Note

- Homework #1
  - Accessible in *Canvas's* "Assignment" section, click the link
  - On Lecture 1-3
  - Due in one week (2/18)
  - Save your hand-written or typed answers into a single PDF file, combining all pages in order of questions. Use any PDF scan app (e.g., CamScanner) or multifunction printers.
  - Submit the PDF file to "Submission for HW #1" link by 2/18 4pm. **No late submission allowed**! Strictly applied.
  - Must check all pages are present and readable after you submit!

KR-PF-PS

PF/PS: Search Methods

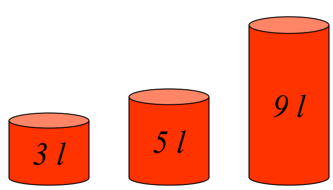
CSC 872

Pattern Analysis and Machine Intelligence

Review

- Last Lecture: Agent-based AI (KR&PF in AI)
  - Learned how to formulate a problem as an AI agent
  - View as the cycle of Percept-Reason-Action interacting with Environment
  - Environment types: PEAS
  - Agent types:
    - simple and model-based reflex agents
    - goal- and utility-based agents
    - learning agents
- Today
  - We will look at one instance of **actual implementation** of the agent-based program for Goal- and Utility-based ones

Measuring with Bucket



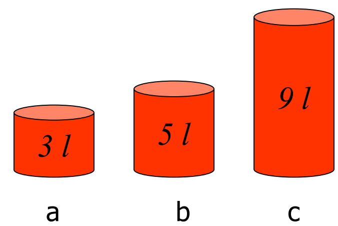
Problem: Using these three buckets, measure 7 liters of water.

From CSC561@USC

Measuring with Bucket

A Solution:

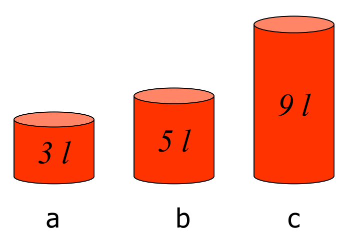
a	b	c	
0	0	0	start



Measuring with Bucket

Another Solution:

a	b	c	
0	0	0	start



Which solution is preferred?

- **Solution 1:**

a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	
3	3	6	
1	5	6	
0	5	**7**	**goal**

- **Solution 2:**

a	b	c	
0	0	0	start
0	5	0	
3	2	0	
3	0	2	
3	5	2	
3	**0**	**7**	**goal**

Problem-Solving Agent

- Four general steps to design this type of agent:

- **Goal formulation**
  – What are the successful world states
- **Problem formulation**
  – What actions and states to consider given the goal
- **Search strategy (Find Solution)**
  – Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.
- Execute
  – Give the solution perform the actions.

Perception

Reasoning

Action

Problem-Solving Agent

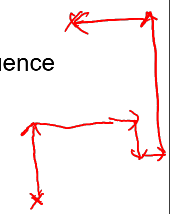
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation
   state ← UPDATE-STATE(state, percept)
   if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```


Example: Measuring with Bucket

- **Formulate goal:**
  - *Have 7 liters of water in 9-liter bucket*
- **Formulate problem:**
  - States: *amount of water in the 3 buckets*
  - Operators: *fill bucket from source, empty bucket to others*
- **Find Solution:**
  - *Sequence of operators that bring you from current state (0,0,0) to the goal state (x,x,7)*

Problem Types

- Deterministic, fully observable
  → single-state problem (chess) (buckets)
  - Agent knows exactly which state it will be in; solution is a sequence
- Non-observable Dead Reckoning
  → sensorless problem (walking in dark)
  - Agent may have no idea where it is; solution is a sequence

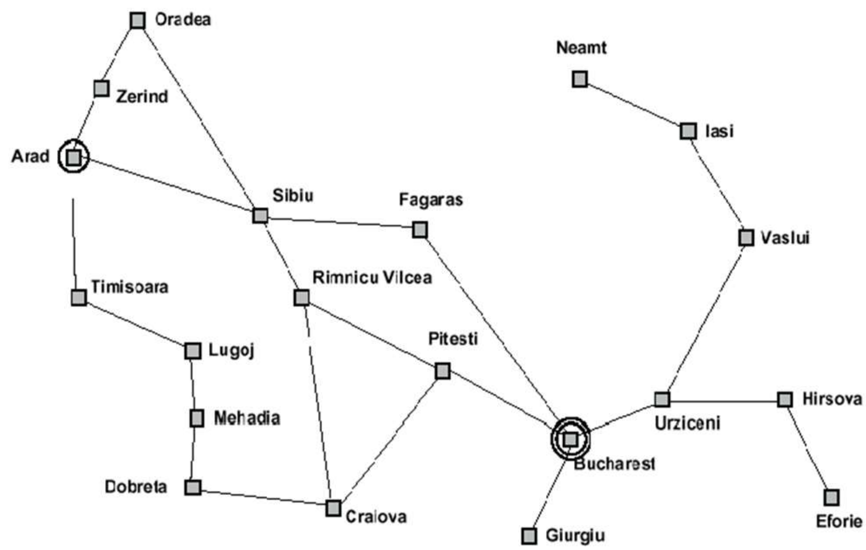


- Nondeterministic and/or partially observable
  → contingency problem (poker)
  - percepts provide new information about current state
  - often interleave search and execution
- Unknown state space
  → exploration problem (maze)

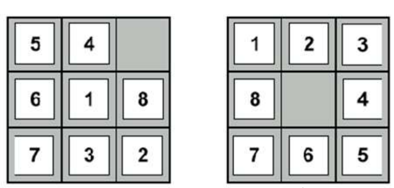
Toy Problem/Model

- Intended to illustrate or exercise various methods with concise and exact description
  - Vacuum World
  - Measuring with Buckets
  - …
- Real-World Problem is the one we want to solve but often hard to describe and solve
  - Robot navigation
  - Playing the game of Go
- Toy problem is used to explore and understand behavior of an algorithm for certain type of problem

Toy Problem: Romania



Toy Problem: 8-puzzle



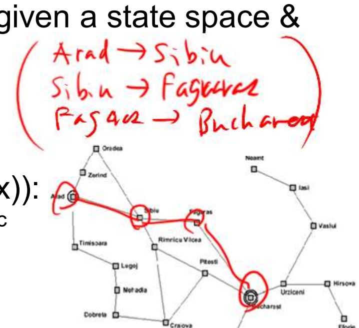
start state goal state

Selecting a State Space

- Real world is absurdly complex
  - *State space must be abstracted for problem solving*
- Abstracting a set of real **states**
  - "Arad" or "Zerind" represents a complex multi-aspect real city whose boundary may be difficult to define.
- Abstracting a complex combination of real **actions**
  - **Abstraction is to say "going from the city A to B costs L_{AB}" rather than actually driving from A to B on possible routes, detours, rest stops etc.**
- Abstracting a set of real paths that are **solutions** in the real world
  - What is true in the abstracted state space must also be true in the real world (**correctness**).
- Finding the right level of abstraction is difficult
- Each abstraction should be "easier" than the original problem

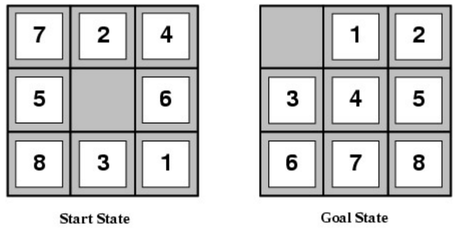
Problem Formulation

- A *problem* is defined by four items given a state space & a goal:



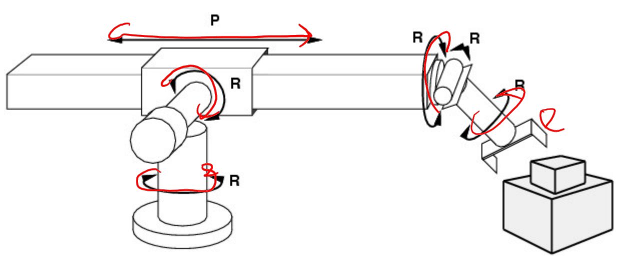
- **initial state**:
  - e.g., "at Arad"
- **operator** (or successor function S(x)):
  - e.g., "Arad → Zerind" and "Arad → Sibiu" etc
- **goal test**:
  - Explicit: "at Bucharest?"
  - Implicit: Checkmate(x)
- **path cost** (additive: how long traveled?):
  - e.g., "the sum of distances" and "number of operators applied" etc
- A *solution* is a sequence of operators leading from the initial state to a goal state

Example: 8-puzzle



- states?
- actions?
- goal test?
- path cost?
-

STOP: Example: Robot Hand



- states?
- actions?
- goal test?
- path cost?
-

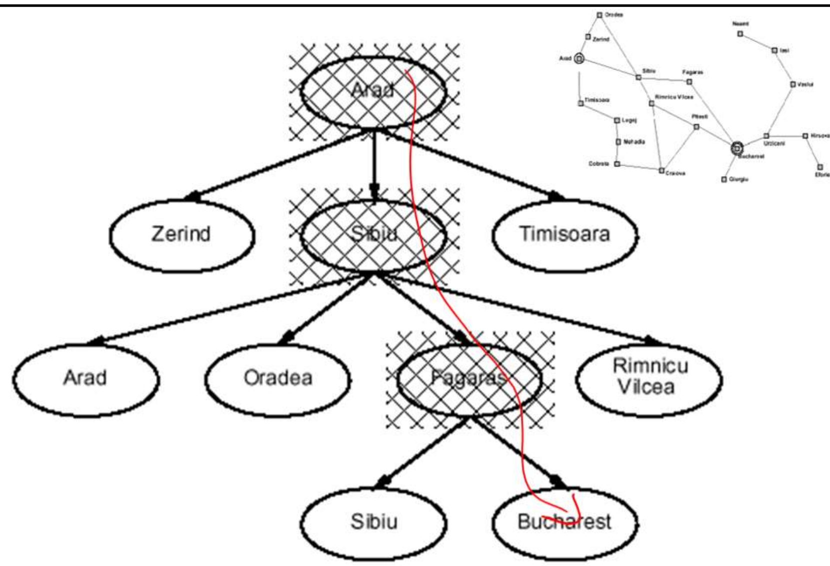
Search (Finding Solutions)

Basic idea: offline, systematic exploration of **simulated** state-space by generating successors of explored states (expanding)

Function General-Search(*problem*, ***strategy***) returns a *solution*, or failure
  initialize the search tree using the initial state
  loop do
    if no more candidates for expansion **then return** failure
    choose a leaf node for expansion according to the **strategy**
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add resulting nodes to the search tree
  end

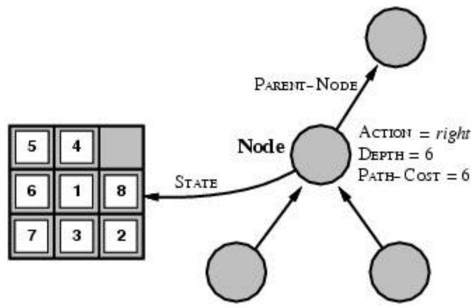
Strategy: the order of node expansion
Solution: a sequence from initial to goal states

Tree Search Example



State Space vs. Search Tree

- Tree node encapsulates state information
  - Node: State, Parent, Action, Depth, Path-Cost
  - **Expand**: create new nodes
  - **Operator**: create corresponding state



Search Strategy

- Order of node expansion defines a search strategy

- Strategies are evaluated in terms of:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?

- Time and space complexity are measured in:
  - ***b**:* maximum branching factor of the search tree
  - ***d**:* depth of the least-cost solution
  - ***m***: maximum depth of the state space (may be ∞)

Uninformed Search

Search Strategy

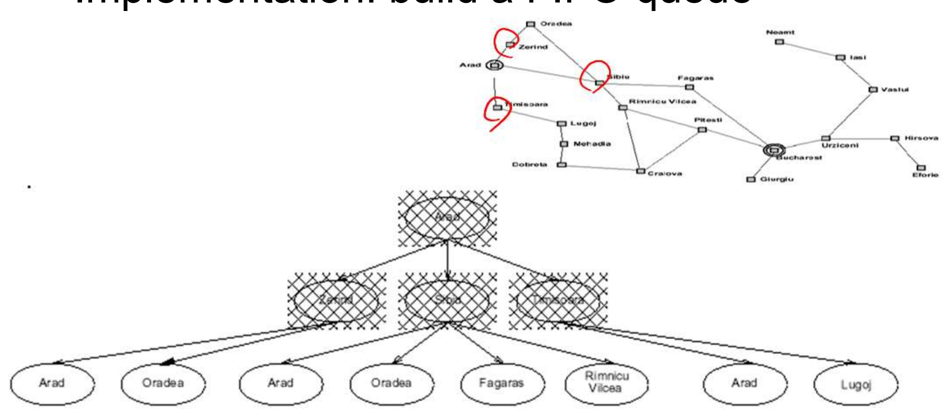
- Use only information available in the problem formulation (Blind Search)

- **Breadth-first**
- **Uniform-cost**
- **Depth-first**
- Depth-limited
- Iterative deepening
- Bidirectional

Breadth-First Search

BFS

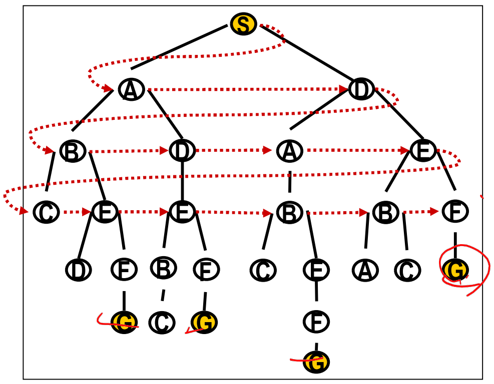
- Expand *shallowest* unexpanded node
- Implementation: build a FIFO queue



Breadth-First Search BFS

Completeness:	**Yes**, if *b* is finite
Time complexity:	$1+b+b^2+\ldots+b^d = O(b^d)$, i.e., exponential in *d*
Space complexity:	$O(b^d)$ *all visited must be stored*
Optimality:	**Yes** (assuming cost = 1 per step)

Move downwards level by level, until goal is reached

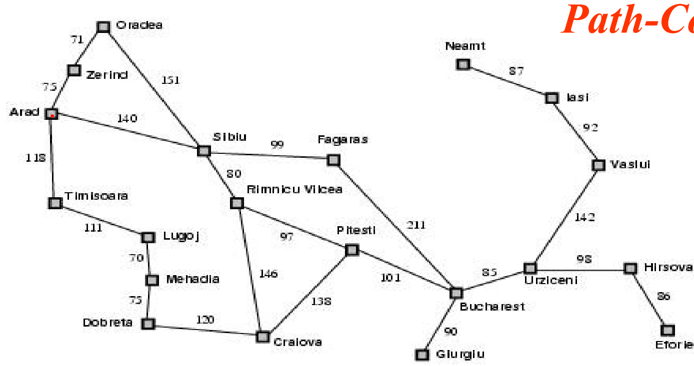


25

Uniform-Cost Search

- Expand node with *lowest path-cost*
- Implementation: a queue sorted by path-cost

Path-Cost ***g(n)***

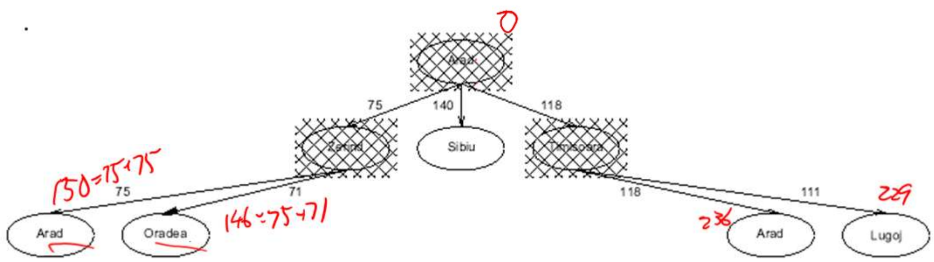


Romania with step cost in KM

26

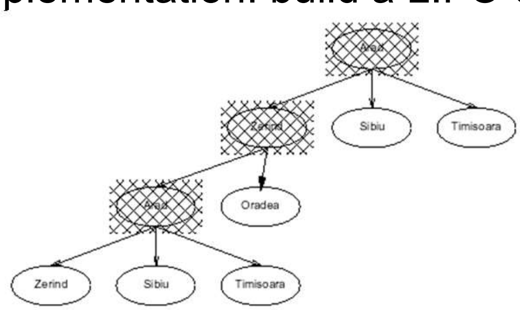
Uniform-Cost Search

Completeness:	**Yes**, if step cost ≥ ε>0
Time complexity:	≤ $\boldsymbol{O(b^d)}$
Space complexity:	≤ $\boldsymbol{O(b^d)}$
Optimality:	**Yes**, as long as path cost never decreases



Depth-First Search DFS

- Expand *deepest* unexpanded node
- Implementation: build a LIFO queue (stack)

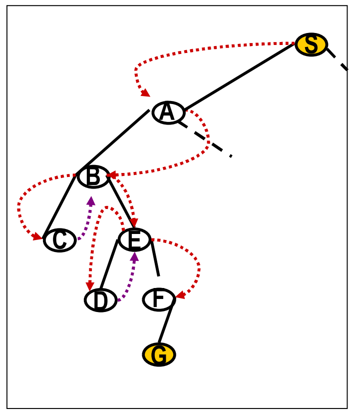


I.e., depth-first search can perform infinite cyclic excursions
Need a finite, non-cyclic search space (or repeated-state checking)

Depth-First Search

Completeness:	**No**, fails in infinite or cyclic state-space
Time complexity:	$O(b^m)$
Space complexity:	$O(bm)$
Optimality:	**No**

Move downwards as deep as you can, then back up



CSC872: PAMI – Kazunori Okada (C) 2025

29

29

Informed Search

- Use problem-specific heuristic to guide search
- Utility-based vs Goal-based Agent

- **Best-First Search**
- **Greedy Search**
- **A* search**
- Local Search (Revisited Later)
  - Hill-Climbing
  - Simulated Annealing
  - Local Beam Search

CSC872: PAMI – Kazunori Okada (C) 2025

30

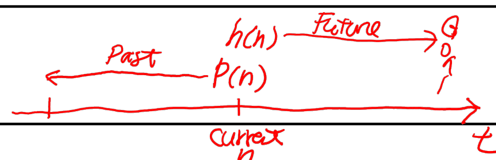
30

Best-First Search

Utility Function

- Idea: Use *evaluation function* *f*(*n*) to estimate *desirability* of each node
- Expand node that *appears* best (most desirable)
- Implementation: a queue sorted by *desirability*
- Special Case of *f(n)*
  - *Greedy Search*
  - *A* Search*

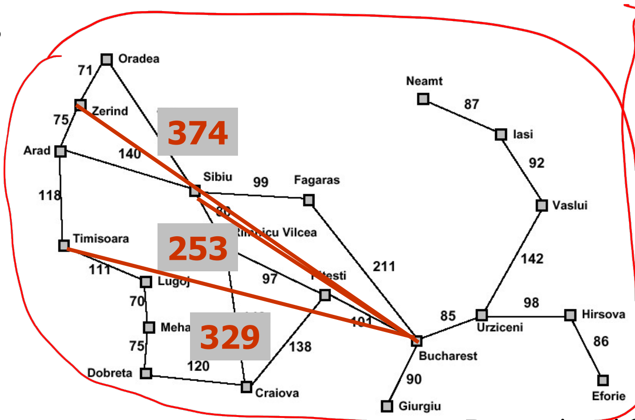
Heuristics



- [dictionary]*"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood."*
- $h(n)$ = **estimated** cost of the cheapest path from node *n* to goal node.
- If *n* is goal then $h(n)=0$

Straight-Line Heuristics

- $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest
-

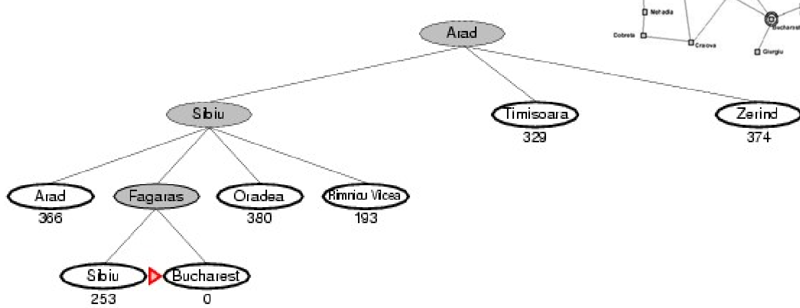


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Romania with Heuristics in KM

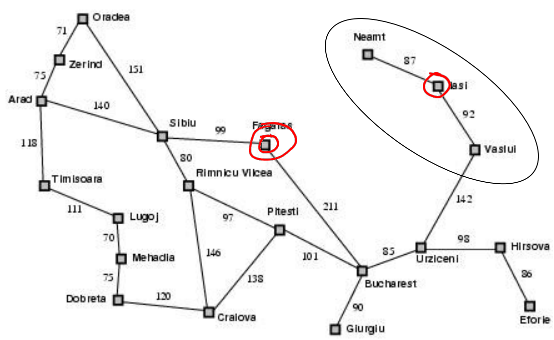
Greedy Search

- Expand node that appears to be *closest* to goal
- Implementation: $f(n) = h_{SLD}(n)$



Greedy Search

Completeness:	**No** (cf. DF-search)
Time complexity:	$O(b^m)$ but good heuristic can improve this
Space complexity:	$O(b^m)$ keep all nodes in memory
Optimality?	**No** (cf. DF-search)

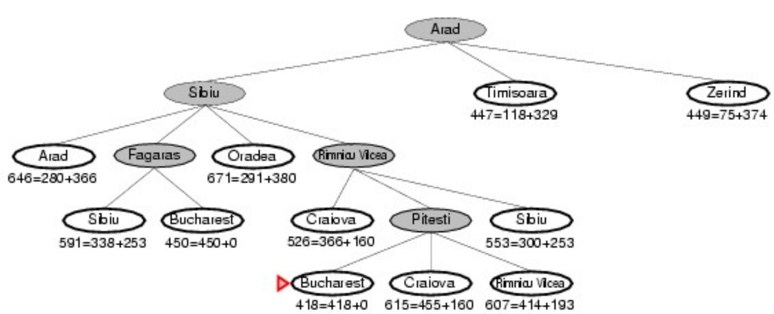


A* Search

- Avoid expanding paths that are already expensive
- Implementation: $f(n) = g(n) + h(n)$

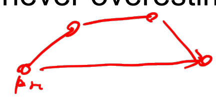
past future

path cost



Admissible Heuristics

- A heuristic is admissible if it *never overestimates* the true cost to reach a goal
- $h(n) \leq h^*(n)$ for all n where $h^*(n)$ is the *true cost* from *n*.
  - $h_{SLD}(n)$ is admissible because it never overestimates actual road distance.



- Admissible heuristic is optimistic

Optimality of A*

- Theorem: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal
- Complete?
  - **Yes** (unless there are infinitely many nodes with $f \leq f(G)$)
- Time?
  - Exponential in length of solution
- Space?
  - Keeps all nodes in memory
- Optimal?
  - **Yes** if $h(n)$ is **admissible**

Summary

- Overview
  - PF: Problem-Solving Agent
  - PF: Goal-based Problem Formulation
  - PS: Uninformed Search (Breadth-First, Depth-First)
  - PS: Informed Search (Greedy, A*)
- MATLAB exercise 2 after the break
- Work on HW1!
- Next Lecture
  - PF: Knowledge-based Agent
  - KR: Propositional Logic
  - PF: Logical Inference
  - PS: Resolution, Model Checking, Forward Chaining
  - MATLAB exercise 3