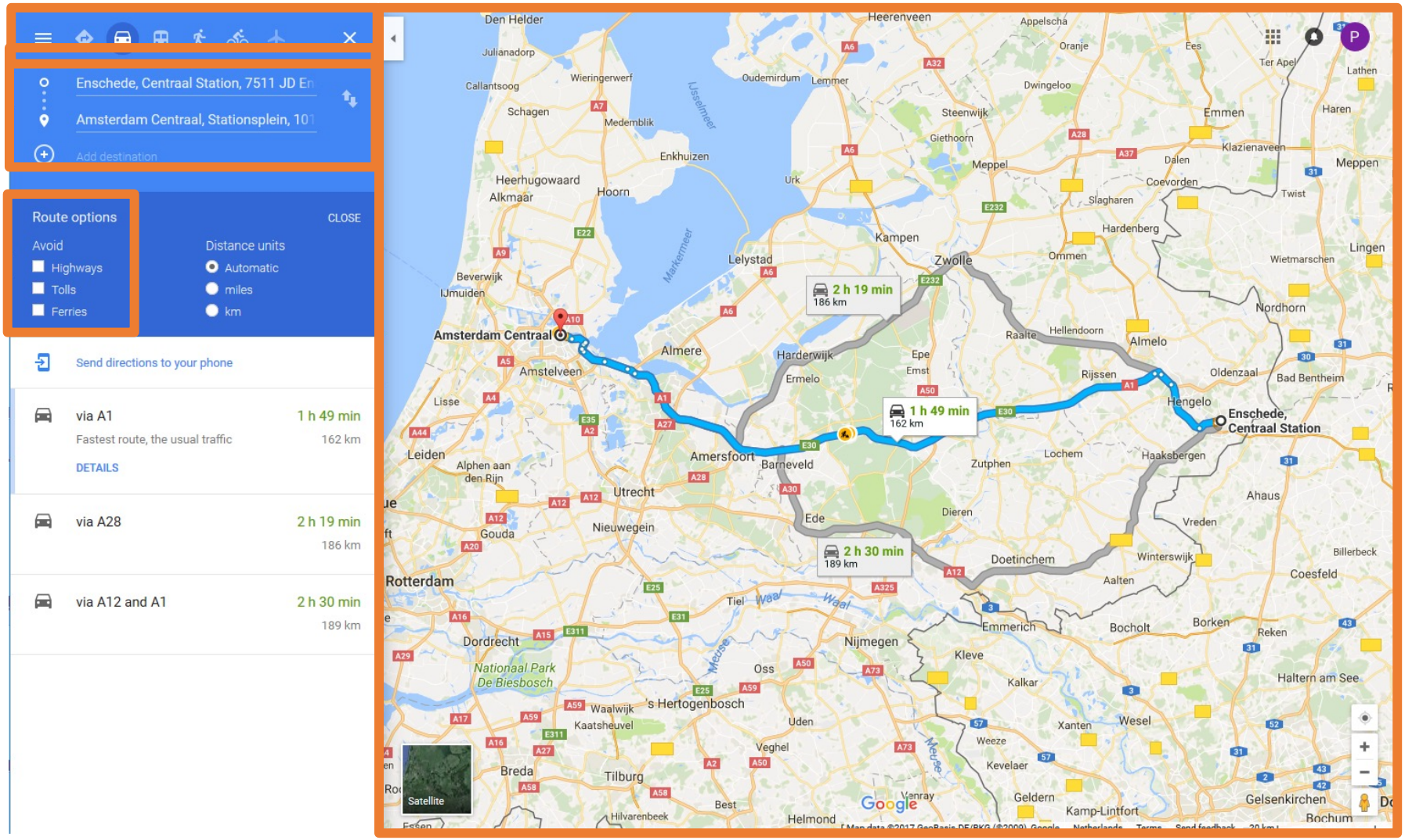
Search
Basic AI Technique
Part I: Defining a Search Problem

Course on AI@IID

Slides adapted from Mannes Poel & Pauline Chevalier
by M. Birna van Riemsdijk

Path Finding - example



Computational Technique

Machine Reasoning:
logic-based approaches

Week 1-3

Optimisation:
algorithmic approaches

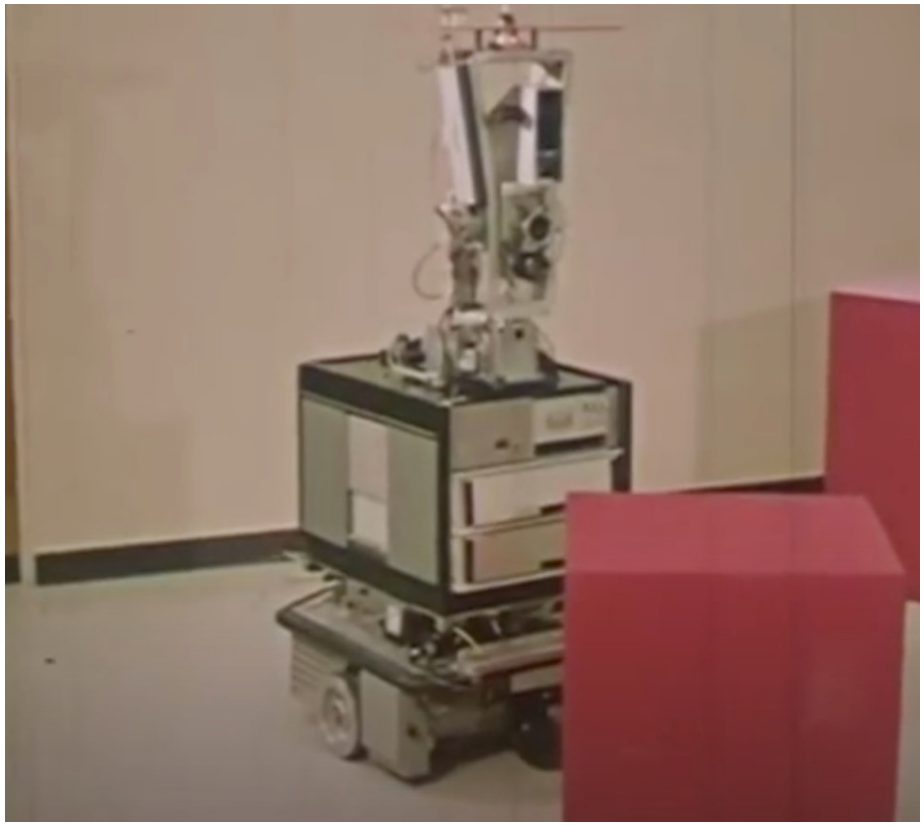
Week 2

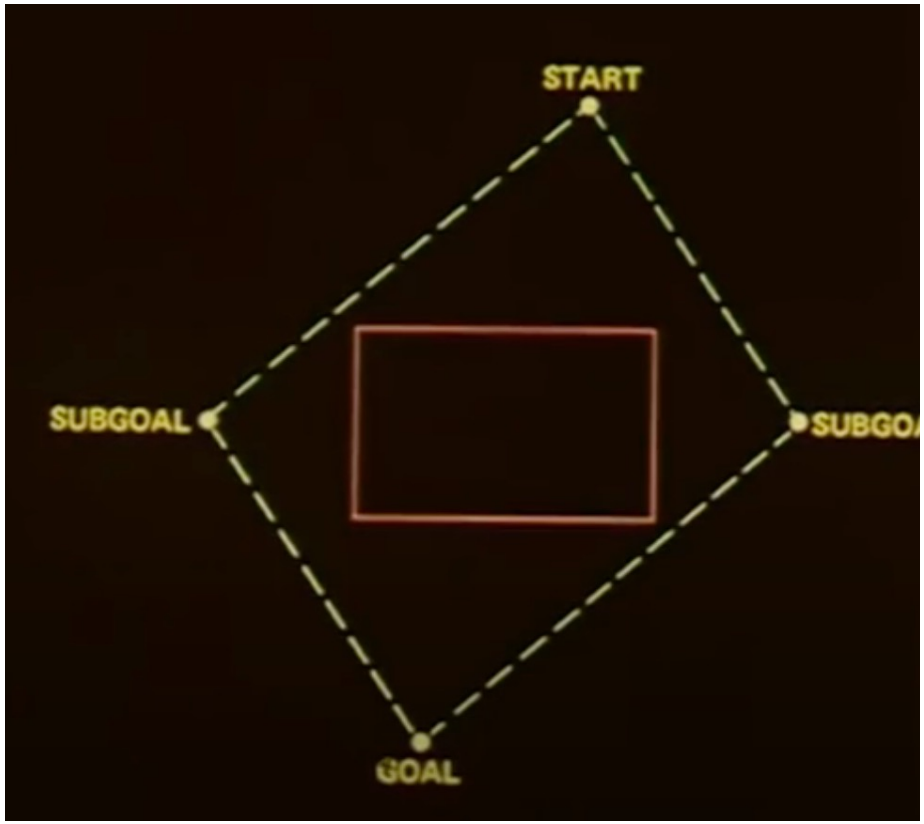
Machine Learning:
data-oriented approaches

Week 4-6

Robot Shakey (1966 – 1972)

https://youtu.be/7bsEN8mwUB8





Path Finding

- Goal-based agent
  - Has a goal and aims to achieve it
  - Considers future actions and the desirability of their outcomes
- Goal formulation
  - Based on the current situation and the agent's performance measure
- Problem formulation
  - The process of deciding what actions and states to consider, given a goal.

Path Finding : Problem formulation

- **Initial state** : position of the agent at start
  - In(A)
- **Actions** : possible actions available to the agent
  - In(A) , {Go(X), Go(Y), Go(Z)}
- **Transition model** : what each action does.
  - RESULT(In(A), Go(X)) = In(X)
- **Goal test** : tests if a given state is the goal state
- **Path cost** : assigns a numeric cost to each path

>> A **solution** to a problem is an action sequence that leads from the initial state to the goal state.

>> **Solution quality** : measured by Path cost function, the lowest being the optimal solution

Finding solutions : Main ingredients

- **Search space** : Set of all states reachable from the initial state by any sequence of action
- **Search tree or graph** : formed by all the possible action sequences starting at the initial state
  - Initial state : the root;
  - Actions : branches;
  - States in the space of the problem : nodes.
- **Search strategy** : the process of expanding nodes until either a solution is found or there are no more state to expand

Measuring problem-solving performances

Evaluation measures:

- **Time complexity**: how long does it take?
- **Space complexity**: how much memory is needed?
- **Completeness**: does it find a solution when there is one?
- **Optimality**: does the search strategy find the optimal solution?

Search
Basic AI Technique

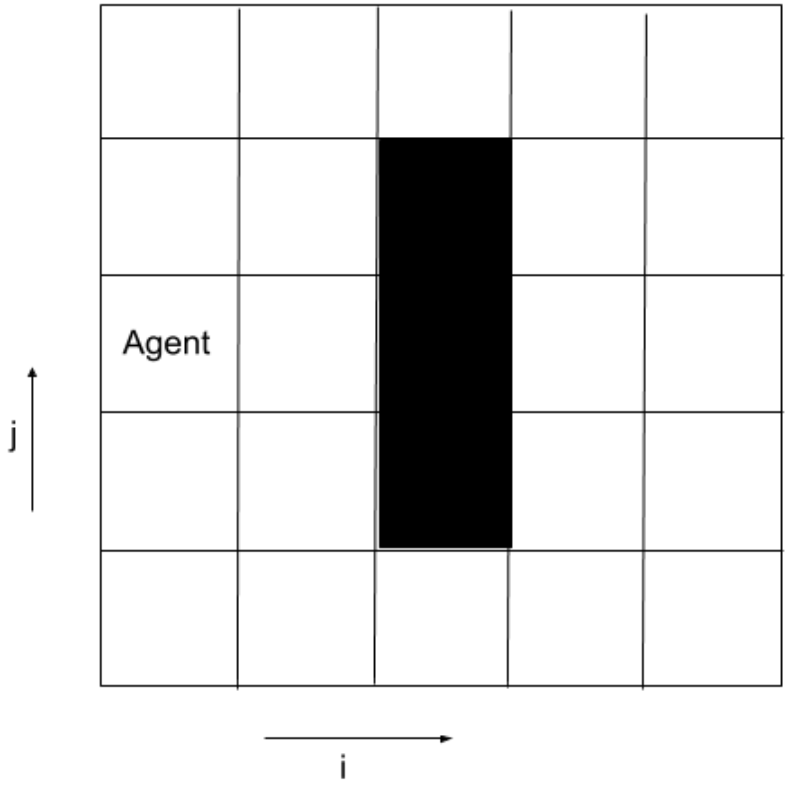
Part II: Path Search Example - Representation

Course on AI@IID

Slides adapted from Mannes Poel & Pauline Chevalier
by M. Birna van Riemsdijk

Simple example of path search

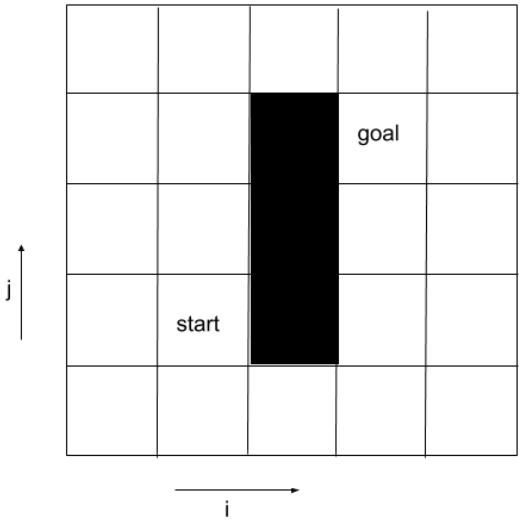
- Consider an agent *A* living in very small grid like world with obstacles.



This state representation is encoded as
In(1,3)

Simple example of path search

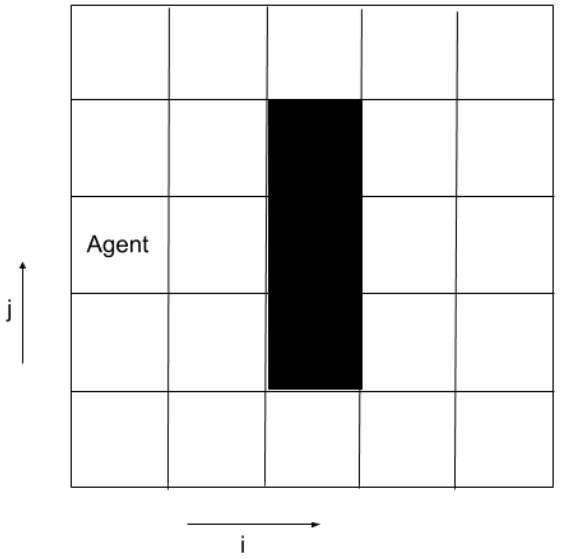
- *A* would like to go from *start* to *goal* in an optimal way (shortest path).
- Not by exploring the environment but finding an optimal path by thinking, in this case a search algorithm executed in the *"mind"* of the agent.
- *A* has complete view on the relevant part of the environment (observable) and should have a valid abstraction in its mind of this environment.
- Valid abstraction → abstract datatype



Start state: encoding *In(2,2)*
Goal state: encoding *In(4,4)*

Abstract datatype for Environment

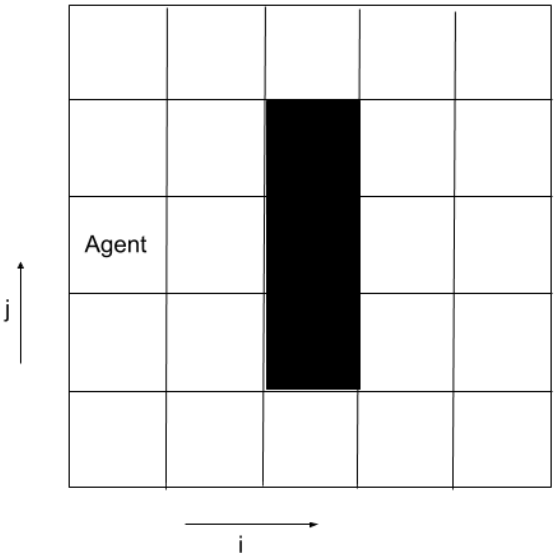
- Matrix *E* of size 5x5:
  - *E(i,j)=0* → free cell
  - *E(i,j)=2* → obstacle
  - *E(i,j)=1* → agent is in cell (i,j)
- For the current state of the environment on the right:
  - *E(i,j)=0* for all *i,j* except
  - *E(3,2)= E(3,3)= E(3,4)= 2*
  - *E(1,3)=1*
- Because only the agent can move the above state is decoded by *In(1,3).*



This state representation is encoded as *In(1,3)*

Abstract actions transforming the value of the datatype

- Actions: *left, down, right, up*
- Design decision: is action *left* allowed in *In(1,3)?* If yes what is the effect?
- Action *left* applicable in *In(i,j)* with *i>1* except for *(i,j) in {(4,2),(4,3),(4,4)}.*
- Action *down* applicable in *In(i,j)* with *j>1* except for *(i,j) in {(3,5)}.*
- Action *right* applicable in *In(i,j)* with *i<5* except for *(i,j) in {(2,2),(2,3),(2,4)}.*
- Action *up* applicable in *In(i,j)* with *j<5* except for *(i,j) in {(3,1)}.*

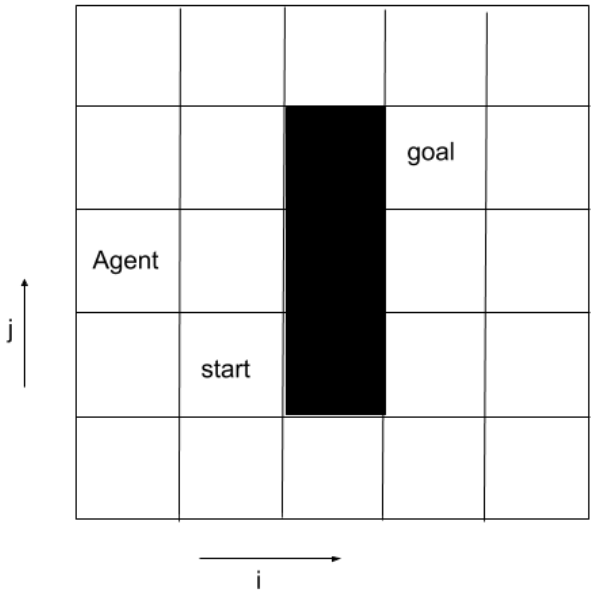


This state representation is encoded as *In(1,3)*

Effect of actions on the value of the datatype

- Actions: *left, down, right, up*
- Action *left:* when applicable in *In(i,j)* then next state will be *In(i-1,j).*
- Action *down:* when applicable in *In(i,j)* then next state will be *In(I,j-1).*
- Action *right:* when applicable in *In(i,j)* then next state will be *In(i+1,j).*
- Action *up:* when applicable in *In(i,j)* then next state will be *In(i,j+1).*

- Additional info: Goal state *In(4,4)* and Initial state *In(2,2).*



Start state: encoding *In(2,2)*
Goal state: encoding *In(4,4)*

Search
Basic AI Technique
Part III: Basic Search Strategies

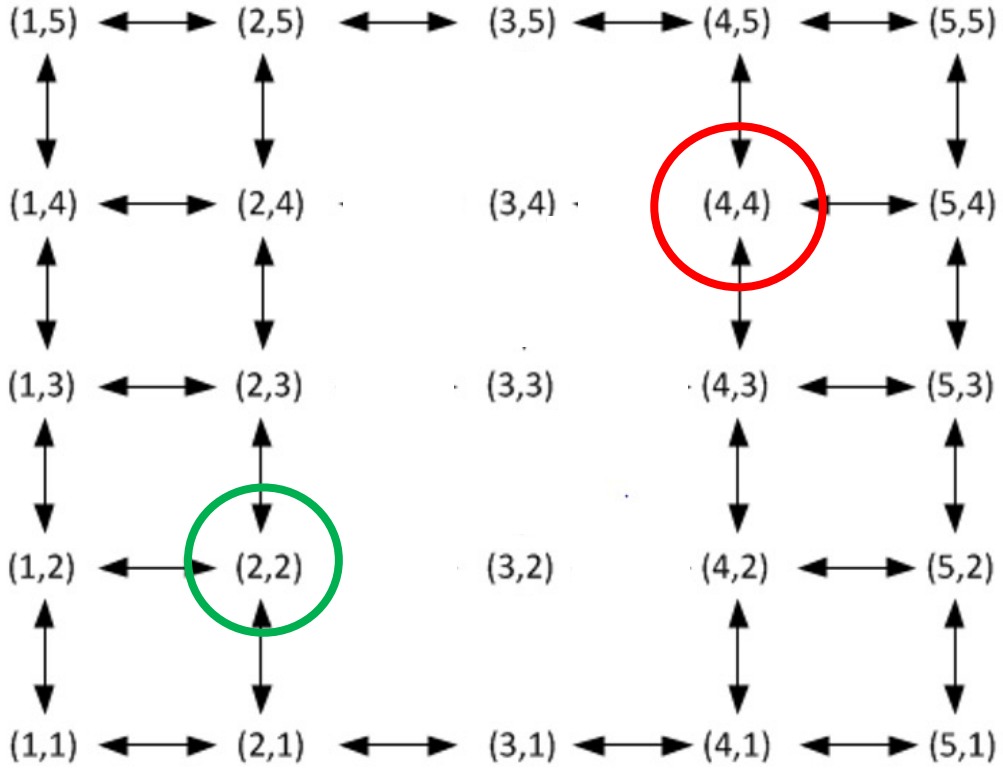
Course on AI@IID

Slides adapted from Mannes Poel & Pauline Chevalier
by M. Birna van Riemsdijk

Graphical representation of the modeling

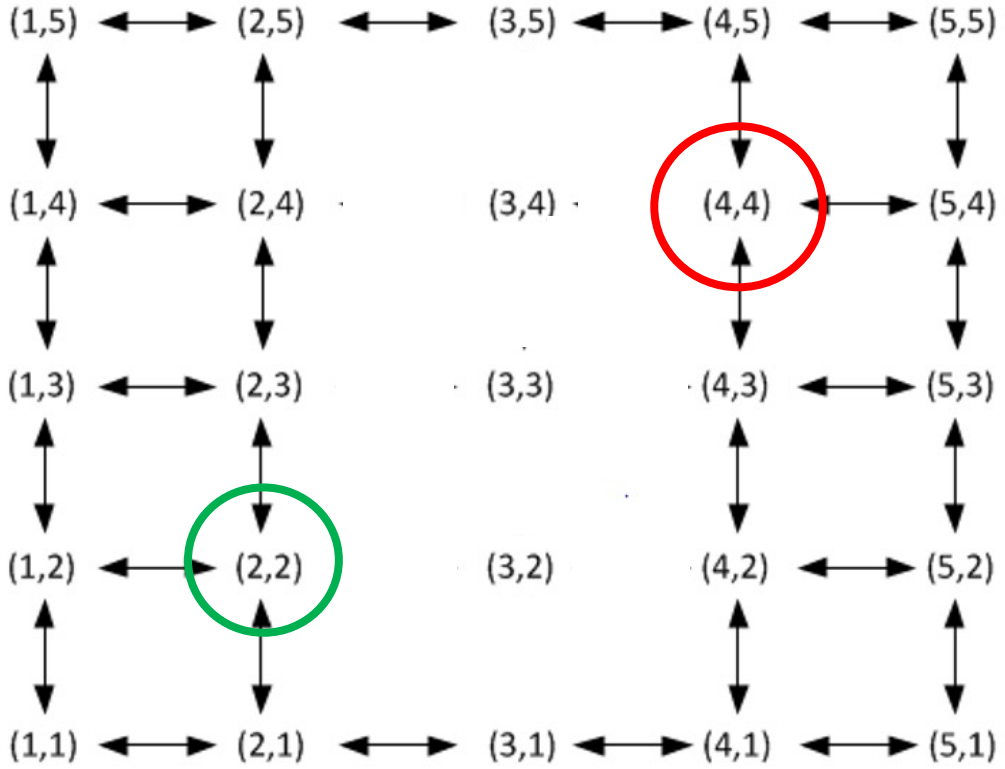
- 2 axis world
- 4 possible actions :
  - Left, down, right, up

- Initial state (2,2)
- Goal state (4,4)
- Problem: find shortest path from initial state to goal state.



Simple example of path search

- Ingredients to solve the problem :
  - Tree
  - List (queue) of frontier nodes (states): *frontier*



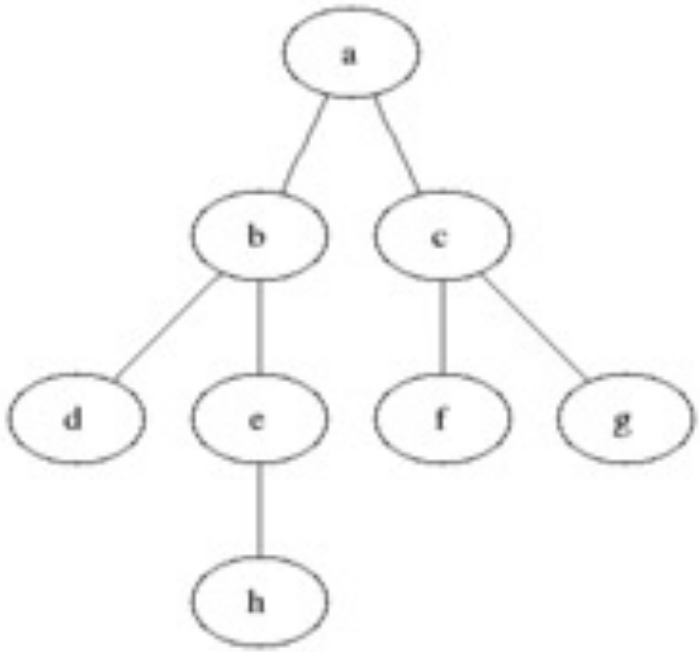
Tree search - Breadth first search

- Initialize frontier using start/initial node (node corresponding to initial state:
  - frontier=[((2,2),Xx)]
- **Loop do**
  1. **If** frontier=[] **then return** failure
  2. **Remove** head node *n* **from** frontier
  3. **If** *n* **contains** goal state **then return** solution
  4. **Expand** *n,* **add** resulting nodes to the tail of the frontier

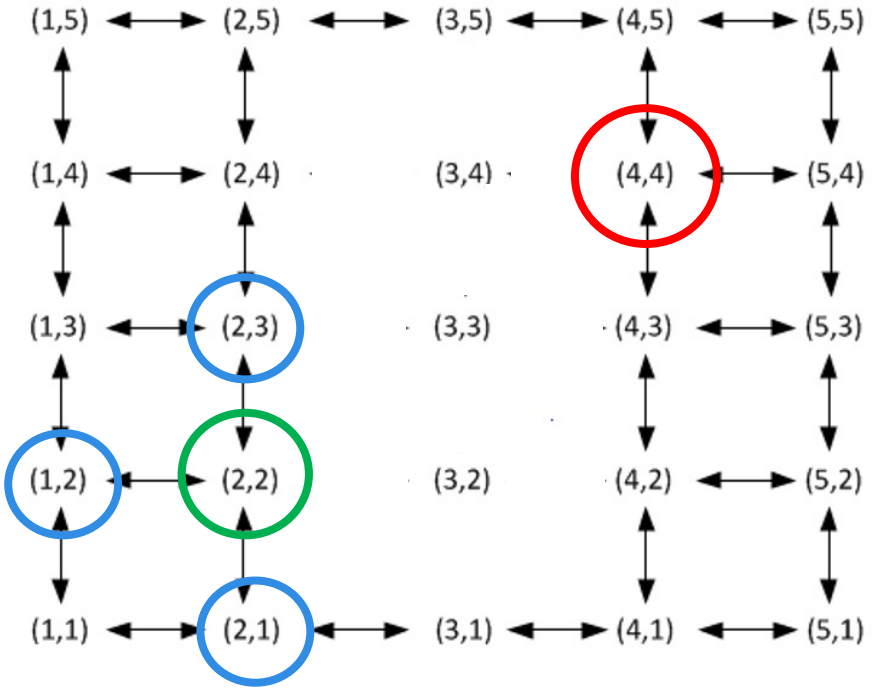
Xx: Additional node info depending on search strategy

Breadth first search

- Simple strategy in which
  - First, the root node is expanded,
  - Then all the successors of the root node are expanded next,
  - Then their successors,
  - And so on…



Tree search - Breadth first search



frontier = [**((2,2), Xx)**]

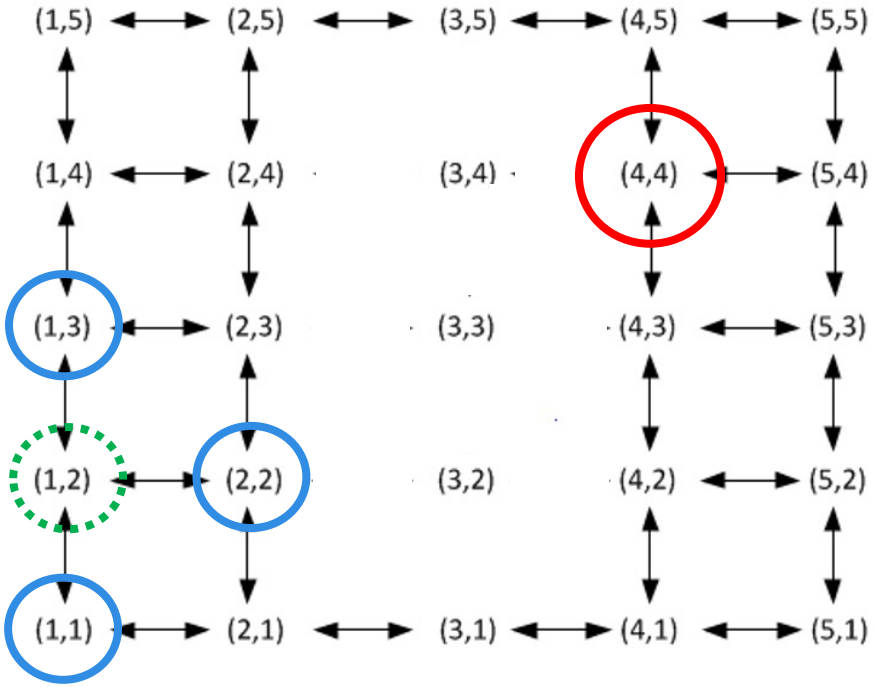
expand

[((1,2), Xx), ((2,1), Xx), ((2,3), Xx)]

Add to *f*

frontier = [((1,2), Xx), ((2,1), Xx), ((2,3), Xx)]

Tree search - Breadth first search



frontier = [**((1,2), Xx)**, ((2,1), Xx), ((2,3), Xx)]

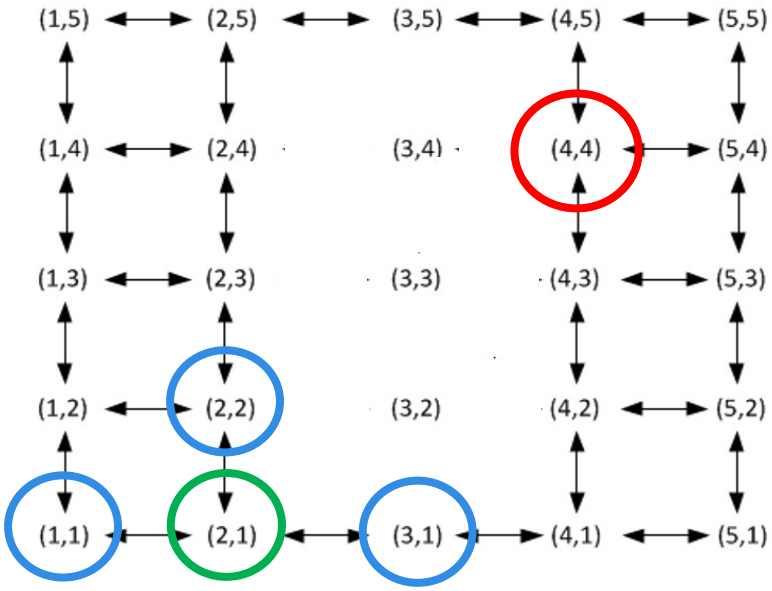
expand

[((1,1), Xx), ((2,2), Xx), ((1,3), Xx)]

Add to *f*

frontier = [((2,1), Xx), ((2,3), Xx),
((1,1), Xx), ((2,2), Xx), ((1,3), Xx))]

Tree search - Breadth first search



frontier = [**((2,1), Xx)**, ((2,3), Xx), ((1,1), Xx), ((2,2), Xx), ((1,3), Xx)]]

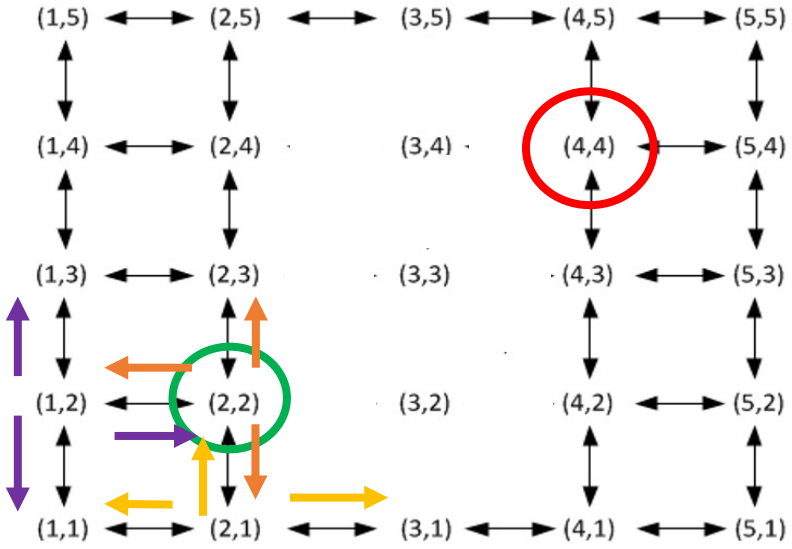
expand

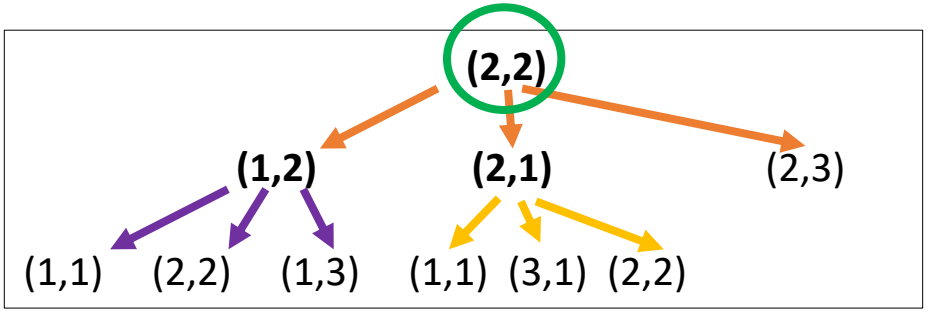
[((1,1), Xx), ((3,1), Xx), ((2,2), Xx)]

Add to *f*

frontier = [((2,3), Xx),
((1,1), Xx), **((2,2), Xx)**, ((1,3), Xx),
((1,1), Xx), ((3,1), Xx), **((2,2), Xx)**]]

Tree search - Breadth first search: Constructed search tree



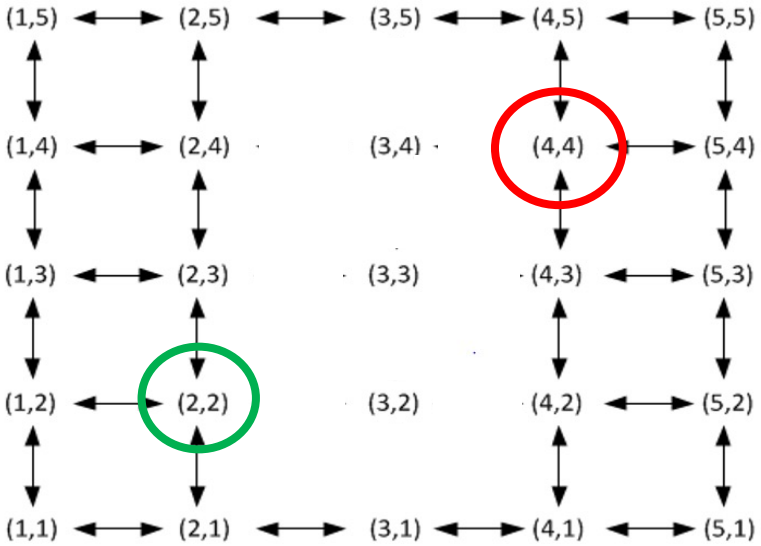


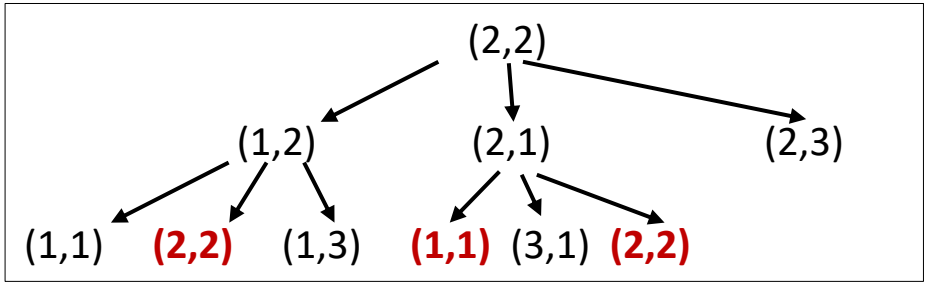
1st expansion: (2,2)
2nd expansion: (1,2)
3rd expansion: (2,1)

Breadth-first search:

- the order in which nodes are expanded is “layer by layer”
- implemented by adding nodes to the tail of the frontier, such that the nodes that were added first, are expanded first (First In First Out – FIFO queue)

Tree search - Breadth first search





- Repeated states !
- Uninformed search strategy: No information on goal state (4,4) used !

>> Need to find a solution for repeated states

Tree search - Breadth first search

- Solution : to augment the Tree search with a data structure called the **explored set** or **closed list** (kind of bookkeeping)

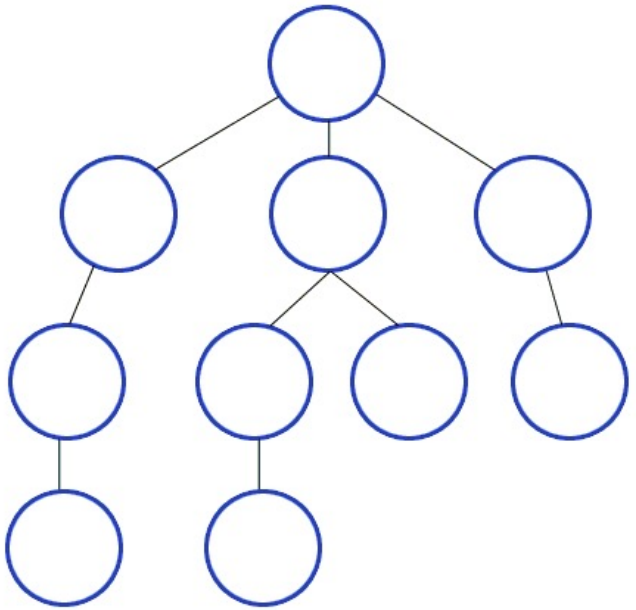
Graph search - Depth first search

- Initialize closed list and frontier using initial state:
  - frontier=[((2,2),xx)]
  - closed=[]
- **Loop do**
  1. **If** frontier=[] **then return** failure
  2. **Remove** head node *n* **from** frontier **and add** corresponding state to closed
  3. **If** *n* **contains** goal state **then return** solution
  4. **Expand** *n,* **add** resulting nodes to the head of the frontier if state is not in frontier and not in closed.

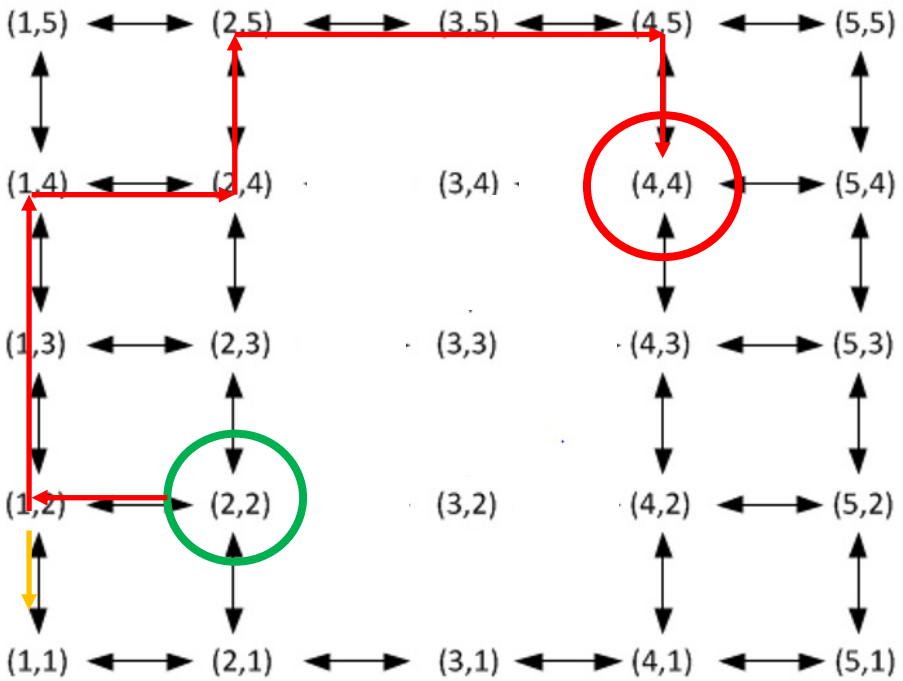
xx: Additional info depending on search strategy

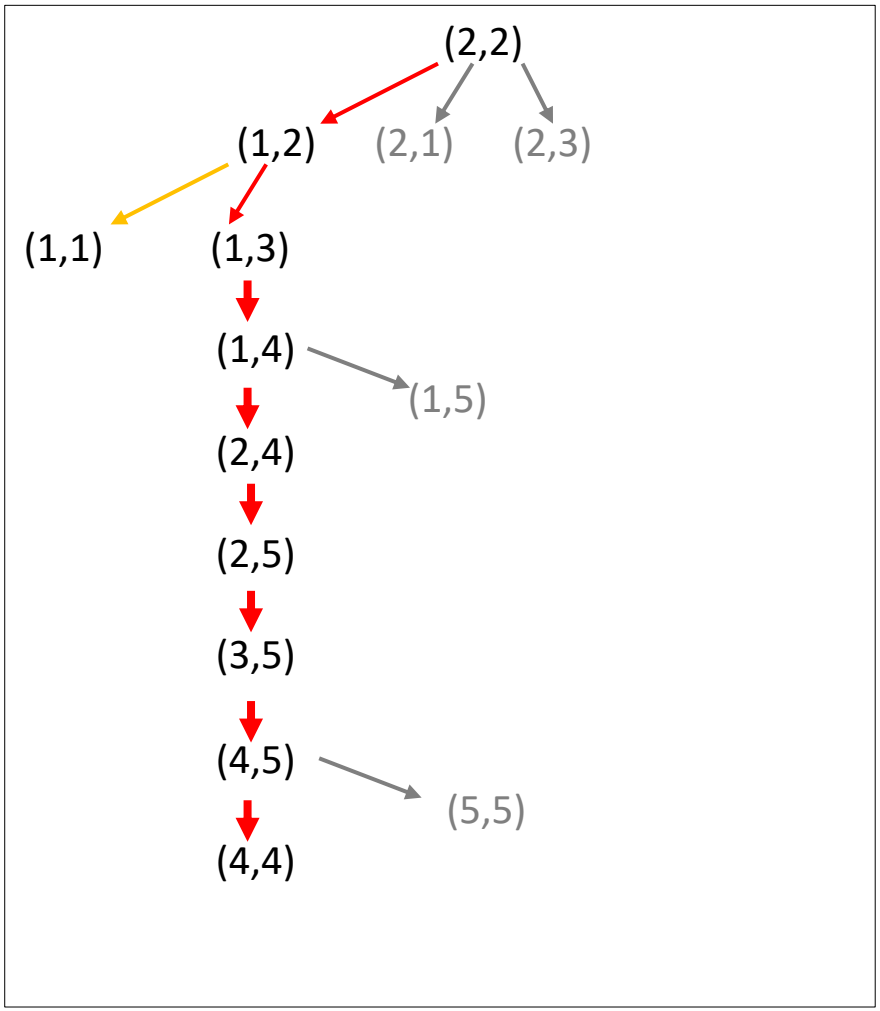
Depth first search

- Simple strategy in which the deepest node is expanded



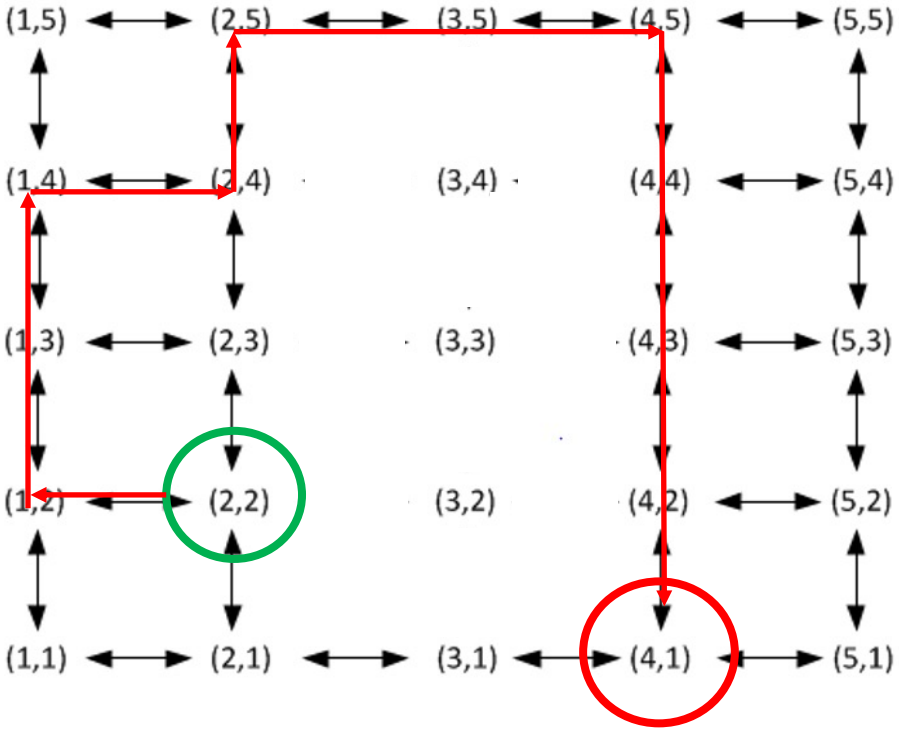
Graph search - Depth first search





Graph search - Depth first search

What will happen if the goal state would be *In(4,1)?*



Moreover different order of actions would lead to completely different path.

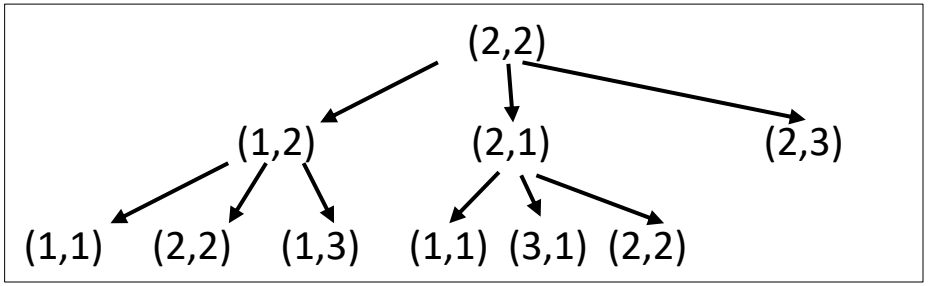
Confusing terminology: Tree Search and Graph search

	Action space	Execution	Closed list?	Search strategy
Tree search	Graph or tree	Search tree	No	BFS or DFS
Graph search	Graph or tree	Search tree	Yes	BFS or DFS

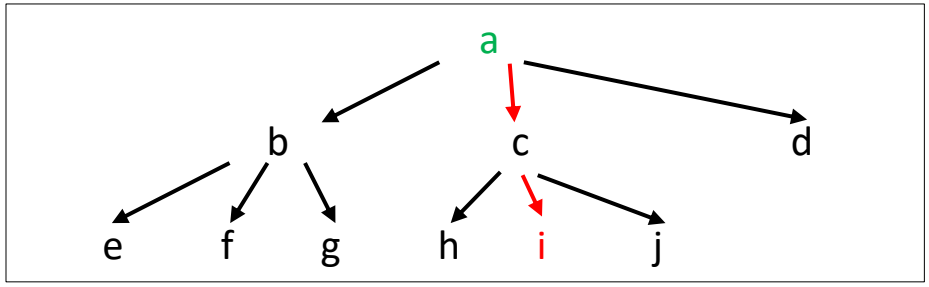
- The difference between *tree search* and *graph search* lies in the use of a closed list
- Benefits of a closed list
  - Search efficiency: Prevents expanding states that were expanded before
  - Prevents searching in loops
  - But: requires more memory due to bookkeeping
- When the problem (= action space) is tree-shaped, no loops occur in the problem specification, so we can use tree search

Question

- Which path would be generated by a breadth-first graph search algorithm?
- When asked to execute (parts of) a search algorithm, *always* draw a search tree!
  - Breadth-first search expands nodes layer by layer
  - Depth-first search expands the deepest node first
  - Frontier: leaf nodes in the search tree



Tree as problem specification



- Problem specification, i.e., specification of the search space (= action space), as a tree
- A link from one node *m* to the next node *n* specifies that the agent can take an action to move from state *m* to state *n*

- Question: find a path from the root node of the tree (a) to the goal node (i) using tree search
- Convention: when using a visual tree to specify the problem or the search tree, nodes are expanded from left to right in case of uninformed search strategies, i.e., are assumed to occur in this order in the frontier constructed while performing the search
- Nodes are removed from the frontier and expanded in the following order until the goal (i) is found
  - BFS: a, b, c, d, e, f, g, h, i
  - DFS: a, b, e, f, g, c, h, i

Uninformed (or blind) search strategies

- No additional information about states beyond the one provided in the problem definition
- Can only generate successors and distinguish a goal state from a non-goal state
- Different algorithms possible
  - Breadth first search;
  - Depth first search;
  - Depth-limited search;
  - Iterative deepening search;
  - Uniform cost search.

Artificial intelligence…

Questions:

What have these basic search algorithms to do with AI?

In other words: where is the “intelligence”?

Search
Basic AI Technique
Part IV: A* Search

Course on AI@IID

Slides adapted from Mannes Poel & Pauline Chevalier
by M. Birna van Riemsdijk

Informed (heuristic) search strategies

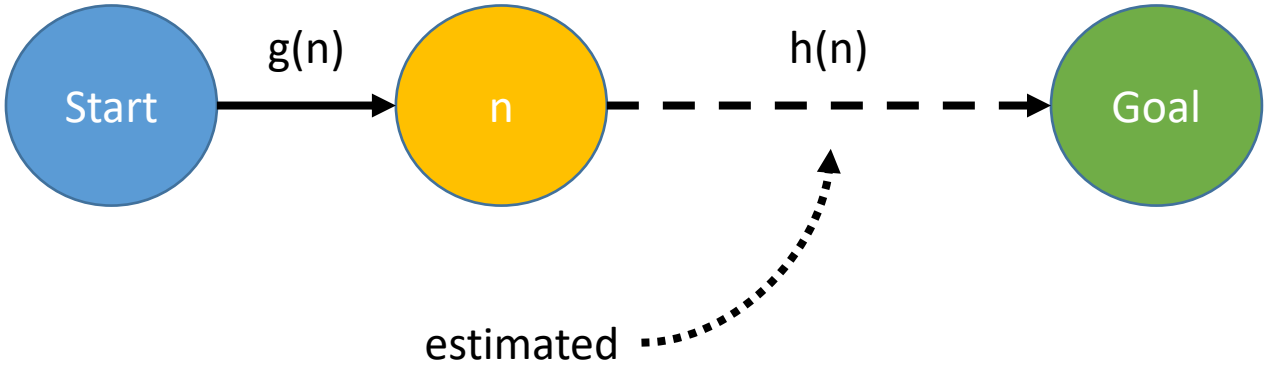
- Use problem-specific knowledge beyond of the problem itself
- More efficient than uninformed search strategies
- Different algorithms possible
  - Greedy best-first search
  - A* search
  - Recursive best-first search (RFBS)
  - Simplified memory-bounded A* (SMA)

Informed (heuristic) search strategies

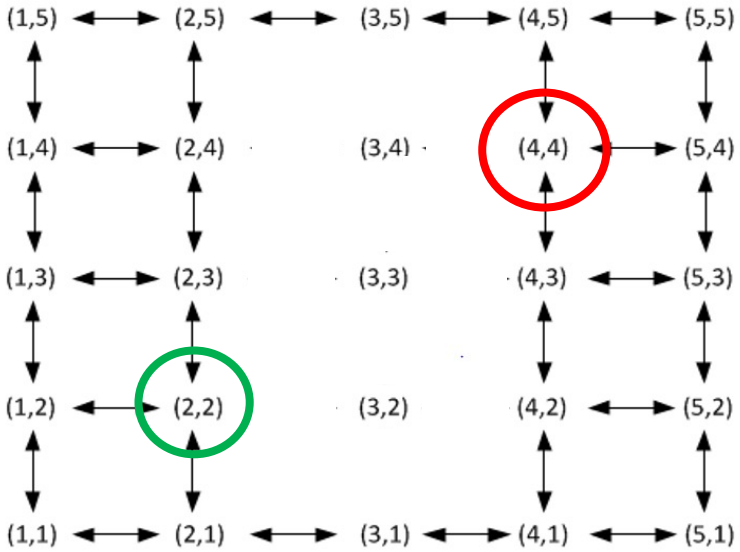
- General approach considered : the Best-first search
  - Instance of Tree or Graph-search algorithm.
  - The node is expanded based on an **evaluation function**, *f(n)* .
  - The node with the lowest cost is expanded first
  - Most Best-first search algorithms use a heuristic function, *h(n)*:
    - *h(n)* = *estimated* cost of the cheapest path from the state at node *n* to a goal state

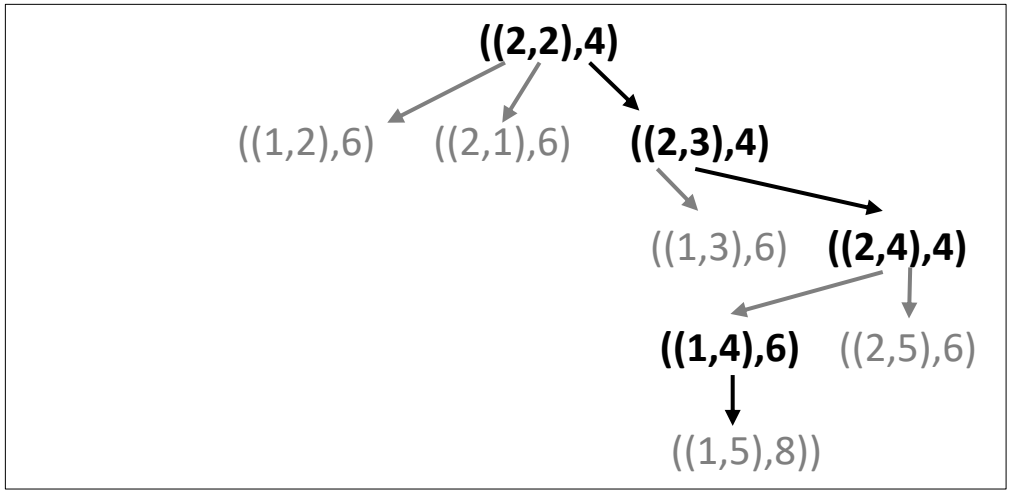
A* search

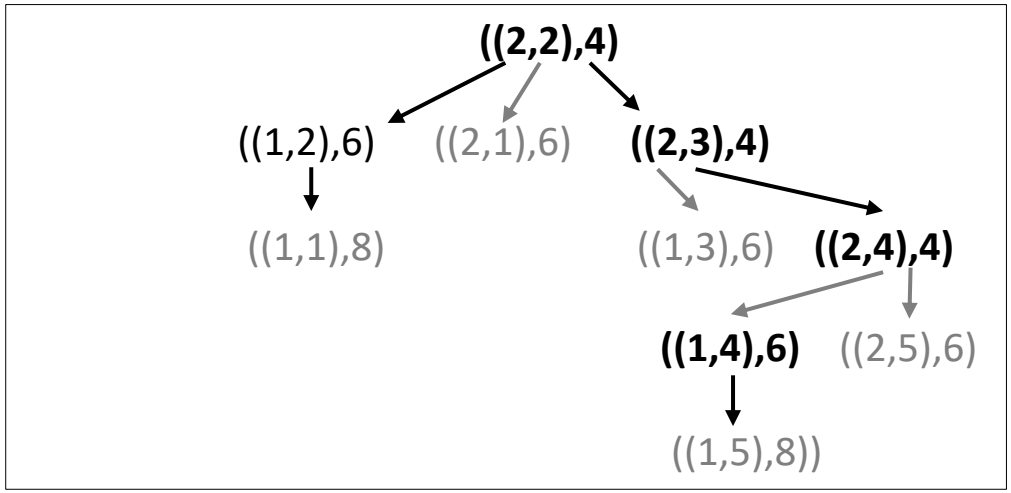
- Evaluates the node by combining the cost to reach the node n from the start state, denoted by *g(n)*, and the *estimated* cost to get from the node n to the goal, denoted by *h(n)*
  - *f(n) = g(n) + h(n)*
  - *f(n)* = estimated cost of the cheapest solution through *n*
- An estimated cost function is called a *heuristic* (*h(n)*)
- Choose the next node that has the smallest *f(n)*



A* search



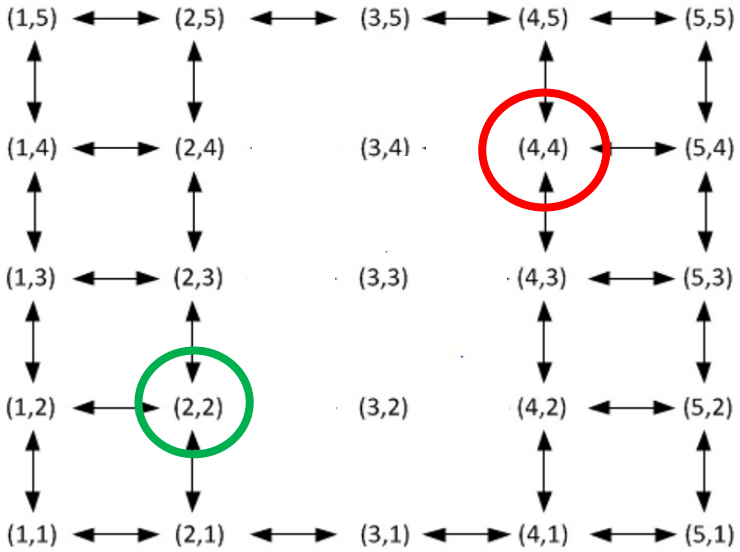




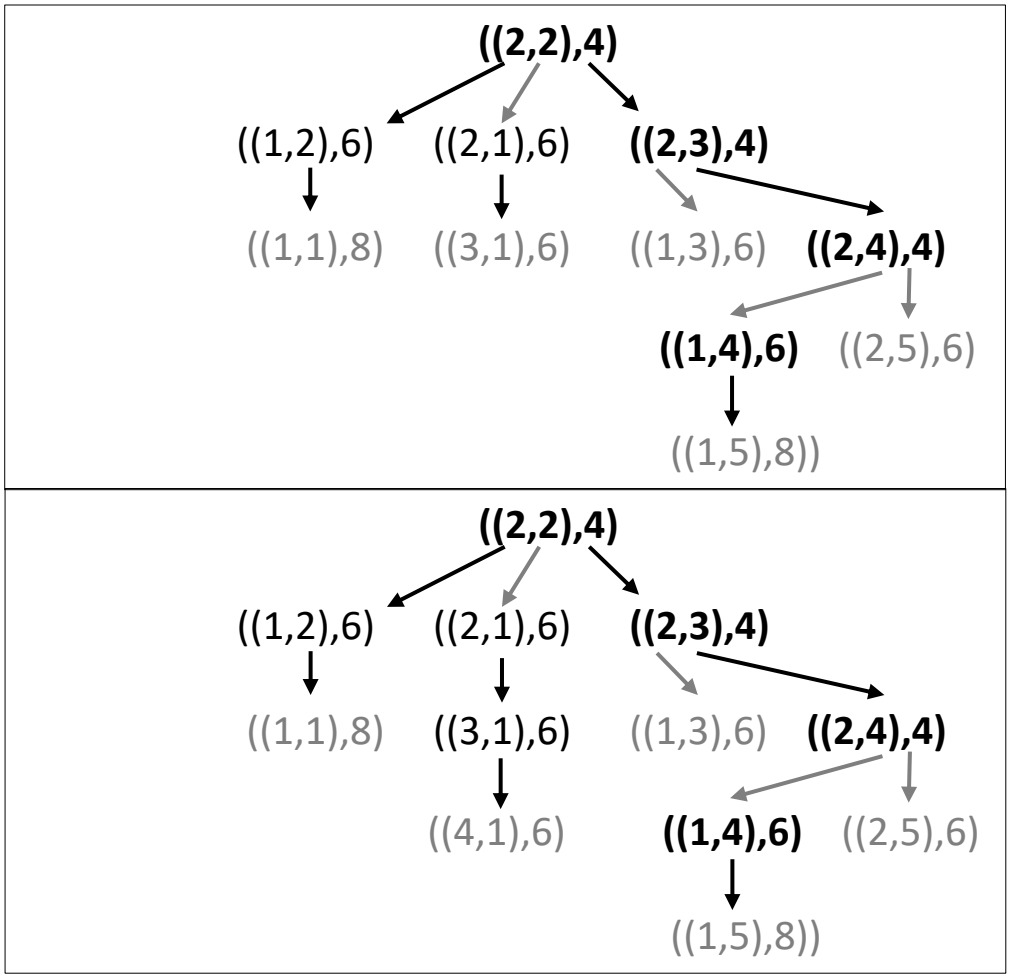
- *g* : step cost 1 per action
- *h :* Manhattan distance to goal

- f(n) = g(n) + h(n) added to node info, e.g., ((2,2), f(n) = 0 + 4 = 4))

A* search



- *g* : step cost 1 per action
- *h :* Manhattan distance

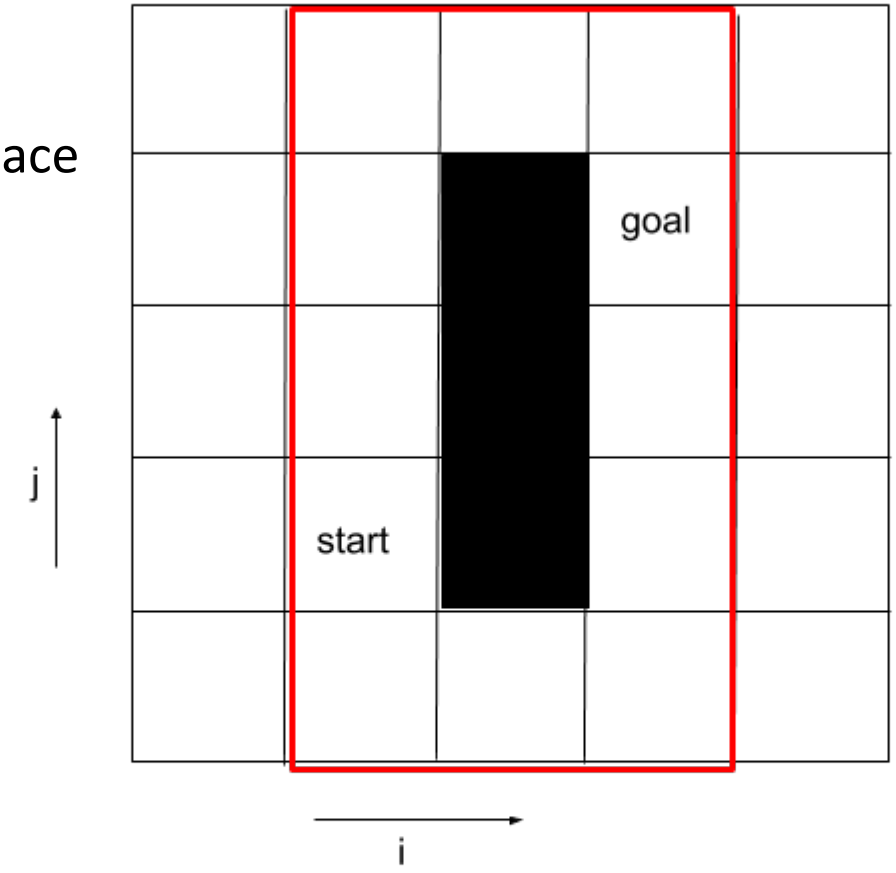


Depending on the choice made (ordering of the queue) we get the optimal path via the top (choosing ((2,5),6) etc.) or via the bottom (choosing ((3,1),6) etc.)

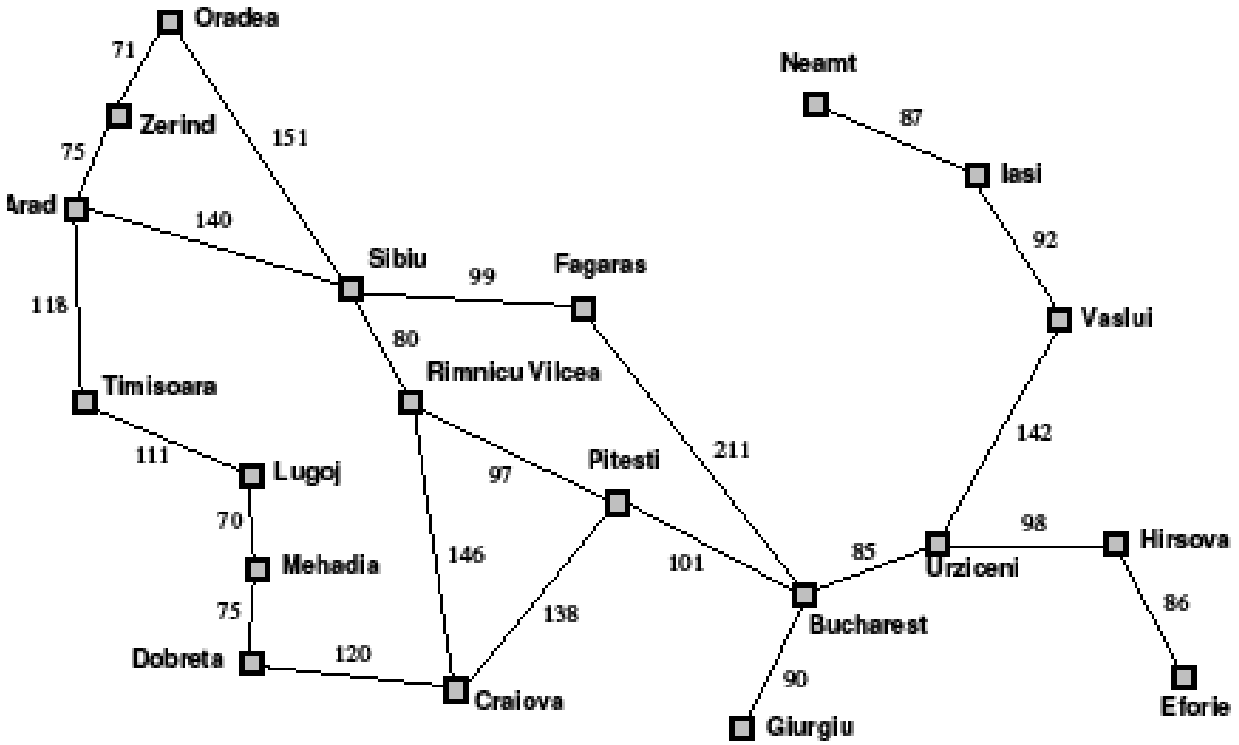
A* search

- This algorithm reduces search space
  - Initial: (2,2)
  - Goal: (4,4)

  - Red rectangle: reduced search space



Another example from book
Route planning



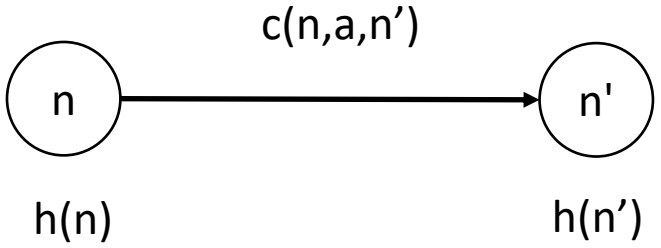
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Heuristic function: straight line distance

A* search - Properties

- A* search is complete (on finite graphs) and
  - Tree search version is optimal if *h* is *admissible.*
  - admissible: *h* never overestimates the cost, i.e. *h* is an optimistic estimate.

  - Graph search version is optimal if h is *consistent* (aka *monotonic*) :
  - consistent: h(n) ≤ cost(n,a,n') + h(n') with n' one-step successor of n (Triangle inequality)



- A *good* heuristic function can reduce the space and time complexity dramatically.

A* search – Terminology ‘Completeness’

- A* Search is *complete*: it always finds a solution if there is one
- A* Search with a good heuristic reduces the search space
  - the search tree generated by executing the algorithm will typically not contain all nodes in the problem graph or tree.
  - A* search will not explore the *complete* search space.

A* search - Commitment

- (A*) Search is an example of blind commitment. The plan (route) is not updated during the execution. For instance traffic jam info or moving obstacles are not taken into account.

Search requirements

- Environment should be fully observable and static.

- Search space modelling should be *sound* and *complete*.
  - Sound: very path in the abstract search space corresponds to a path in the real environment.
  - Complete: every path in the real environment corresponds to a path in the abstract search space.

What did we learn

- Basic search strategies:
  - Tree Search versus Graph Search
  - Uninformed: Depth First, Breadth First
  - Informed: A* search

- More search strategies can be found in the book.