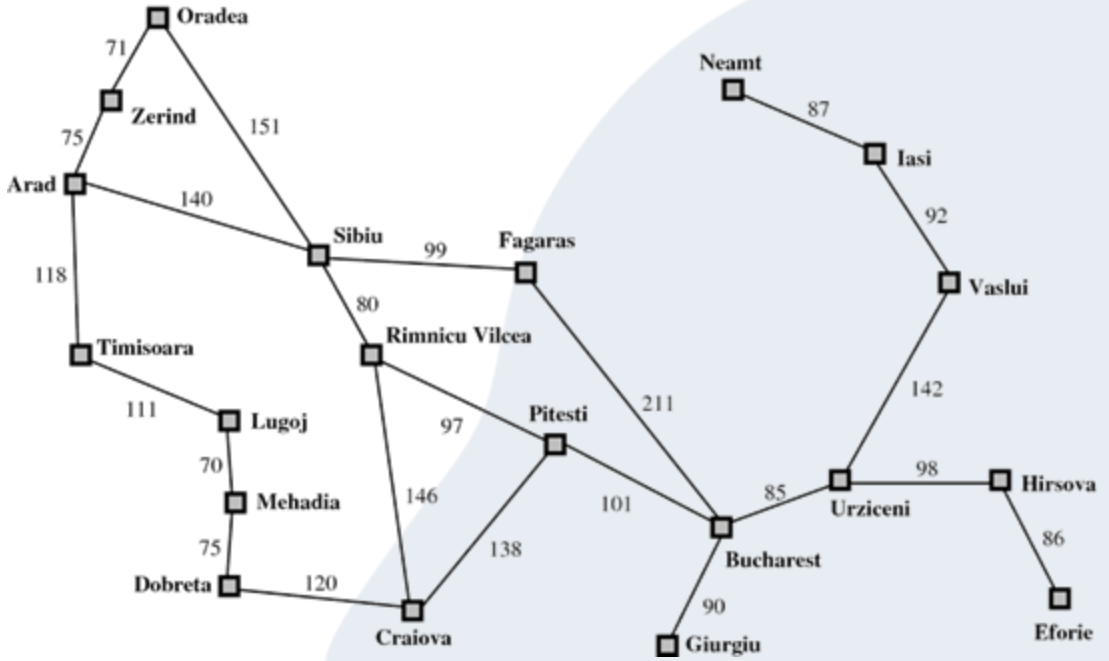
We construct both on demand – and we construct as little as possible.

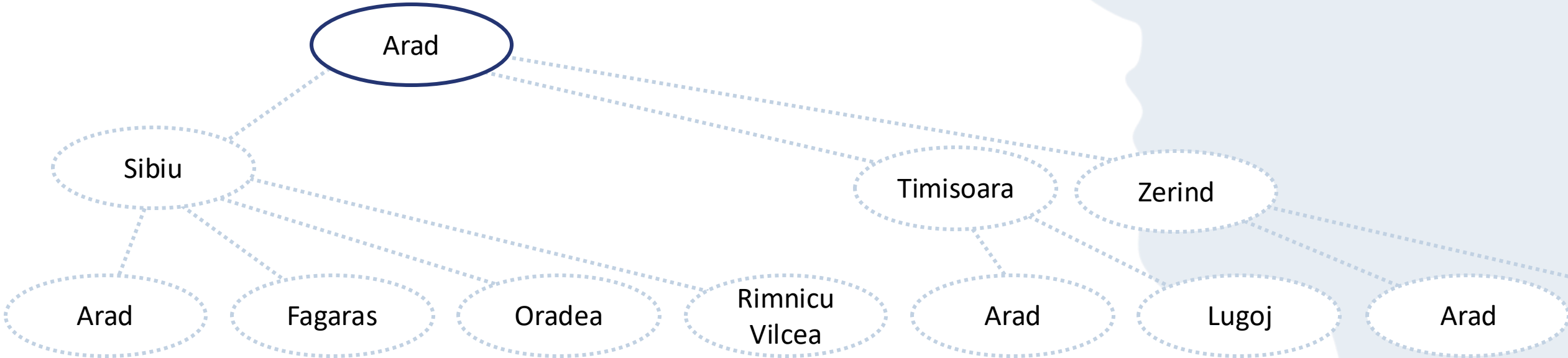



```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the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

Tree Search Algorithms
Example: Romania

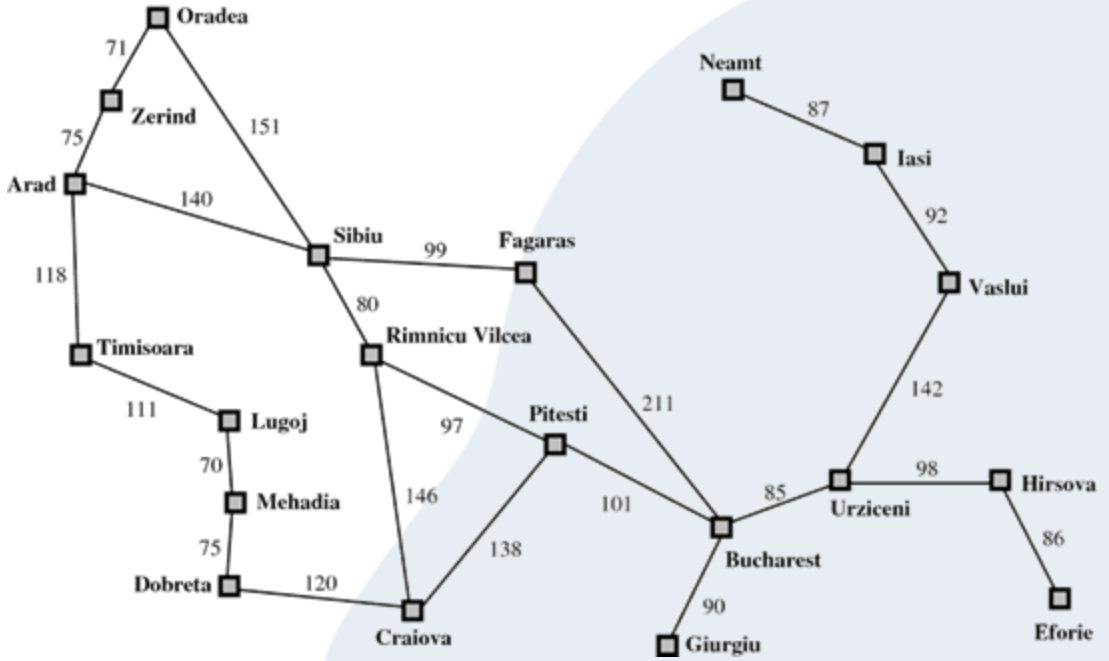
1. Initial State: start with node Arad

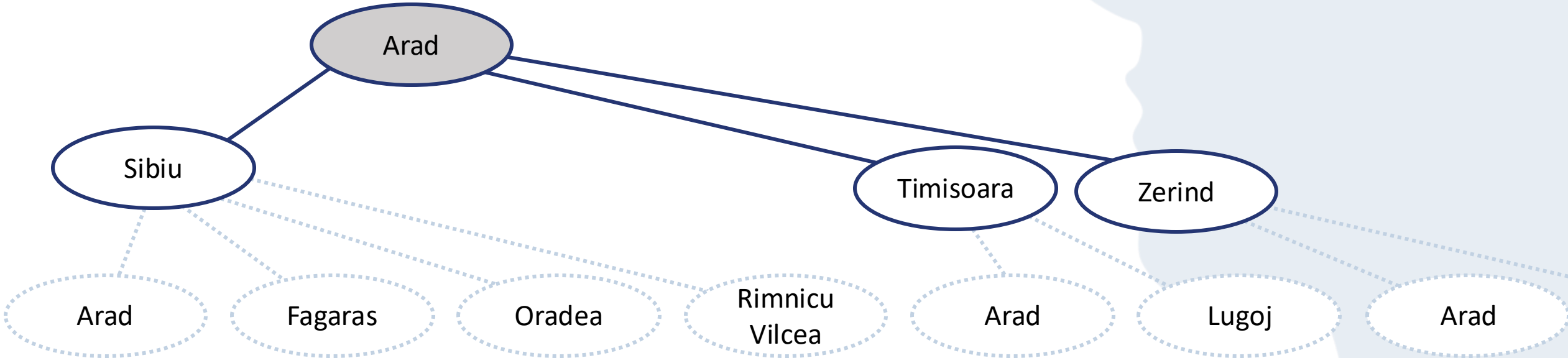




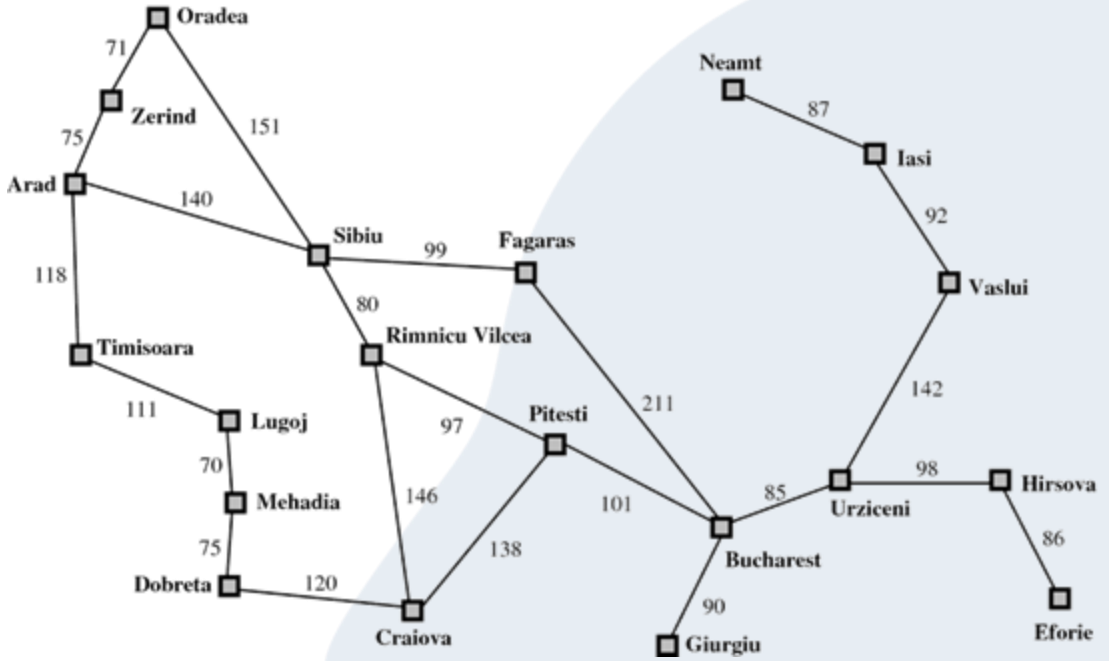
Tree Search Algorithms
Example: Romania

1. Initial State: start with node Arad
2. Expand node Arad

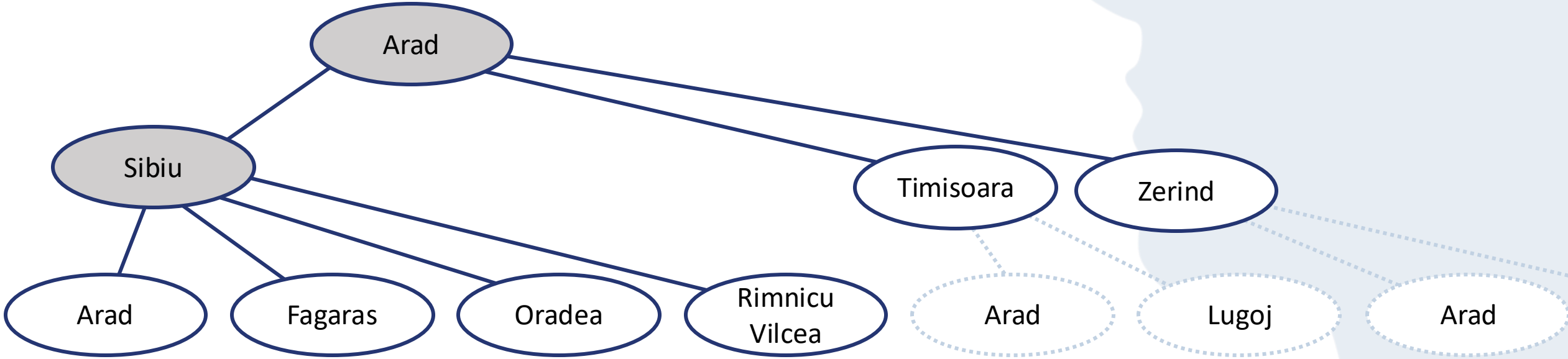




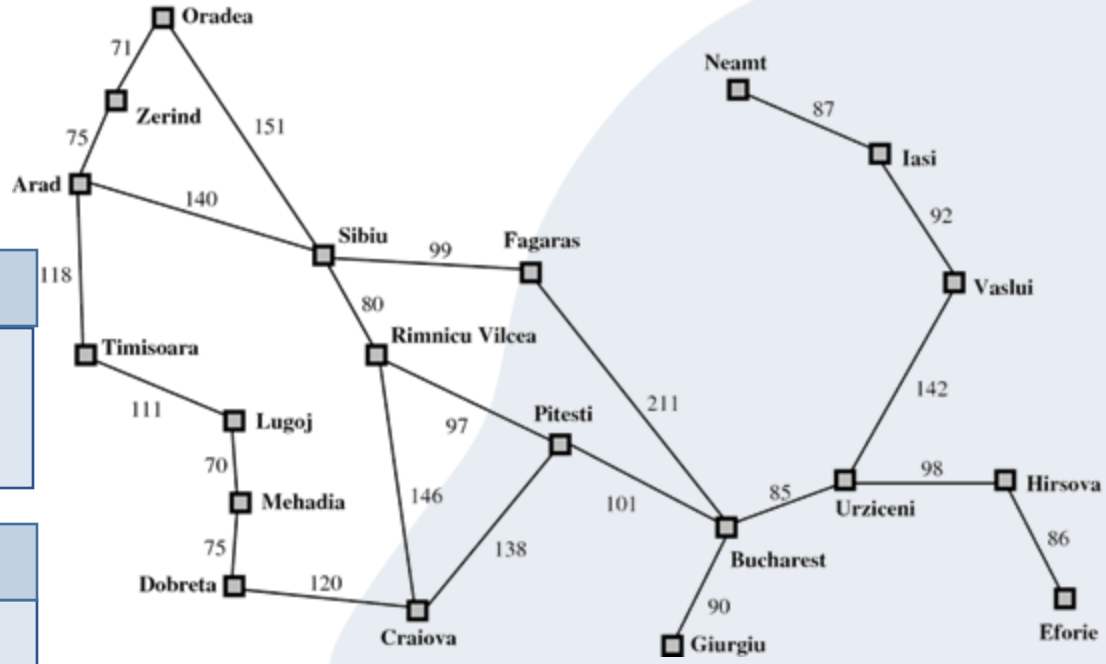
Tree Search Algorithms
Example: Romania



1. Initial State: start with node Arad
2. Expand node Arad
3. Expand node Sibiu



Tree Search Algorithms
Example: Romania

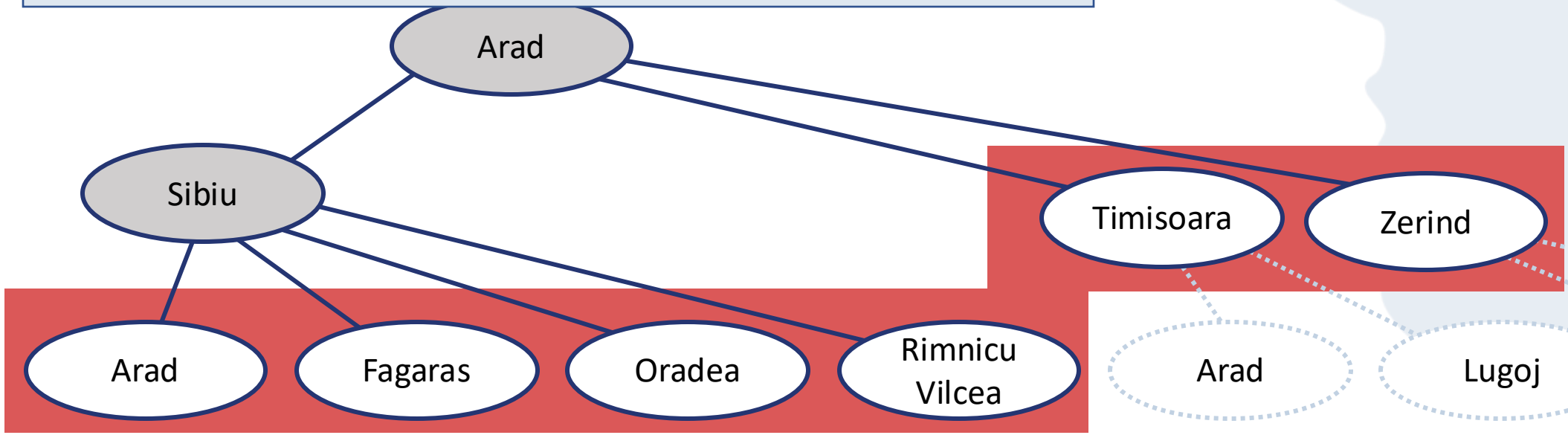


Fringe

The set of all nodes *at the end* of all visited paths is called fringe. (other names are frontier or border)

Depth

Number of levels in the search tree.



depth of the search tree

fringe of the search tree

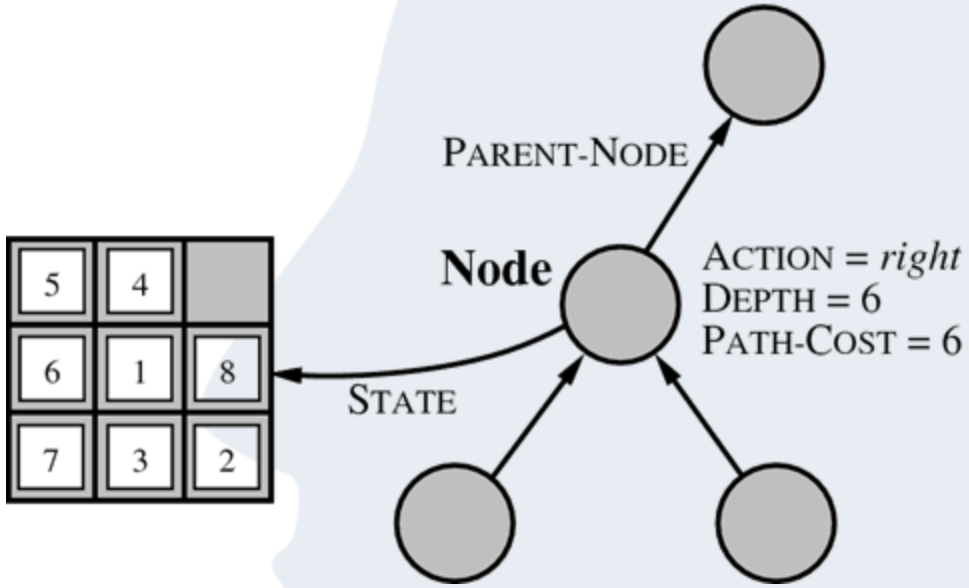
Difference between States and Nodes

State

Representation of a **physical configuration**. Describes a specific situation in our environment.
(see earlier slide)

Node

A **data structure** to present a part of a search tree. It includes a state, a parent node, the taken action, the path costs and the current depth of the tree.



Expanding a node

Expanding a node, creates new nodes (leaf/successor nodes of the chosen node) and updates the tree

Search Strategies

Search Strategy

A search strategy is defined by picking the order of node expansion.

Different search strategies are described/evaluated along the following dimensions:

- **Completeness**: Does it always find a solution if on exists?
- **Time Complexity**: Number of Node expansions
- **Space Complexity**: Maximum number of nodes in memory
- **Optimality**: Does it always find the optimal (least costs) solution?

In our case the time and space complexity is measured in terms of:

- b: the maximum branching factor of the search tree
- d: the depth of the optimal solution
- m: the maximum depth of the state space tree (may be \inf)

Search Strategies

There are two main groups of search strategies

Uninformed Search	Informed Search
Do not have any information except the problem definition.	Have additional knowledge about the problem and an idea where to “look” for solutions.
Breadth-first search (BFS)	Greedy Best-first Search
Uniform-cost search	A* Search
Depth-first search (DFS)	Memory-Bounded Heuristic Search
Depth-limited search	
Iterative deepening	

Uninformed Tree Search Strategies
Uniform-Cost Search and Breadth-First Search (BFS)

Uniform-Cost Search UCS

Each node is associated with a fixed cost (nodes can have different costs) and they accumulate over the path within the search. Uniform-Cost Search uses the lowest cumulative cost to find a path.

Breadth-First Search (BFS)

A special case of the uniform-cost search, when all costs are equal (*). It starts at the tree root and explores the tree level by level. (*) Actually, BFS stops as soon as it generates a goal, whereas UCS examines all the nodes at the goal's depth to see if one has a lower cost.

Completeness : Yes, if each step has a positive cost, otherwise infinite loops are possible.

Hence, BFS is also complete

Complexity: $O(b^d)$ for BFS, $O(b^{1+\text{floor}(\text{OptCost}/\text{eps})})$ for UCS OptCost=Cost of optimal solution, every actions costs at least eps

Optimality: Yes, since nodes expand in increasing order of path costs. In turn BFS is optimal.

BFS and uniform-cost search are often implemented using a priority queue ordered by costs

Uninformed Tree Search Strategies

Computational Costs of BFS: **an exponential complexity bound is scary**

While conceptually simple, Breadth-First Search (BFS)
memory consumption can be too costly!!!!!

Depth	Nodes	Time	Memory
2	110	.11 msecs	107 kB
4	11 110	11 msecs	10.6 MB
6	10^6	1.1 secs	1 GB
8	10^8	2 minutes	103 GB
10	10^{10}	3 hours	10 TB
12	10^{12}	13 days	1 PetaBytes
14	10^{14}	3.5 years	99 PetaBytes
16	10^{16}	350 years	10 ExaBytes

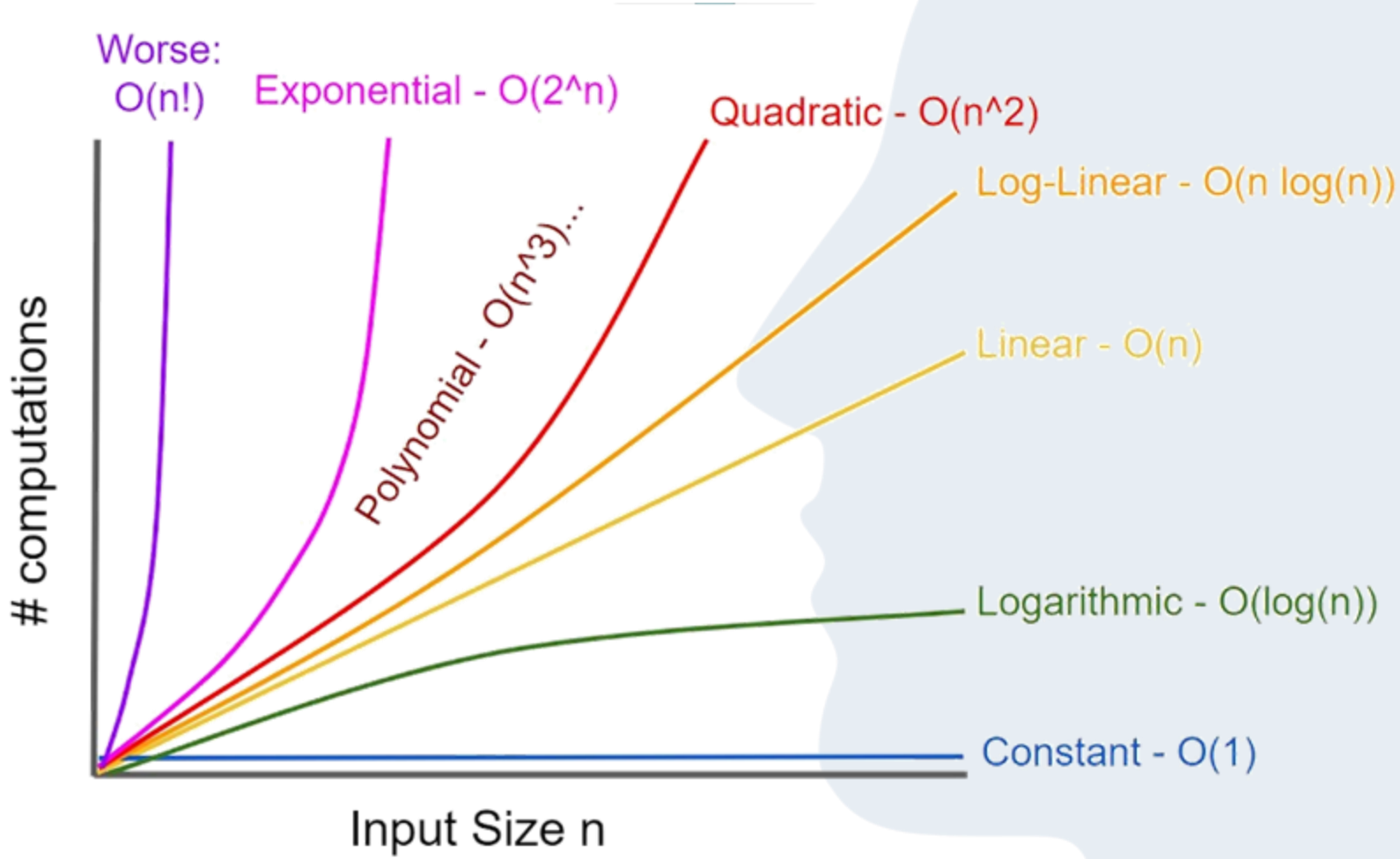
You may wait 13 days,
Put standard computer do not have 1 PB of memry

The table assumes that b=10, that 1 million nodes can be generated per second and that a node requires 1000 bytes of storage.

Uninformed Tree Search Strategies

Computational Costs of BFS: **an exponential complexity bound is scary**

In general, exponential-complexity search problems cannot be solved by uninformed methods for any but the smallest instances.



Uninformed Tree Search Strategies
Depth-First Search (DFS)

Depth-First Search (DFS)

It starts at the tree root and explores the tree as far as possible along one branch before going step-wise back and explore alternative branches.

Completeness: No, fails in infinite-depth search spaces and spaces with loops

- Can be modified to be complete by avoiding repeated states and limit depth

Time Complexity: Explores each branch until max depth *m, i.e.,* $O(b^m)$

- Terrible if *m>d* (depth of goal node), but may be good in dense settings

Space Complexity: Only a branch and their unexpanded siblings has to be stored

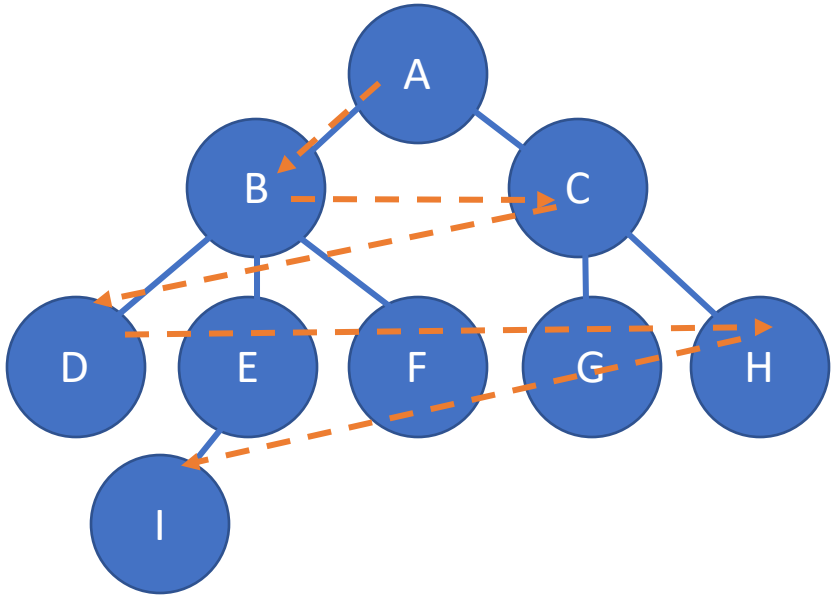
- Therefore linear complexity, i.e., O(b*m)

Optimality: No, longer solutions may be found before shorter solutions

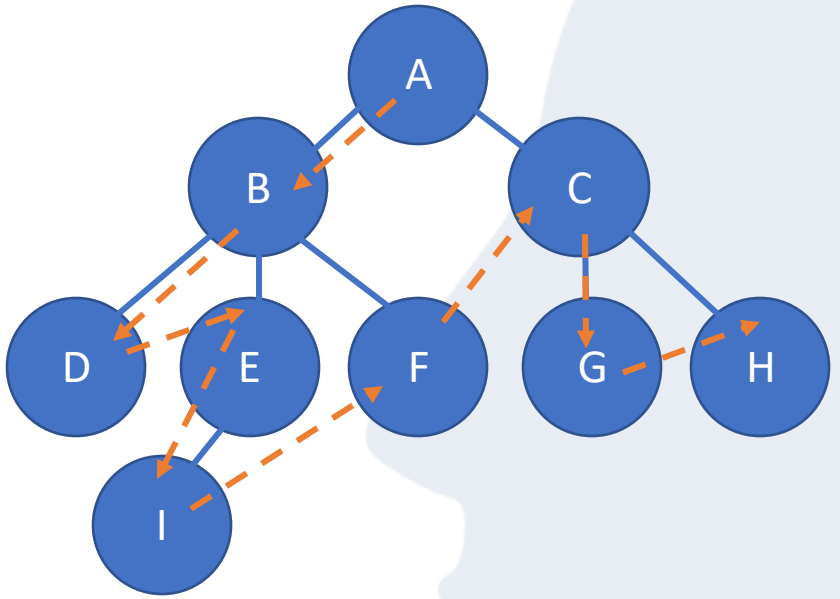
- Solution could be more expensive then the optimal one

Traversing a Tree

traversal order



BFS: A,B,C,D,E,F,G,H,I



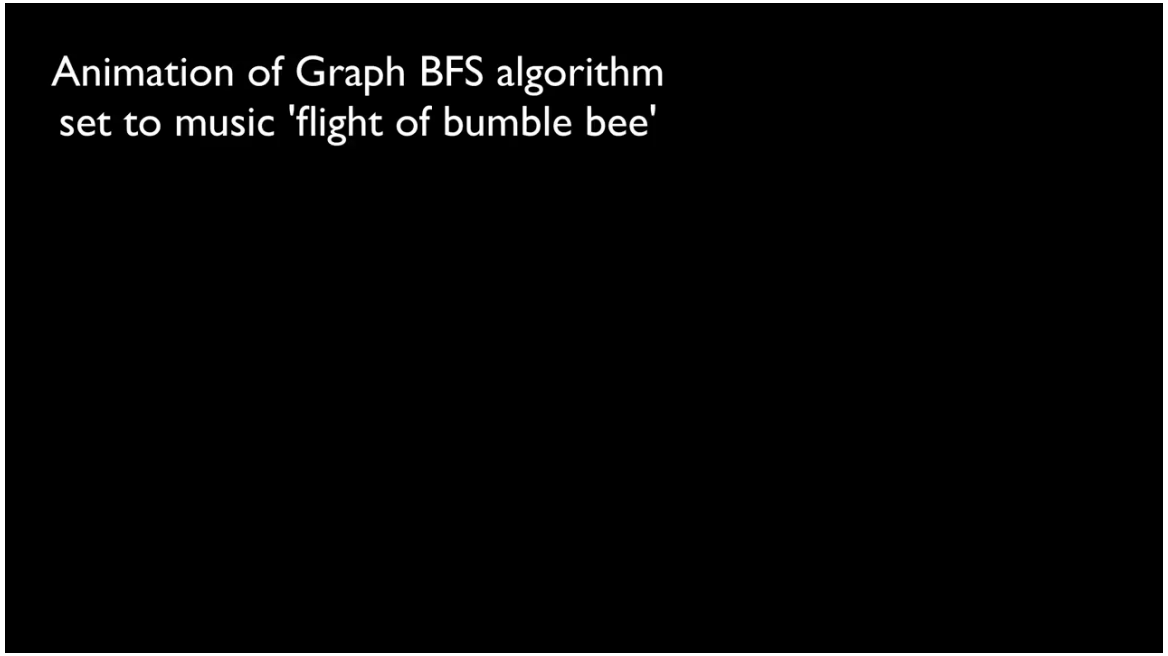
DFS: A,B,D,E,I,F,C,G,H

What would you use?

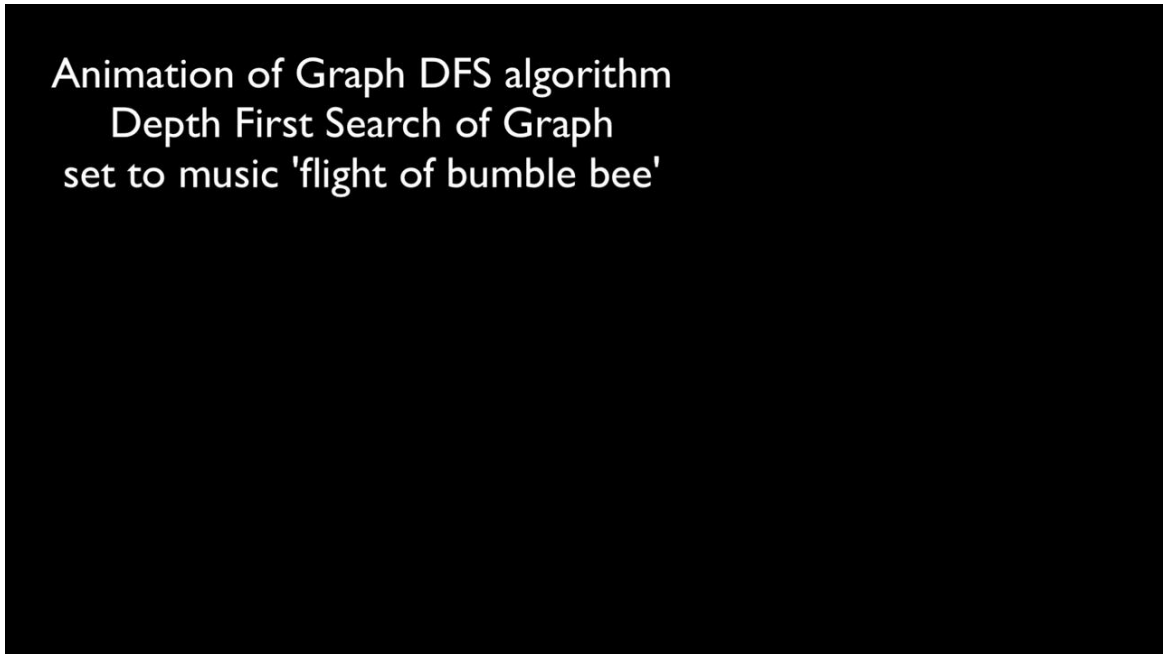
Uninformed Tree Search Strategies
BFS versus DFS

Criteria	BFS	DFS
Concept	Traversing tree level by level	Traversing tree sub-tree by sub-tree
Data Structure (Queue)	First In First Out (FIFO)	Last In First Out (LIFO)
Time Complexity	O(Vertices + Edges)	O(Vertices + Edges)
Backtracking	No	Yes
Memory	Requires more memory	Less nodes are stored normally (less memory)
Optimality	Yes	Not without modification
Speed	In most cases slower compared to DFS	In most cases faster compared to BFS
When to use	If the target is relatively close to the root node	If the goal state is relatively deep in the tree

BFS vs DFS



https://www.youtube.com/watch?v=x-VTfcmrLEQ



https://www.youtube.com/watch?v=NUgMa5coCoE&t=5s

Uninformed Tree Search Strategies
BFS versus DFS

So, can we get the best of both worlds?

Uninformed Tree Search Strategies
Depth-limited Search

Depth-limited Search

The depth within the search is limited to l. Nodes with depth $d>l$ are not considered.

Completeness: No

Time Complexity: $O(b^l)$

Space Complexity: $O(b \times l)$

Optimality: No, see DFS

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST(problem, STATE[node]) then return node
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff-occurred? ← true
      else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```

How do we select l?

Uninformed Tree Search Strategies
Iterative Deepening Search

Iterative Deepening Search

Increase l after each failed search, i.e. l = 1, 2, 3, …

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

- **Completeness**: Yes
- **Time Complexity**: first levels has to be search *d* times

$$\Rightarrow d \cdot b + (d-1)b^2 + \ldots + 1 \cdot b^d = \sum_{i=1}^{d} (d-i+1) \cdot b^i$$

- **Space Complexity:** linear complexity

$$O(b \cdot d)$$

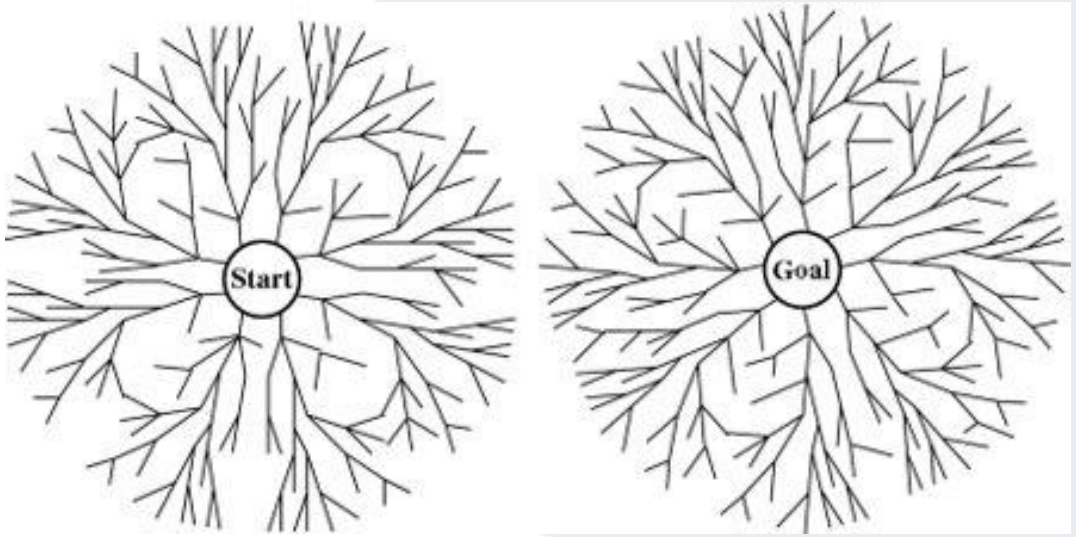
- **Optimality**: Yes, the shortest path is found

Uninformed Tree Search Strategies
Bidirectional Search

Bidirectional Search

Perform two search simultaneously, starting with the root and goal state. Stop if node occurs in both searches.

- Reduction in complexity $(b^{d/2} + b^{d/2} << b^d)$
- Only possible if actions can be reversed
- Search paths may not meet for depth-first bidirectional search



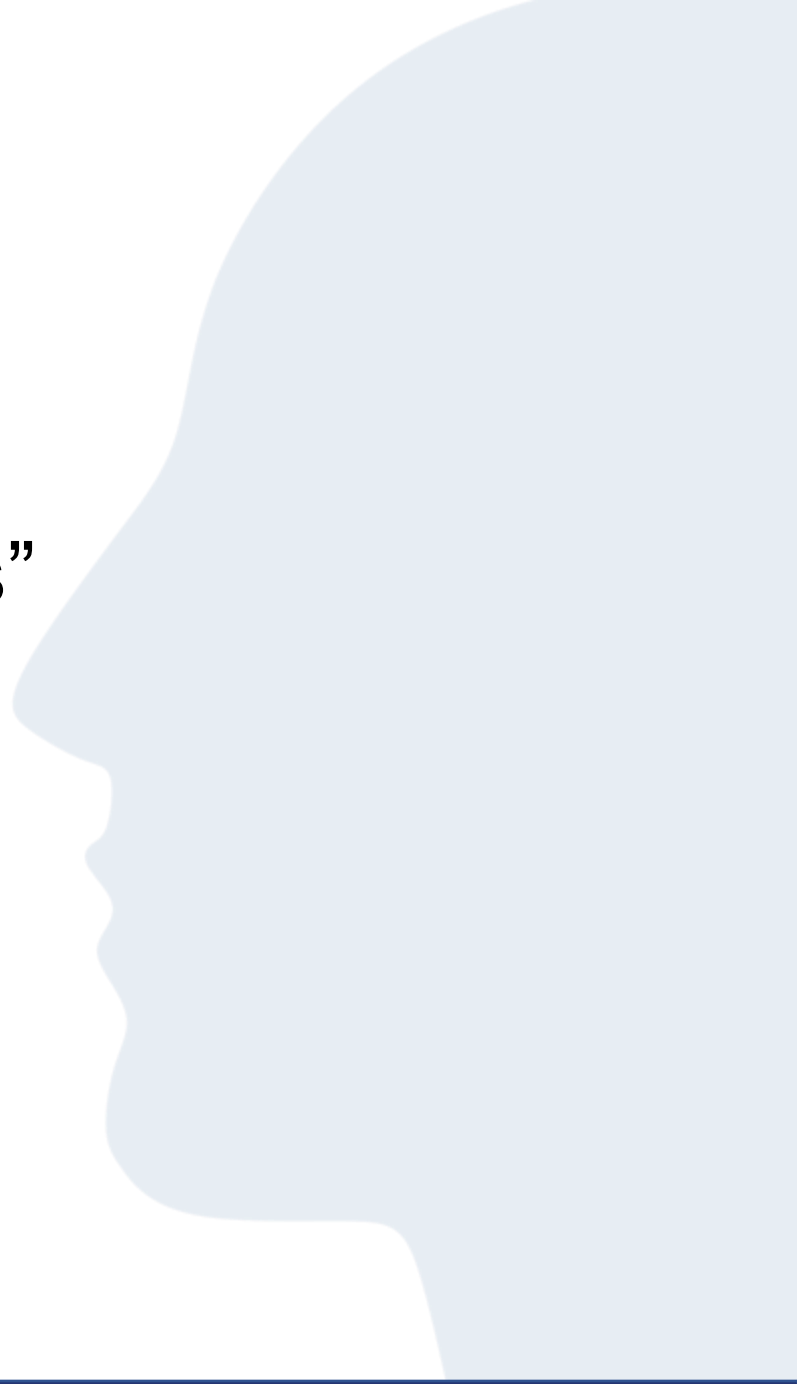
Uninformed Tree Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

Informed Tree Search Strategies

Can we improve performance?

What about giving the search algorithm “hints” about the desirability of different states?



Informed Tree Search Strategies

Uninformed search algorithms are inefficient.

Idea: try to be more clever with what nodes to expand, bring knowledge into the process

- i.e. straight-line distances may be a good approximation for the remaining distance to travel

Heuristics h

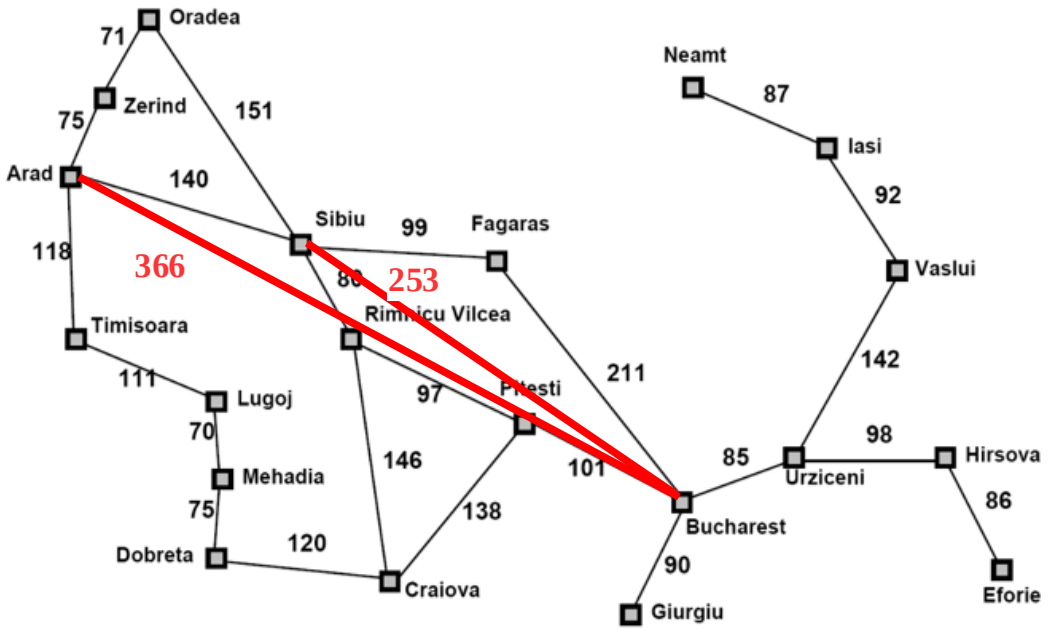
Informally denotes a „rule of thumb“, i.e. a rule that may be helpful in solving the problem.
In tree-search, a heuristic denotes a function h that estimates the remaining costs to reach the goal.

Note:

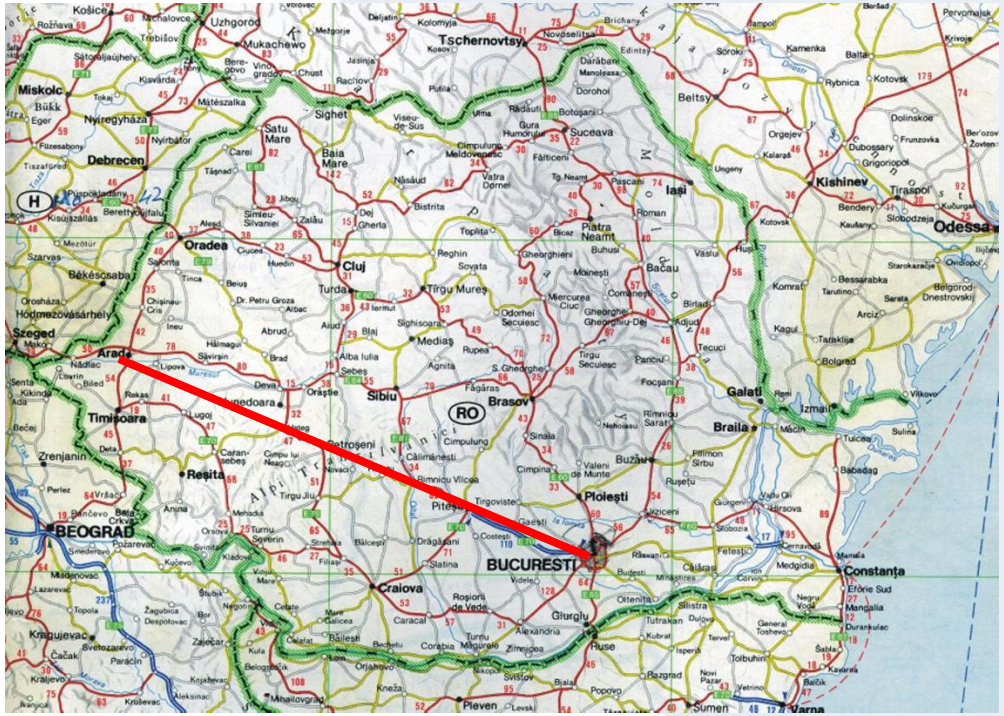
Heuristic can also go wrong!

Informed Tree Search Strategies
Greedy Best-first Search

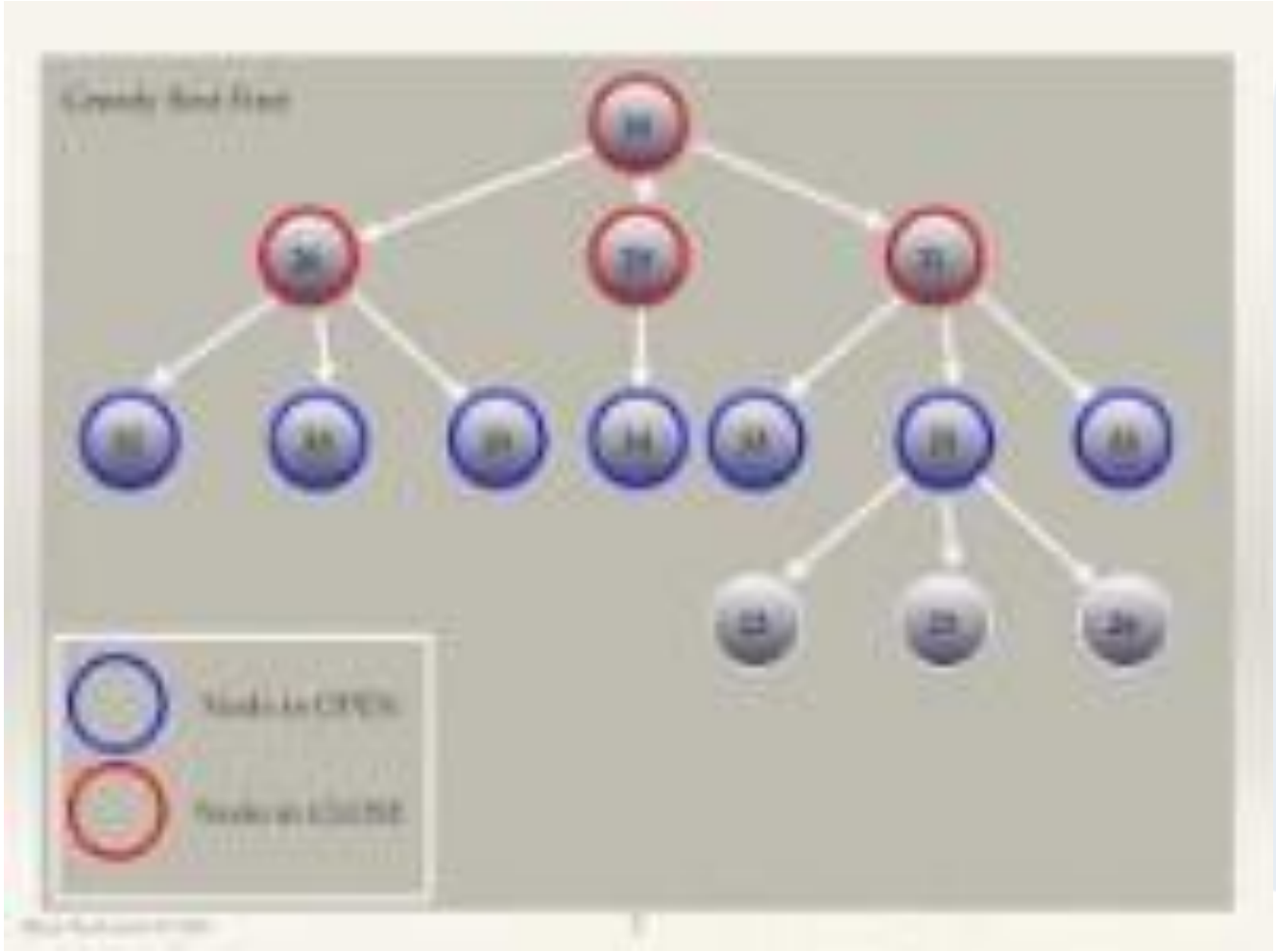
- Using the evaluation function $f(n) = h(n)$ to estimate the cost from node n to goal
- e.g. $h(n)=hSLD(n)$ = straight-line distance from n to Bucharest
- Expand the tree with smallest cost according to $f(n)$, i.e., here the heuristic



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Informed Tree Search Strategies
Greedy Best-first Search



Shaul Markovitch, https://www.youtube.com/watch?v=A8pmud1Uh0Q&t=1s

Informed Tree Search Strategies
Greedy Best-first Search

Completeness: No, we can get stuck in loops

- Is complete in finite state space when we make sure to avoid repeating states

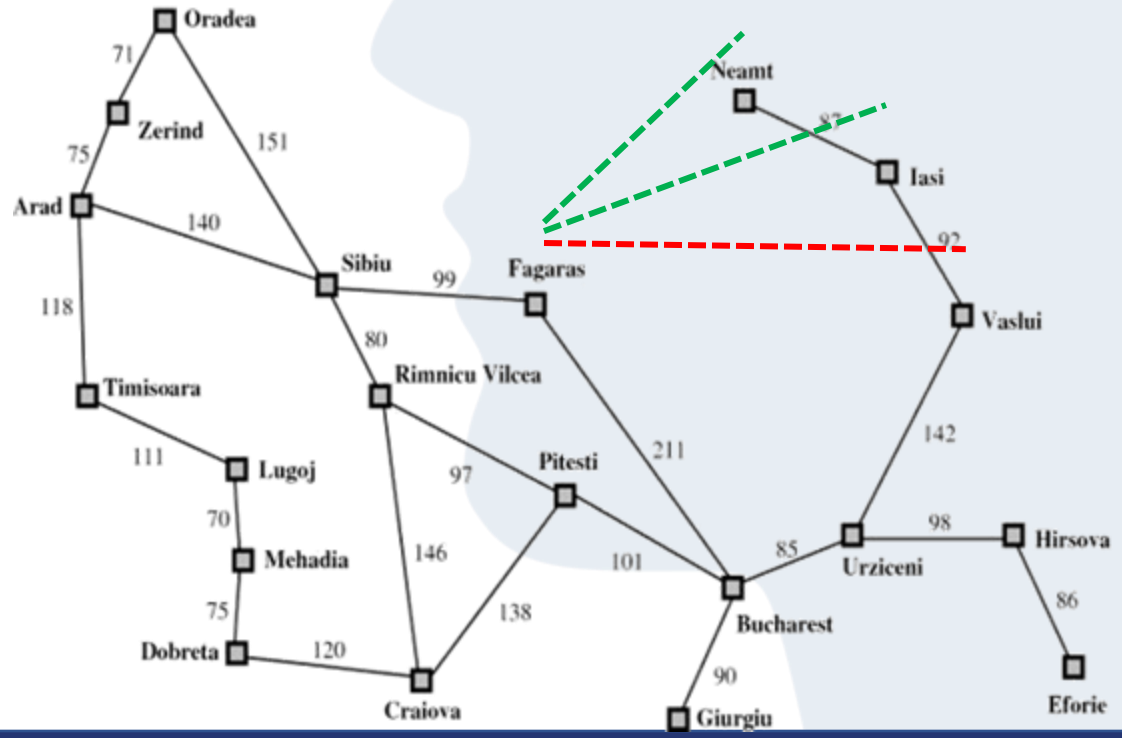
Time Complexity: Worst case $O(b^m)$, same as DFS but can be improved using good heuristics

Space Complexity: has to keep all nodes in memory, worst case $O(b^m)$

Optimality: No, solution depends on heuristic.

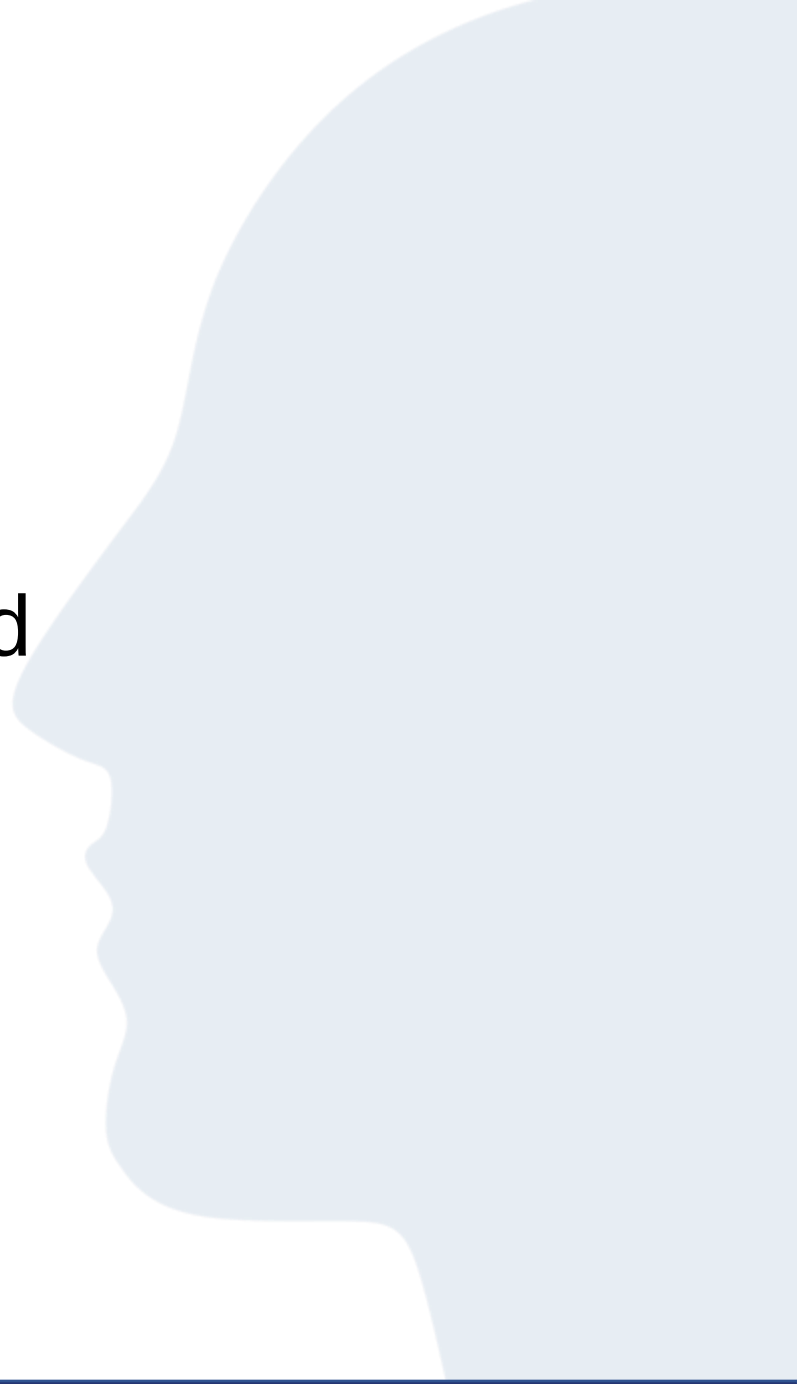
Example, where our heuristic is not helping:

- We want to get from Iasi to Fagaras
- Problem: Neamt is “closer” to Fagaras than Vaslui



Informed Tree Search Strategies

Hmm, is there a chance make it complete and terminating?



Informed Tree Search Strategies
A* Search

A* Search

An informed tree search algorithm, build on best-first search.
Tries to minimize not only the estimated cost *h(n)* but also the true costs so far *g(n)*.

Best-known form of Best-First Search

Idea: Avoid expanding paths that are already expensive

- evaluate complete path cost not only remaining costs
- i.e. $f(n) = g(n) + h(n)$

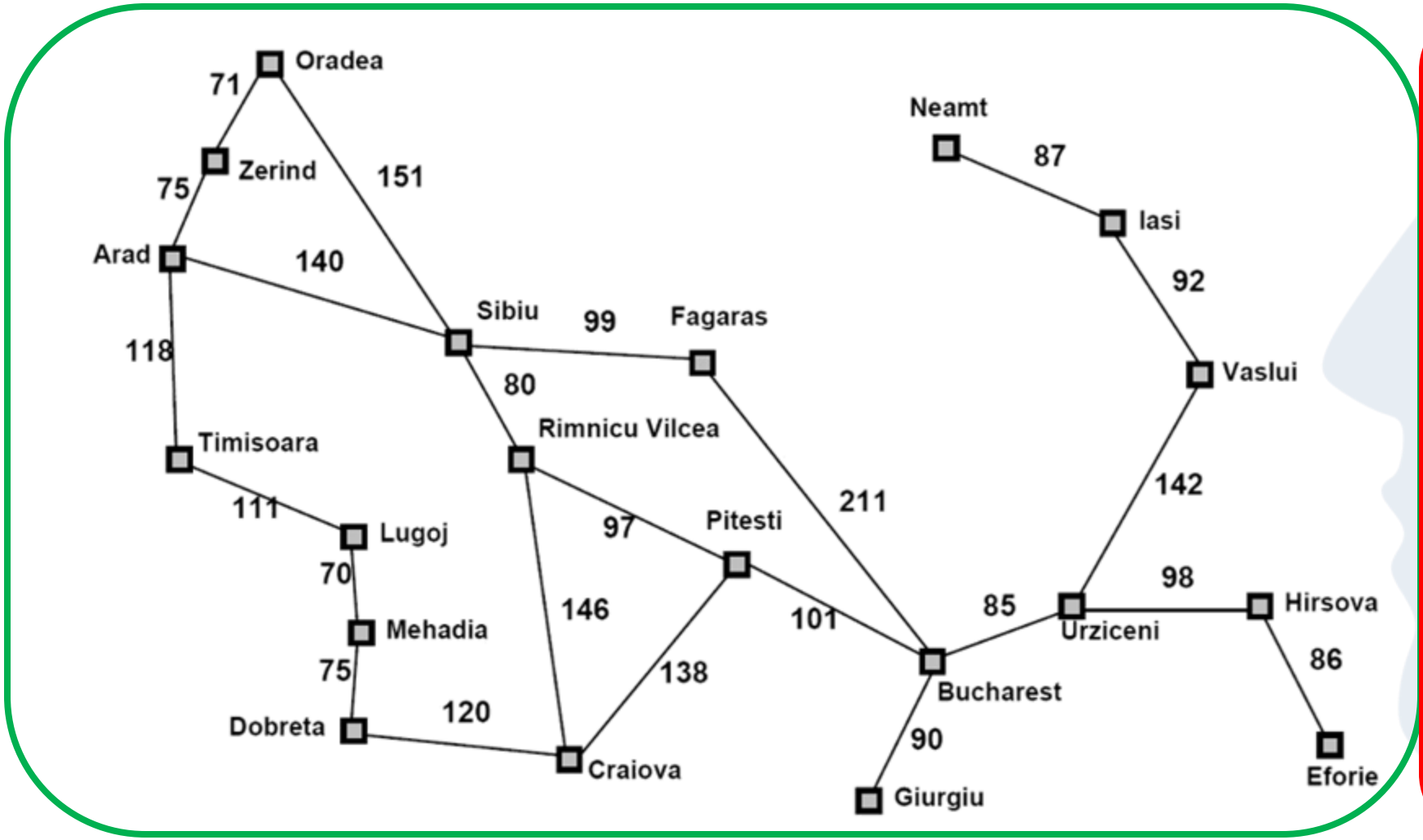
Evaluation Function:

- $g(n)$ = cost so far to reach node n
- $h(n)$ = estimated cost to get from n to goal
- $f(n)$ = estimated cost of path to goal via n

Informed Tree Search Strategies

A* Search

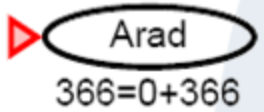
$g(n)$

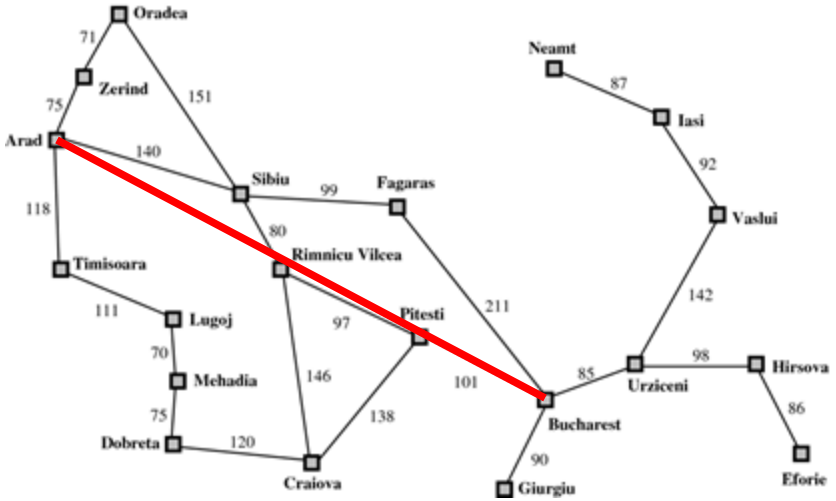


$h(n)$

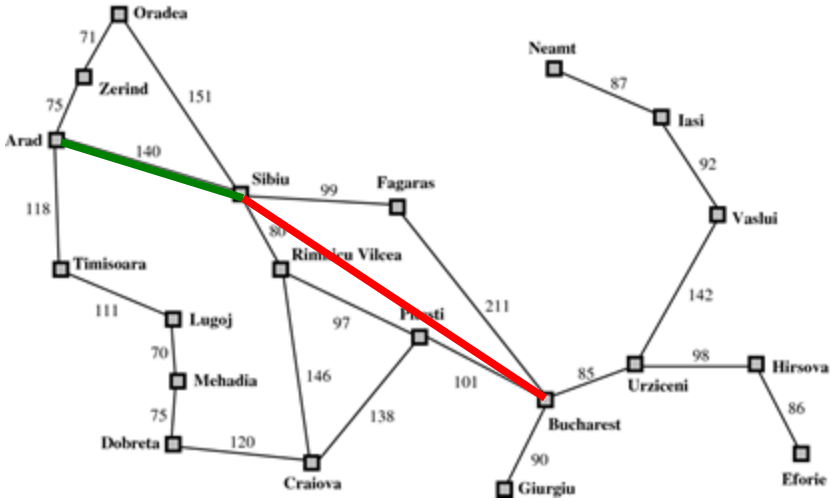
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Informed Tree Search Strategies
A* Search: Example

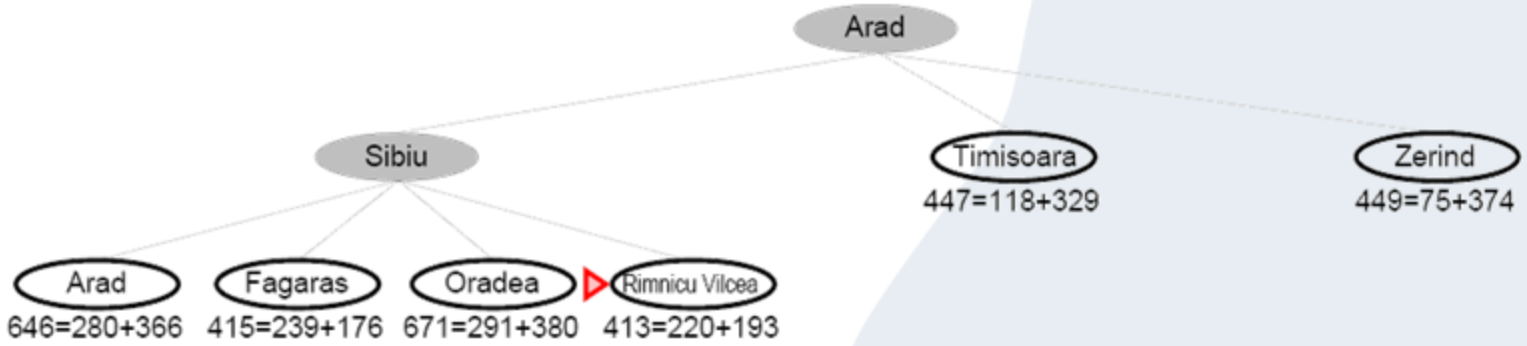


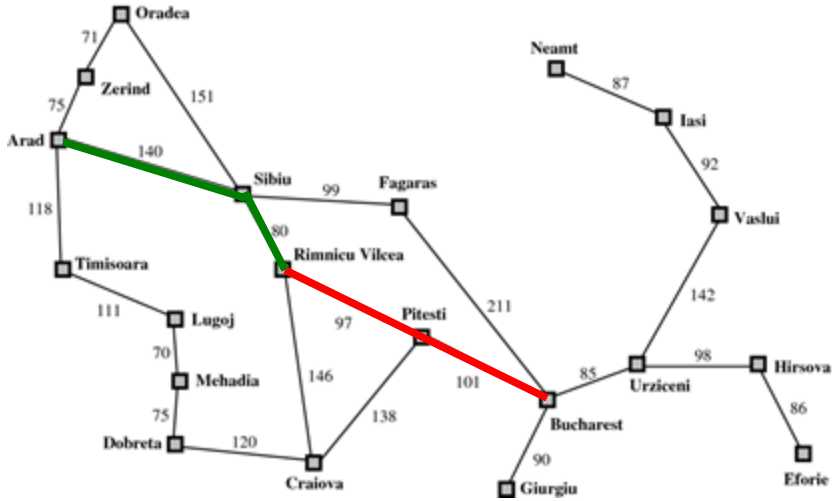


Informed Tree Search Strategies
A* Search: Example

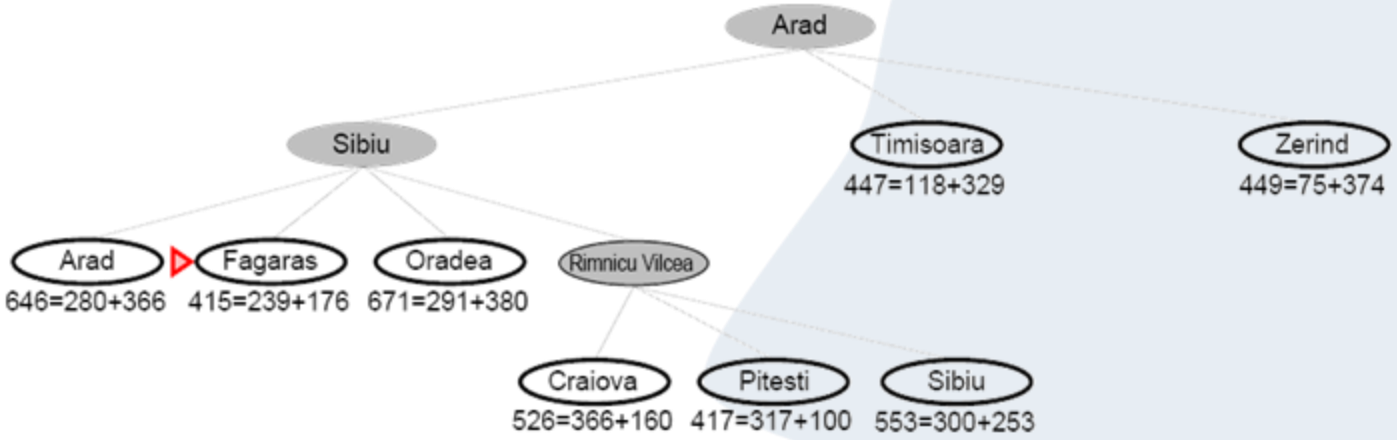


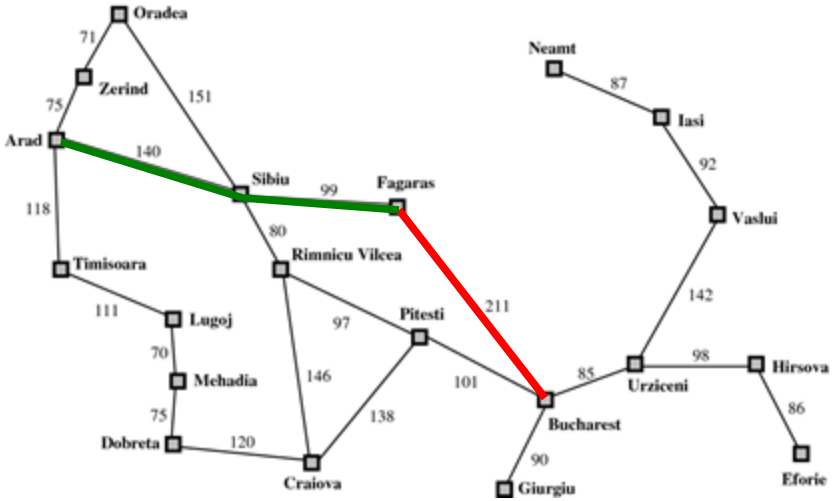
Informed Tree Search Strategies
A* Search: Example





Informed Tree Search Strategies
A* Search: Example

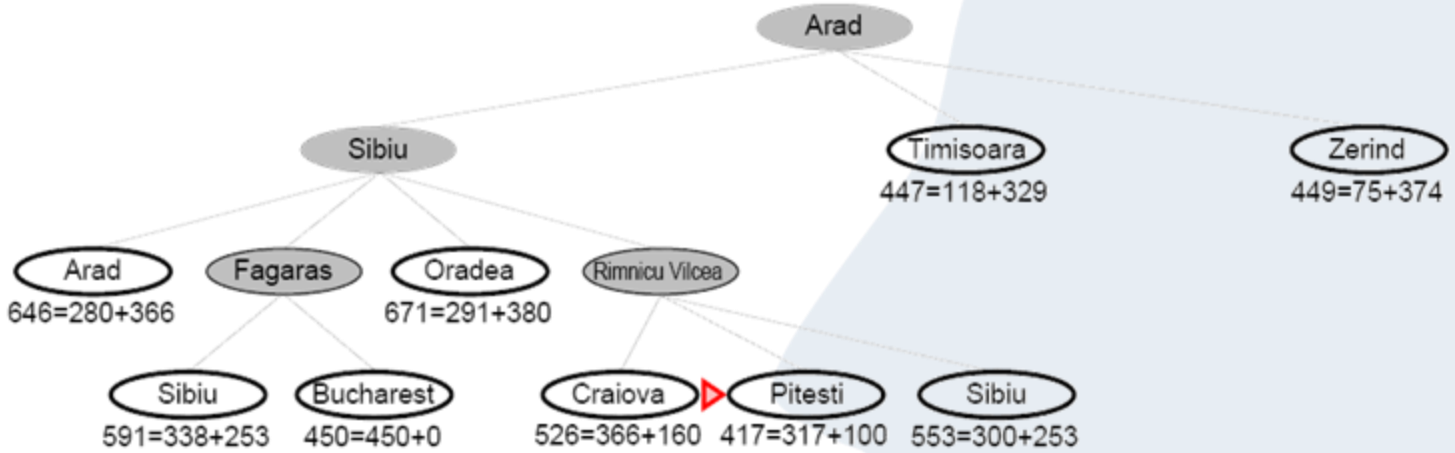


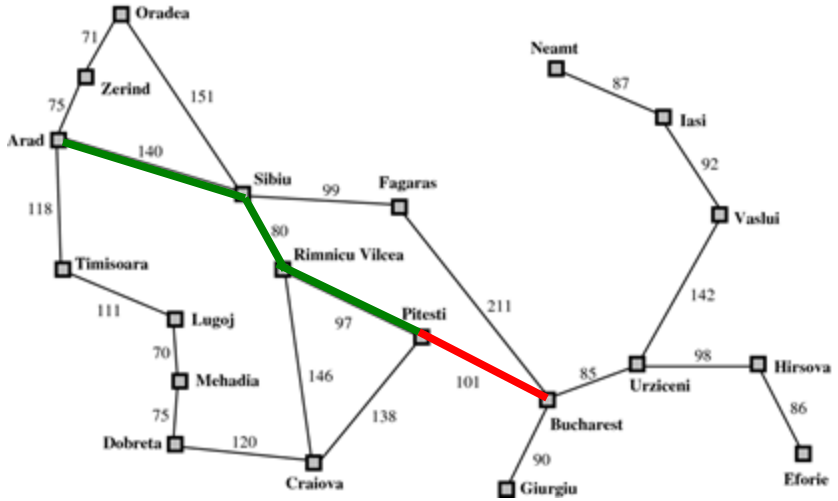


Informed Tree Search Strategies
A* Search: Example

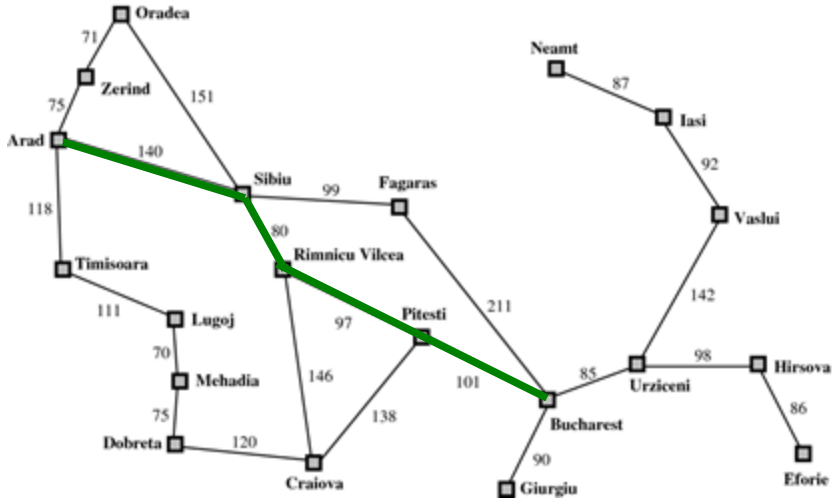
Note that Pitesti will be expanded even though Bucharest is already in the fringe! This is good, because we may still find a shorter way to Budapest. We want to have certificate of optimality!

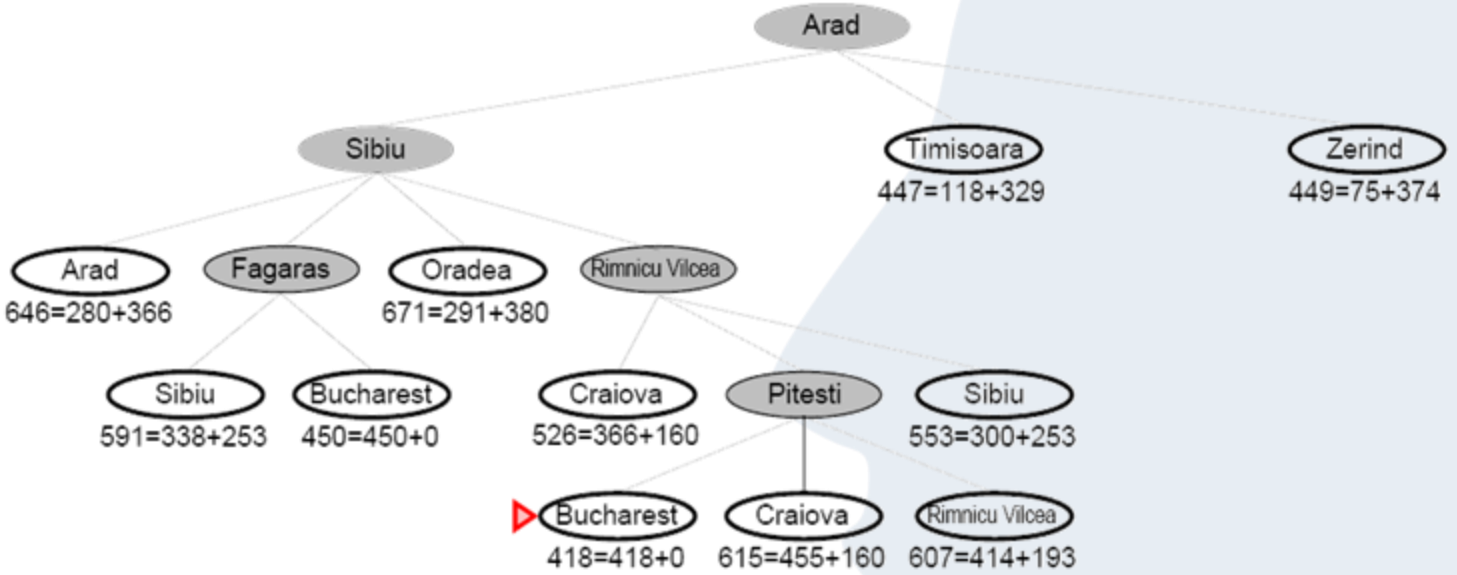
Greedy Search would not do that.



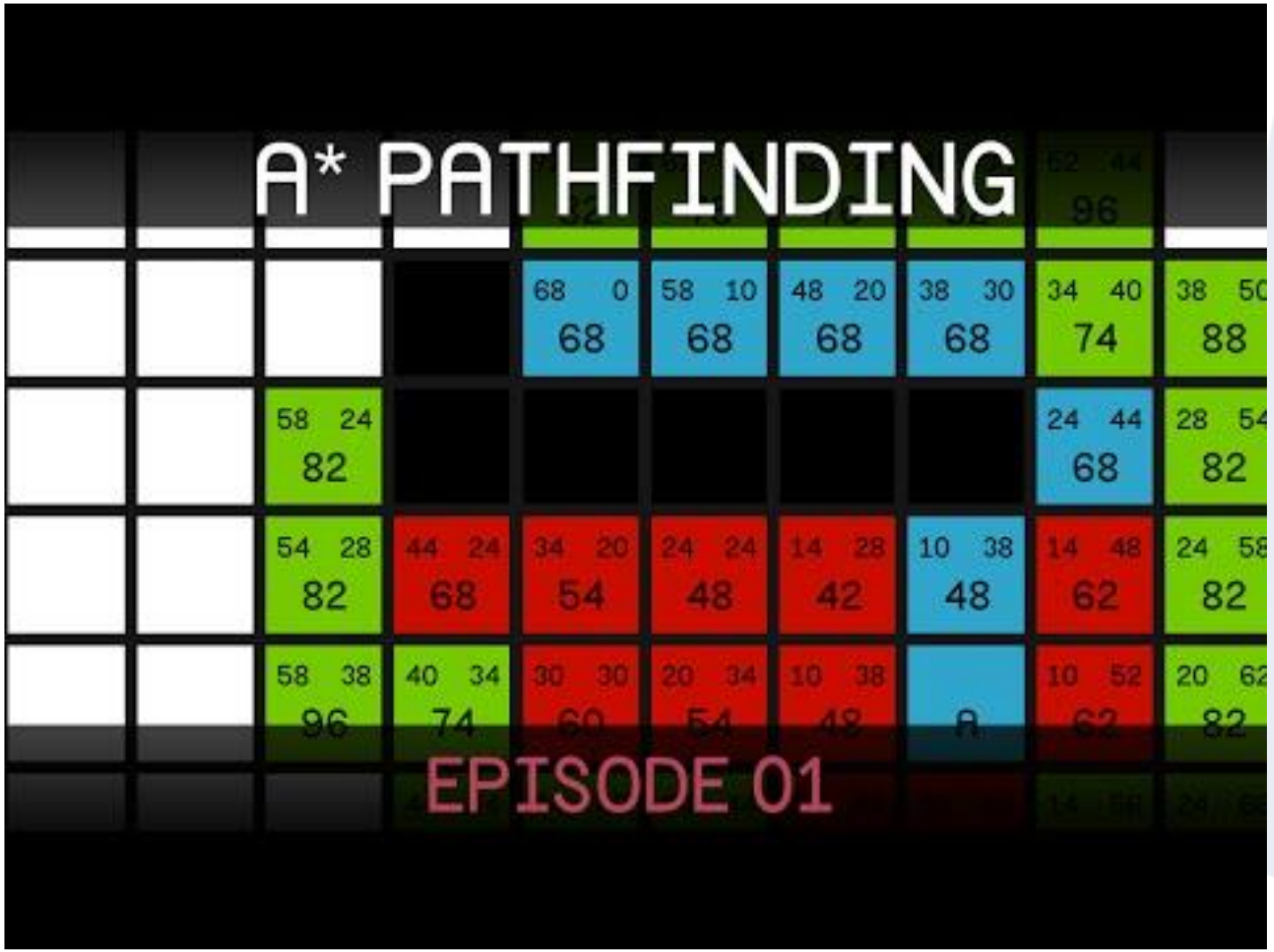


Informed Tree Search Strategies
A* Search: Example





Informed Tree Search Strategies
A* Search: Example



Sebastian Lague, https://www.youtube.com/watch?v=-L-WgKMFuhE

Informed Tree Search Strategies
A* Search

Completeness: Yes

Exception: If there are infinitely many nodes with $f(n) \leq f(G)$

Time Complexity:

It can be shown that the number of nodes grows exponentially unless the error of the heuristic $h(n)$ is bounded by the logarithm of the value of the actual path cost $h^*(n)$, i.e.

$$| h(n) - h^*(n) | \leq O\left(\log h^*(n)\right)$$

Space Complexity: has to keep all nodes in memory

Optimality: Depends on the heuristic, see next pages

Optimality of Heuristics

Example: Dominant Strategies on the Example of 8-Puzzles

Comparison of number of nodes searched by A* and Iterative Deepening Search (IDS)

- average of 100 different 8-puzzles with different solutions
- We see the efficency of h2(n) [Manhatten] over h1(n) [Missing Tiles]

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Informed Tree Search Strategies

Hmm, is there a chance to make it optimal?

The Kingdom of Heuristics
Admissible Heuristics

Heuristics (Reminder)

Informally denotes a „rule of thumb“, i.e. a rule that may be helpful in solving the problem.
In tree-search, a heuristic denotes a function h that estimates the remaining costs to reach the goal.

Admissible Heuristics

A heuristic is admissible if it never overestimates the cost to reach a goal

Formally: $h(n) \leq h^*(n)$ if $h^*(n)$ are the true cost from n to goal

Example

- Straight-line distances ($hSLD(n)$) are admissible for the actual distances

The Kingdom of Heuristics
Consistent Heuristics

Consistent Heuristics (kind of triangle inequality)

A heuristic is consistent if for every node n and every successor n' generated by any action a it holds that $h(n) \leq c(n,a,n') + h(n')$. Thus, a heuristic is consistent if, when going from neighboring nodes a to b, the heuristic difference/step cost never overestimates the actual step cost.

This makes sense:

if there were a route from n to the goal that was cheaper than h(n), that would violate the property that h(n) is a lower bound on the cost to reach the goal (kind of triangle inequality

Lemma 1:

Every consistent heuristic is admissible.

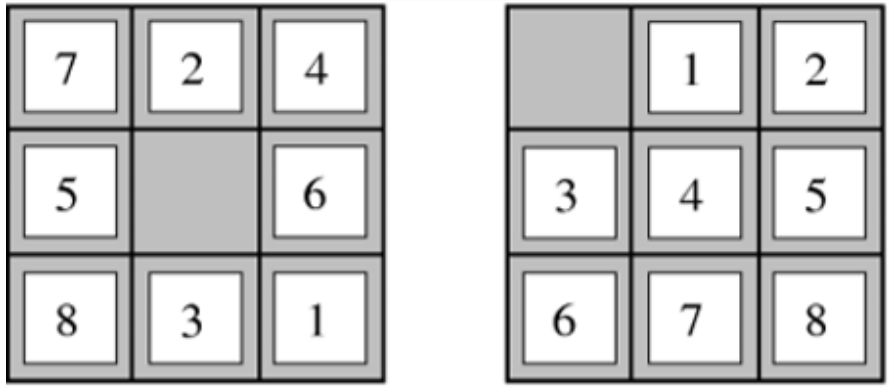
Lemma 2:

If $h(n)$ is consistent, then the values of $f(n)$ along any path are non-decreasing.

The Kingdom of Heuristics
Designing Heuristics: Heuristics for the 8-puzzle

Two Ideas

h1(n) = Number of misplaced tiles
h2(n) = Total Manhattan distance (number of squares from desired location of each tile)



h1(start) = 8

h2(start) = 3+1+2+2+2+3+3+2 = 18

Are h1 and h2 admissible?

The Kingdom of Heuristics
Heuristics for Relaxed Problems

Relaxed Problems

A problem with fewer restrictions on the actions is called a relaxed problem

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

Examples:

- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h\text{MIS}$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h\text{MAN}$ gives the shortest solution

Thus, looking for relaxed problems is a good strategy for inventing admissible heuristics.

The Kingdom of Heuristics
Dominance

Dominance

If h1 and h2 are both **admissible** heuristics and h2(n) ≥ h1(n) for all n, then h2 dominates h1

If h2 dominates h1 it will perform better as it will always be closer to the optimal heuristic h^*

Example:

h2 (Manhatten) dominates h1 (misplaced tiles) because if a tile is misplaced, its Manhattan distance is ≥ 1

Why is this important

Search algorithms expand every node with f(n) < C* vs. h(n) < C* – g(n)

Thus, dominant heuristics result in less expansion

The Kingdom of Heuristics
Combining Heuristics

Suppose we have a collection of admissible heuristics h1(n), h2(n), ..., hm(n), but none of them dominates the others

How to combine these?

Theorem (Combining admissible heuristics):

If *h*1 and *h*2 are two admissible heuristics than

$$h(n) = max\{h1(n), h2(n), \ldots, hm(n)\}$$

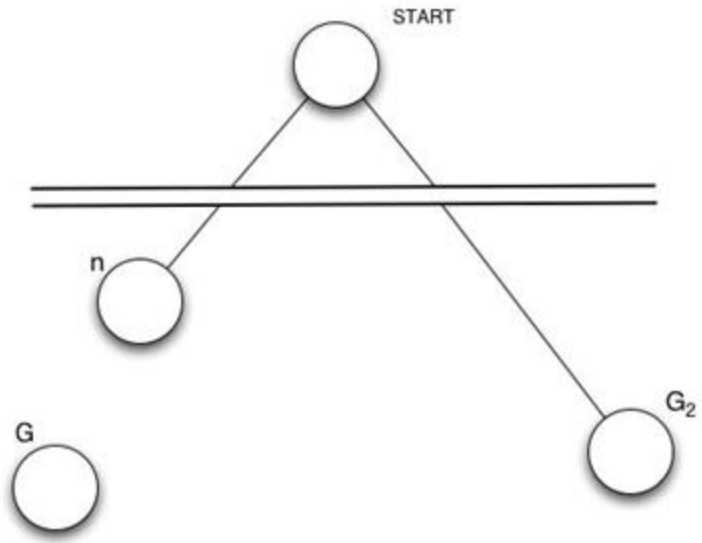
is also admissible and dominates *h*1 and *h*2

Optimality of Heuristics

Is A* optimal?

If $h(n)$ is admissible, A* using TREE-SEARCH is optimal.

Proof:



Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G with path cost $C^* = g(G)$.

$C^* = g(n) + h^*(n)$

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

$h(n) \leq h^*(n)$

because h admissible

$f(n) \leq C^* < f(G_2)$

G_2 will never be expanded, and G will be returned

Suppose some suboptimal goal G_2 has been generated and is in the fringe.

$g(G_2) > C^*$

because G_2 suboptimal

$f(G_2) = g(G_2)$

because $h(G_2) = 0$

(holds for all goal nodes)

Optimality of Heuristics
Is A* optimal?

Assumption: If A* selects a path p, p is the shortest (i.e., lowest-cost) path.
Proof:

- Assume for contradiction that some other path p′ is actually the shortest path to a goal
- Consider the moment just before p is chosen from the fringe. Some part of path p′ will also be on the fringe; let's call this partial path p″.
- Because p was expanded before p″, $f(p) \leq f(p'')$.
- Because p is a goal, $h(p) = 0$. Thus $cost(p) \leq cost(p'') + h(p'')$.
- Because h is admissible, $cost(p'') + h(p'') \leq cost(p')$ for any path p′ to a goal that extends p″
- Thus $cost(p) \leq cost(p')$ for any other path p′ to a goal. This contradicts our assumption that p′ is the shortest path.

Informed Tree Search Strategies

Three alternatives to A*: Memory-Bounded Heuristic Search

Problem: A* (and Best-First search) has a space problem

1. **Iterative-deepening A* (IDA*)**
   - like iterative deepening
   - cutoff information is the *f*-cost ($g + h$) instead of depth
2. **Recursive best-first search (RBFS)**
   - recursive algorithm that attempts to mimic standard best-first search with linear space.
   - keeps track of the *f*-value of the best alternative
   - path available from any ancestor of current node and heuristic evaluations are updated with results of successors
3. **(Simple) Memory-bounded A* ((S)MA*)**
   - drop the worst leaf node when memory is full
   - its value will be updated to its parent
   - may need to be re-searched later

Informed Tree Search Strategies
Graph Search

Failure to detect repeated states can turn a linear problem into an expoential one!

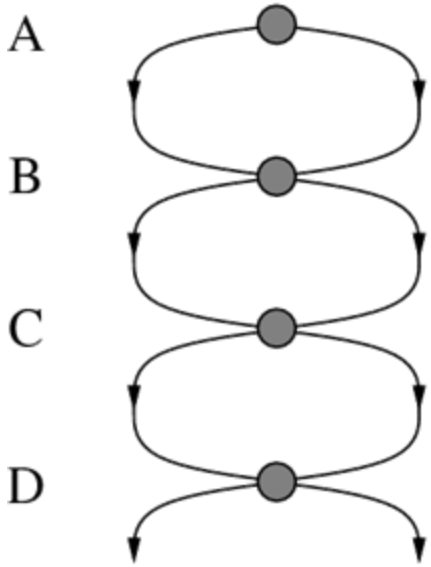
Ribbon and Grid Examples

- two resp. four connections from each state to another one

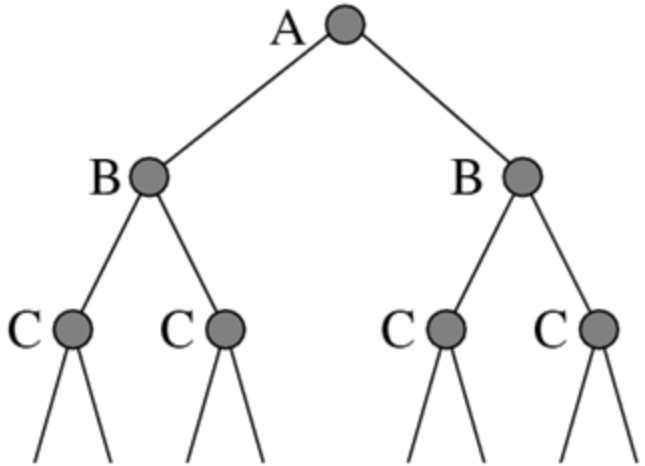
d states

but state space is 2^d

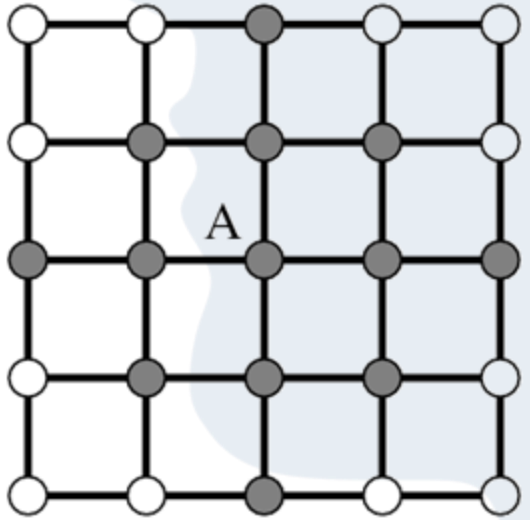
or e.g. 4^d though only about $2d^2$ different states ar reachable in *d* steps



(a)



(b)



(c)

Informed Tree Search Strategies
Graph Search

Remembers the states that have been visited in a list

Example

Dijkstra's algorithm is the graph-search variant of uniform cost search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Informed Tree Search Strategies

Tree Search vs. Graph Search

Tree search

- each node in the search tree represents one possible path to the associated domain state
- e.g., one can go directly from Arad to Sibiu or via Oradea

Problem

- In some cases the path that is detected later may be the better path
- Previous found paths have to be re-investigated with the new, cheaper path

Two solutions

1. Add ability to detect repeated states
   - → graph search
   - general solution that also works for BFS, DFS, ...
2. Or ensure that the cheaper path is always taken first
   - → consistent heuristics
   - specific solution for A* (next slides)

International Conference on Automated Planning and Scheduling (ICAPS) 2021

Improving AlphaZero Using Monte-Carlo Graph Search

Johannes Czech[1], **Patrick Korus**[1], **Kristian Kersting**[1, 2, 3]
[1] Department of Computer Science, TU Darmstadt, Germany
[2] Centre for Cognitive Science, TU Darmstadt, Germany
[3] Hessian Center for Artificial Intelligence (hessian.AI), Darmstadt, Germany
johannes.czech@cs.tu-darmstadt.de, patrick.korus@stud.tu-darmstadt.de, kersting@cs.tu-darmstadt.de

Abstract

The *AlphaZero* algorithm has been successfully applied in a range of discrete domains, most notably board games. It utilizes a neural network that learns a value and policy function to guide the exploration in a Monte-Carlo Tree Search. Although many search improvements such as graph search have been proposed for Monte-Carlo Tree Search in the past, most of them refer to an older variant of the Upper Confidence bounds for Trees algorithm that does not use a policy for planning. We improve the search algorithm for *AlphaZero* by generalizing the search tree to a directed acyclic graph. This enables information flow across different subtrees and greatly reduces memory consumption. Along with Monte-Carlo Graph Search, we propose a number of further extensions, such as the inclusion of ε-greedy exploration, a revised terminal solver and the integration of domain knowledge as constraints. In our empirical evaluations, we use the *CrazyAra* engine on chess and crazyhouse as examples to show that these changes bring significant improvements to *AlphaZero*.

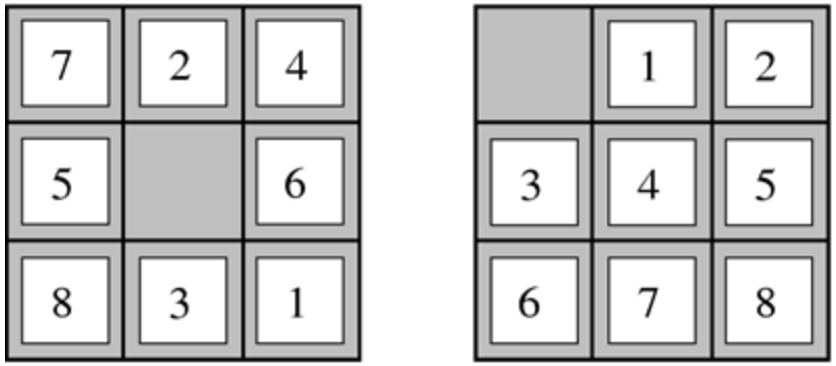
Introduction

The planning process of most humans for discrete domains is said to resemble the *AlphaZero* Monte-Carlo Tree Search (MCTS) variant (Silver et al. 2017), which uses a guidance policy as its prior and an evaluation function to distinguish between good and bad positions (Hassabis et al. 2017). The human reasoning process, however, may not be as stoic and structured as a search algorithm running on a

Figure 1: It is possible to obtain the King's Knight Opening (A) with different move sequences (B, C). Trajectories in **bold** are the most common move order to reach the final position. As one can see, graphs are a much more concise representation.

Admissible Heuristics: 8-Puzzle

- $h_{\text{MIS}}(n)$ = number of misplaced tiles
  - admissible because each misplaced tile must be moved at least once
- $h_{\text{MAN}}(n)$ = total Manhattan distance
  - i.e., no. of squares from desired location of each tile
  - admissible because this is the minimum distance of each tile to its target square
- Example:

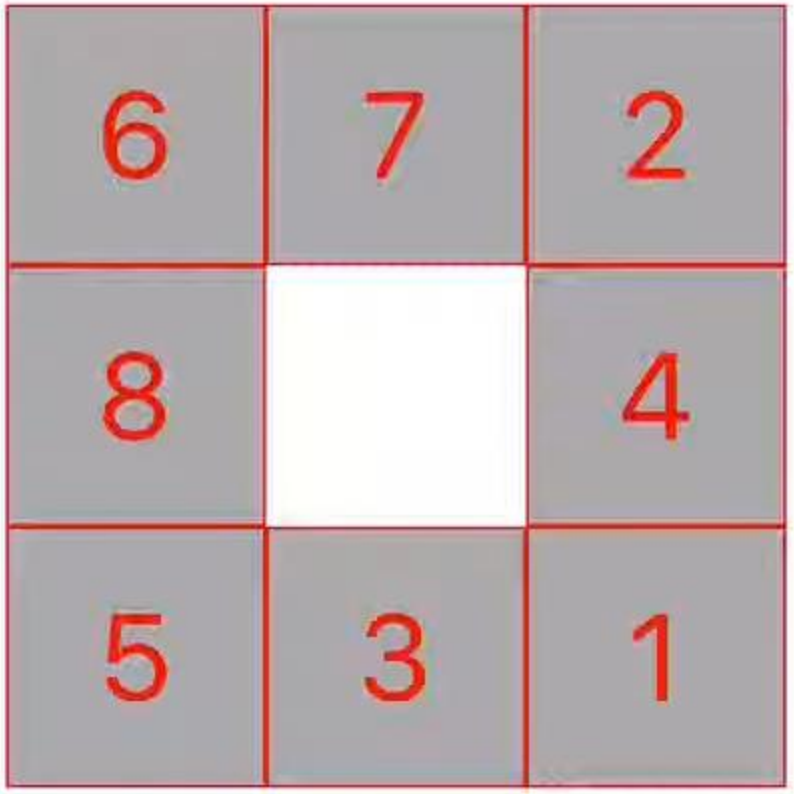


$h_{MIS}(start) = 8$

$h_{MAN}(start) = 18$

$h^*(start) = 26$

Solving 8-puzzle using A*



https://www.youtube.com/watch?v=_UyyJK3jeF4

Summary

- How to define problems
- How to find solutions for single-state problems
- The idea behind tree and graph search
- Some basic (un-)informed search strategies/algorithm
  - DFS, BFS,...
  - Greedy Best-first Search, A*
- Good heuristics can reduce the search time drastically

You should be able to:

- Formulate a real-world problem as a search problem
- Differentiate between search strategies
- Explain the difference between uninformed and informed search
- Iterate though DFS, BFS, A*,...
- Define properties of good heuristics

Next Time: Local Search Algorithms and Adversarial Search