

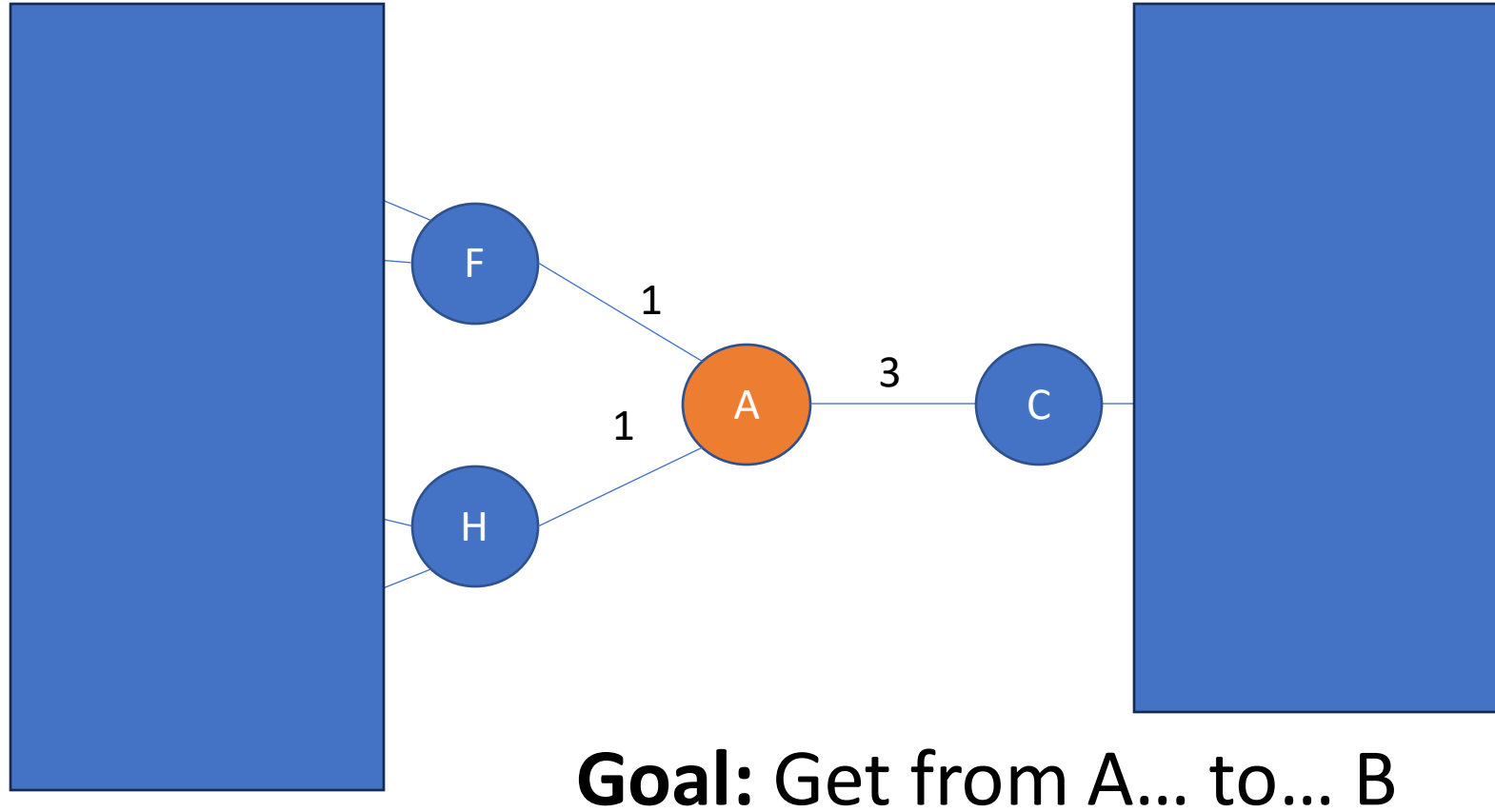
Artificial Intelligence

Lecture 03: Informed Search
(AIMA C3.5)

Lecture Outline

- Best-first Search
 - Greedy Search
 - A* Search
 - Heuristics

When UCS goes wrong...



Best-First Search

Definition:

- A node is selected for expansion based on an **evaluation function**.

The idea:

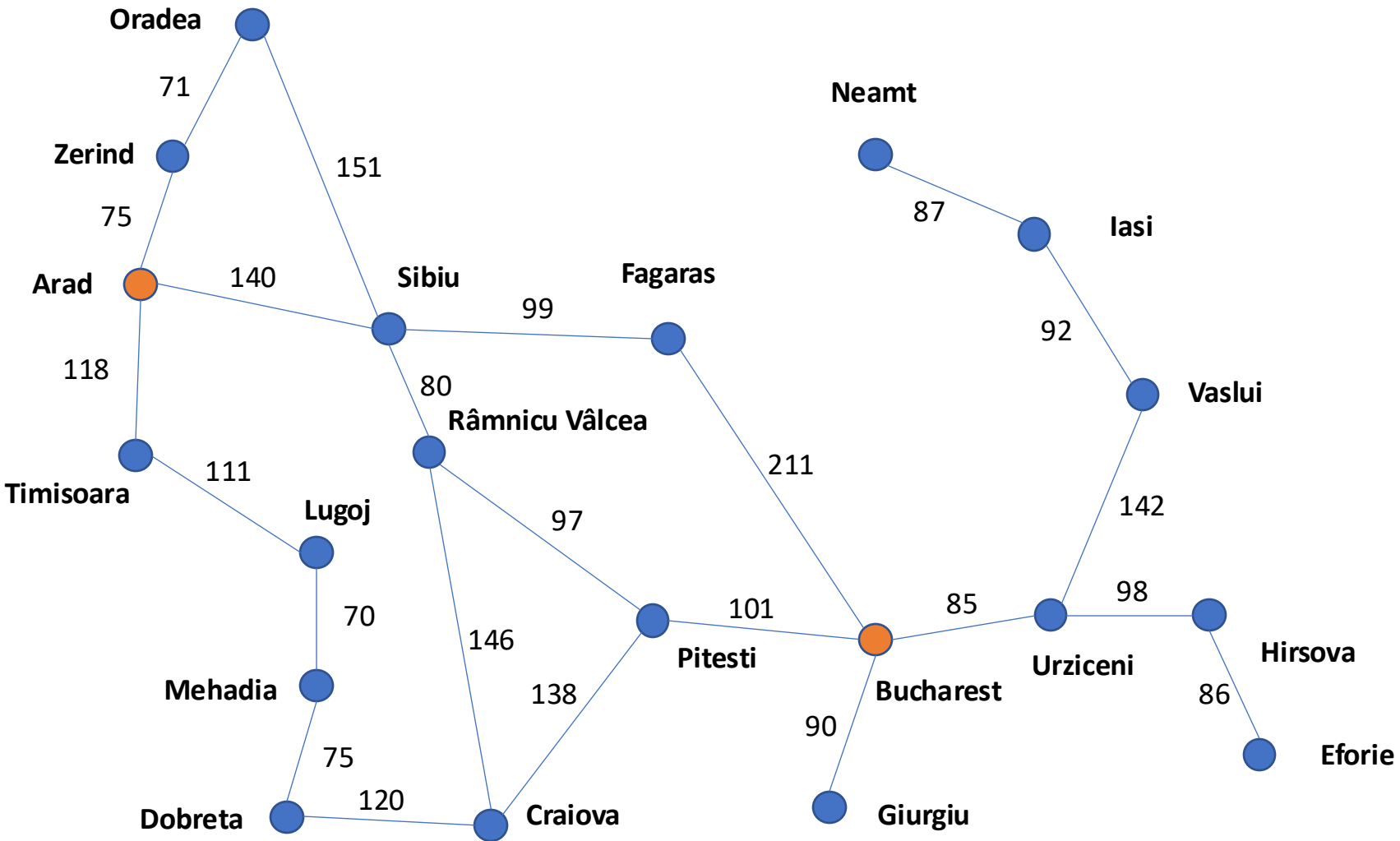
- What if we had an '**evaluation function**' which gave us an idea of the '**desirability**' of each node
- We then expand the most '**desirable**' node first

Implementation:

- The **fringe** is now sorted by **desirability**

The Problem!

How to get from...
Arad to Bucharest!
But...



Straightline to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Faragas	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy “Best-first” Search

Evaluation function $h(n)$

(h for heuristic)

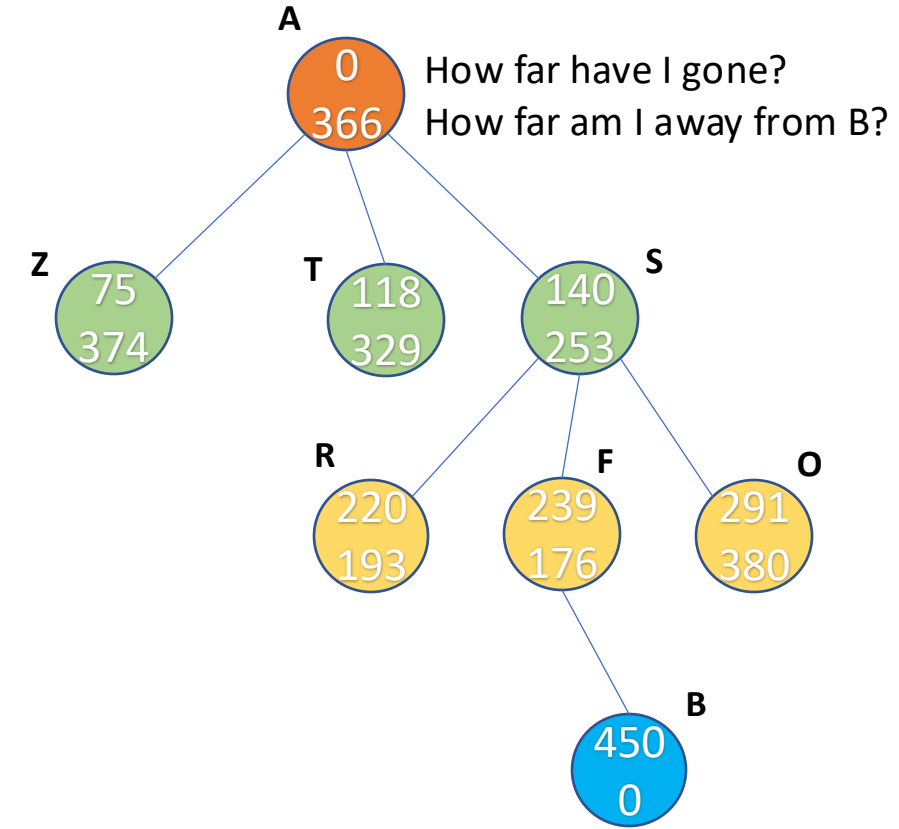
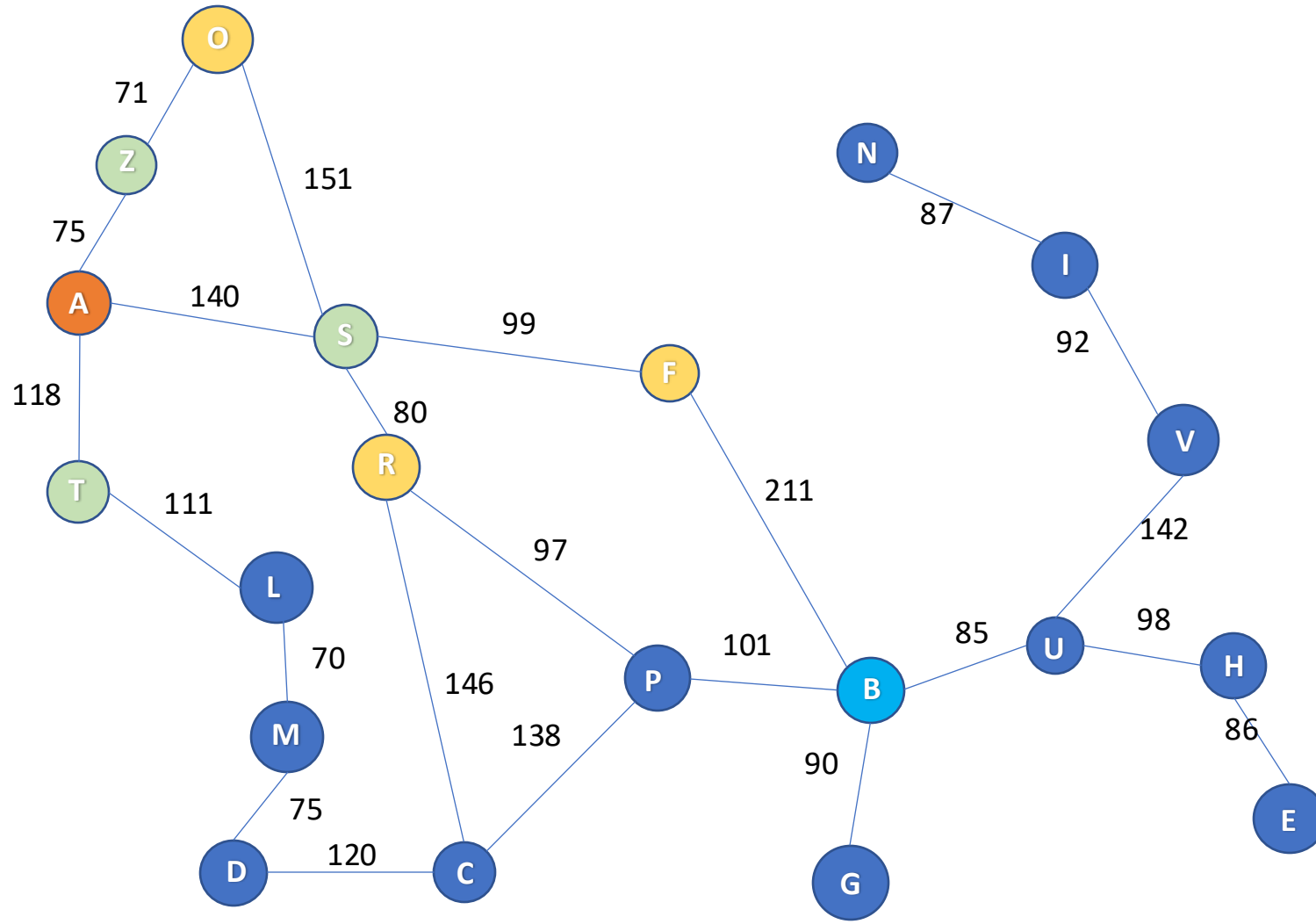
In our case, $h(n)$ is the straight-line distance to Bucharest

Greedy search expands the node that **appears** to be the closest to the goal

Side note: this use of the term **Greedy** isn't exactly the same as the usual use of a 'greedy algorithm'

Local optimal vs global optimal? Check “Stanford marshmallow experiment”

Greedy Search



Straightline to B

A	366
B	0
F	176
O	380
R	193
S	253
T	329
Z	374

Greedy Search Analysis

Complete?

- No (Yes if finite and state checks)

Time Complexity?

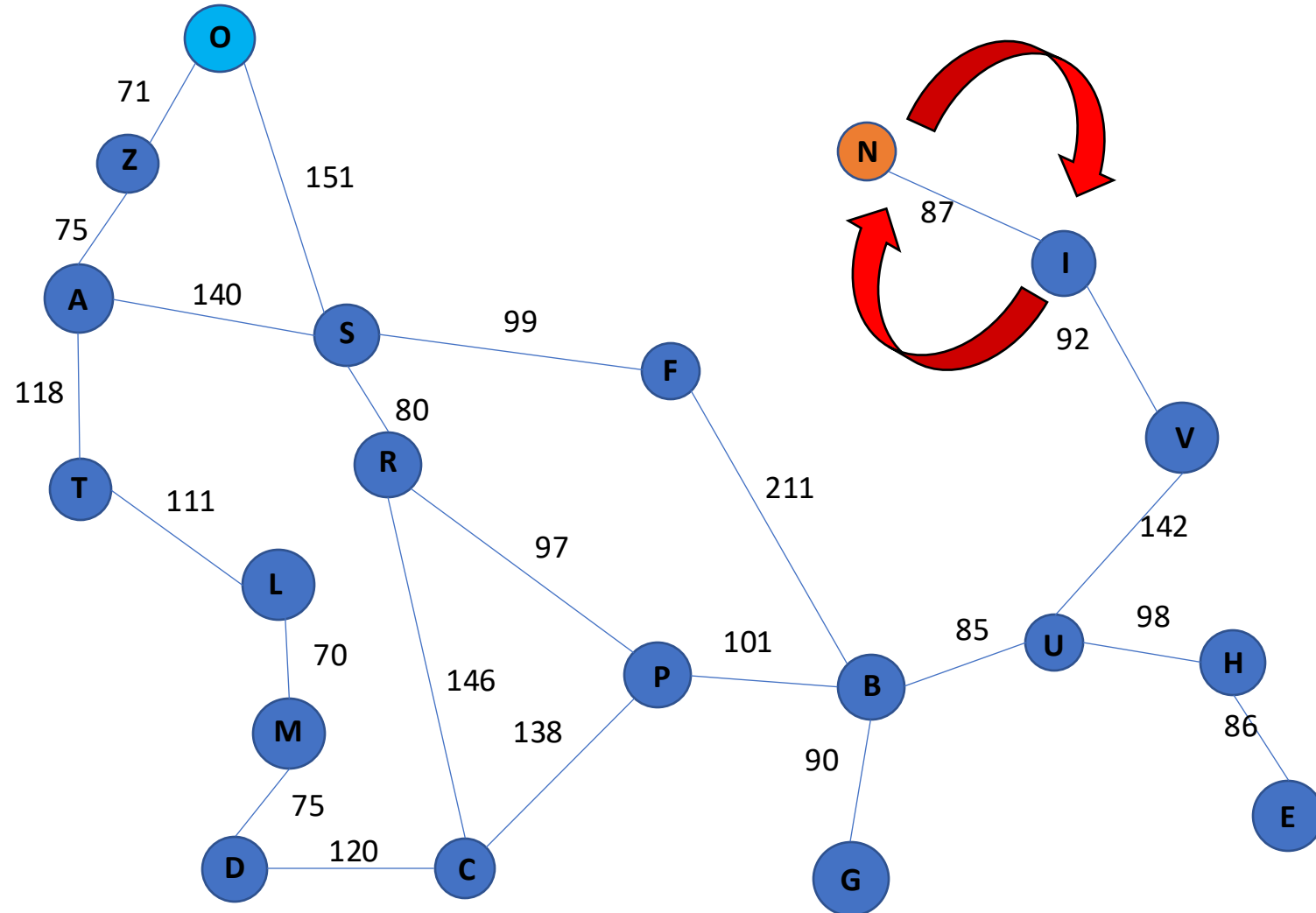
- $O(b^m)$
- Dependent on heuristic

Space Complexity?

- $O(b^m)$
- Keeps all nodes in memory

Optimal?

- No...



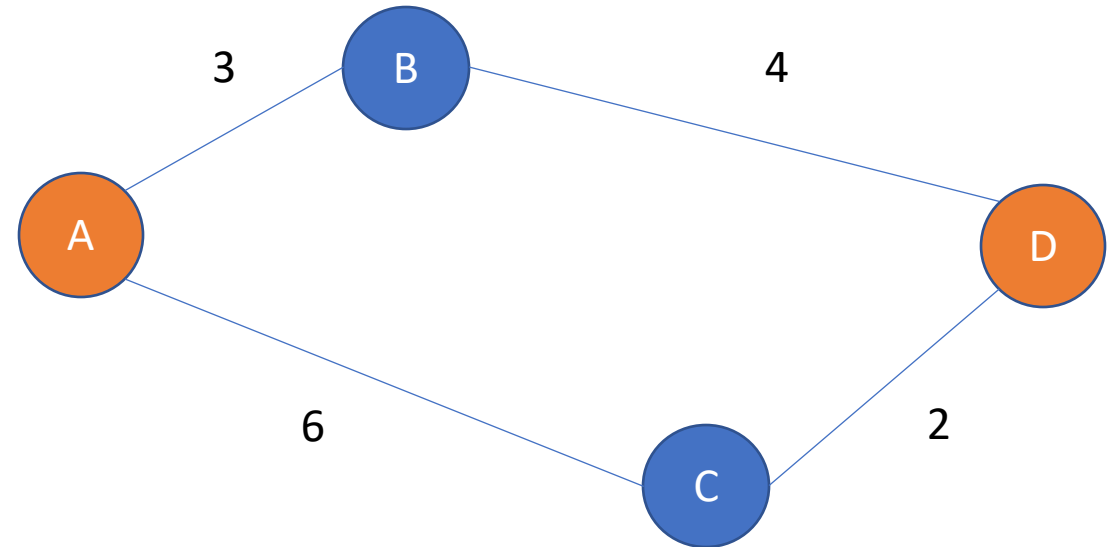
A* Search

Goal is $A \Rightarrow D$

Greedy Search would expand C before B... because C is closer to D

But... , C has already cost me 6 plus an extra 2 is 8

Not as good as B which is 3 plus an extra 4 is 7



A* Search

A subtle variation on greedy search

Two main components:

#1: What is the current path cost already?

#2: What is the estimate of the remaining path?

In economics & business

Retrospective (Sunk) cost

Perspective cost

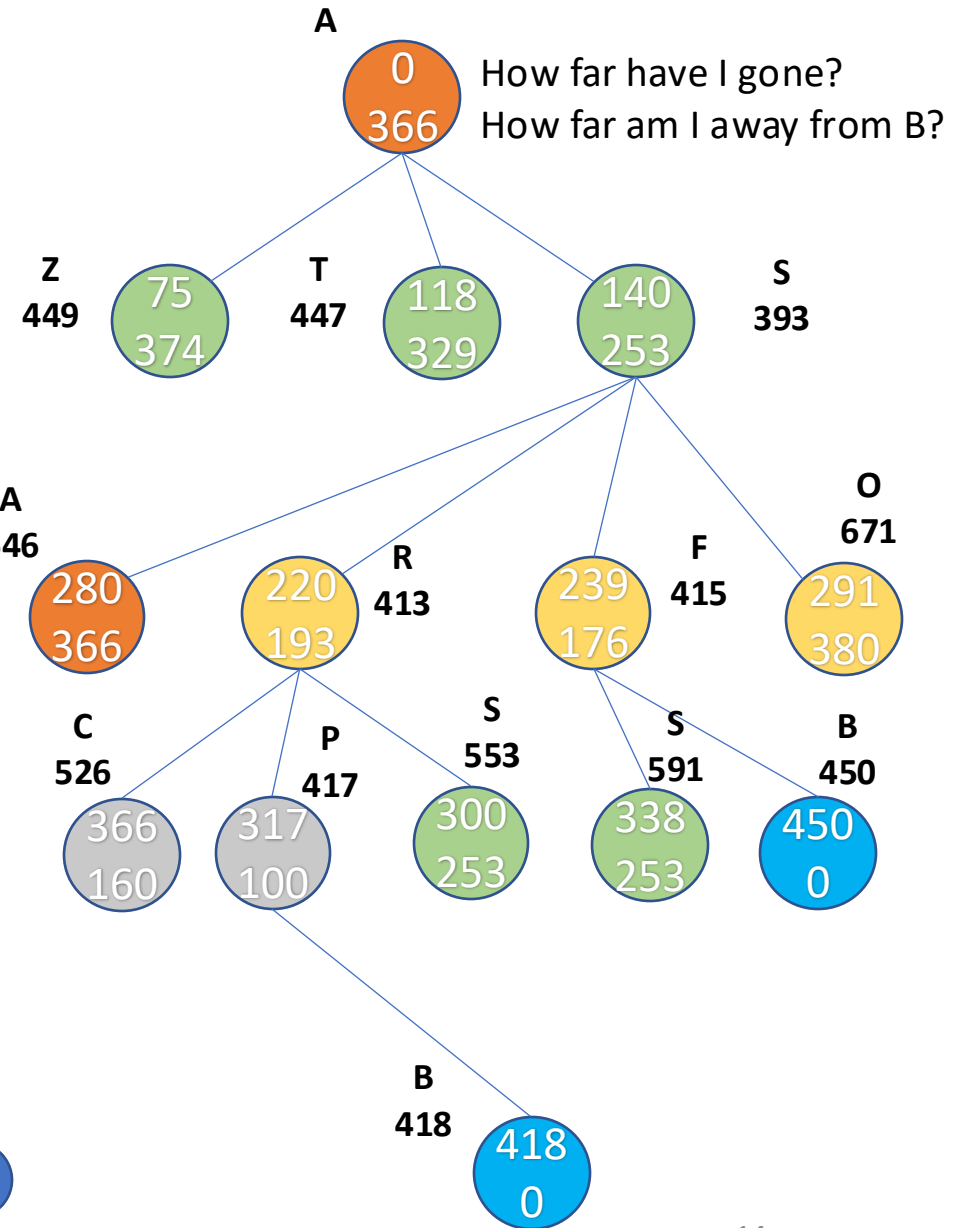
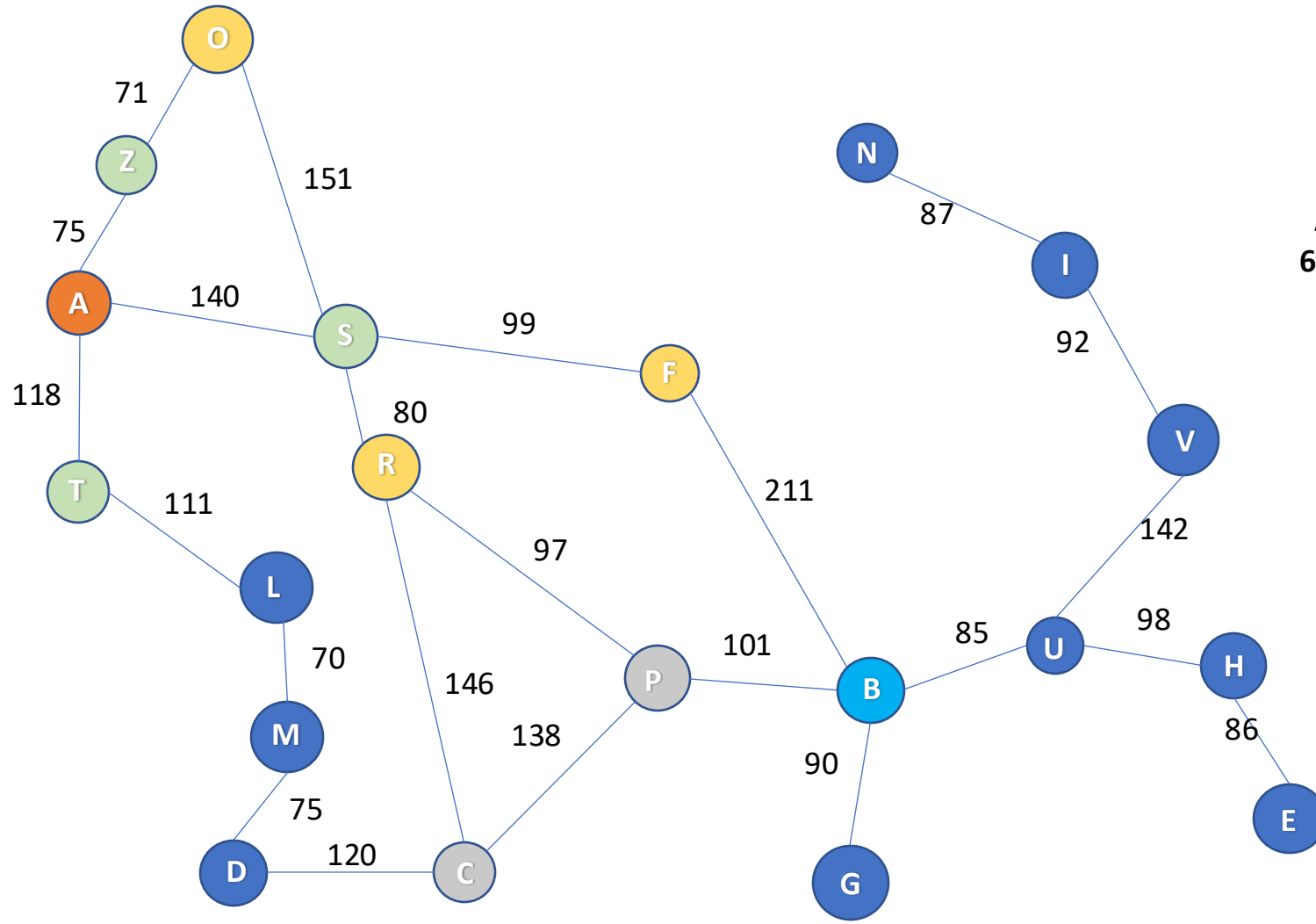
#1: $g(n)$, #2: $h(n)$

Evaluation Function

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

What if we only have $g(n)$?

A* Search



How does A* work?

A* only works if the heuristic function $h(n)$ underestimates the remaining distance. It means A* is an **admissible heuristic** search algorithm.

For our road-map version, the straight-line distance is always shorter than the actual road distance (can be equal).

i.e., $h(n) \leq h^*(n)$ ($h^*(n)$ is the true cost)

For example...

$h(G) = 0$ (G is the goal node)

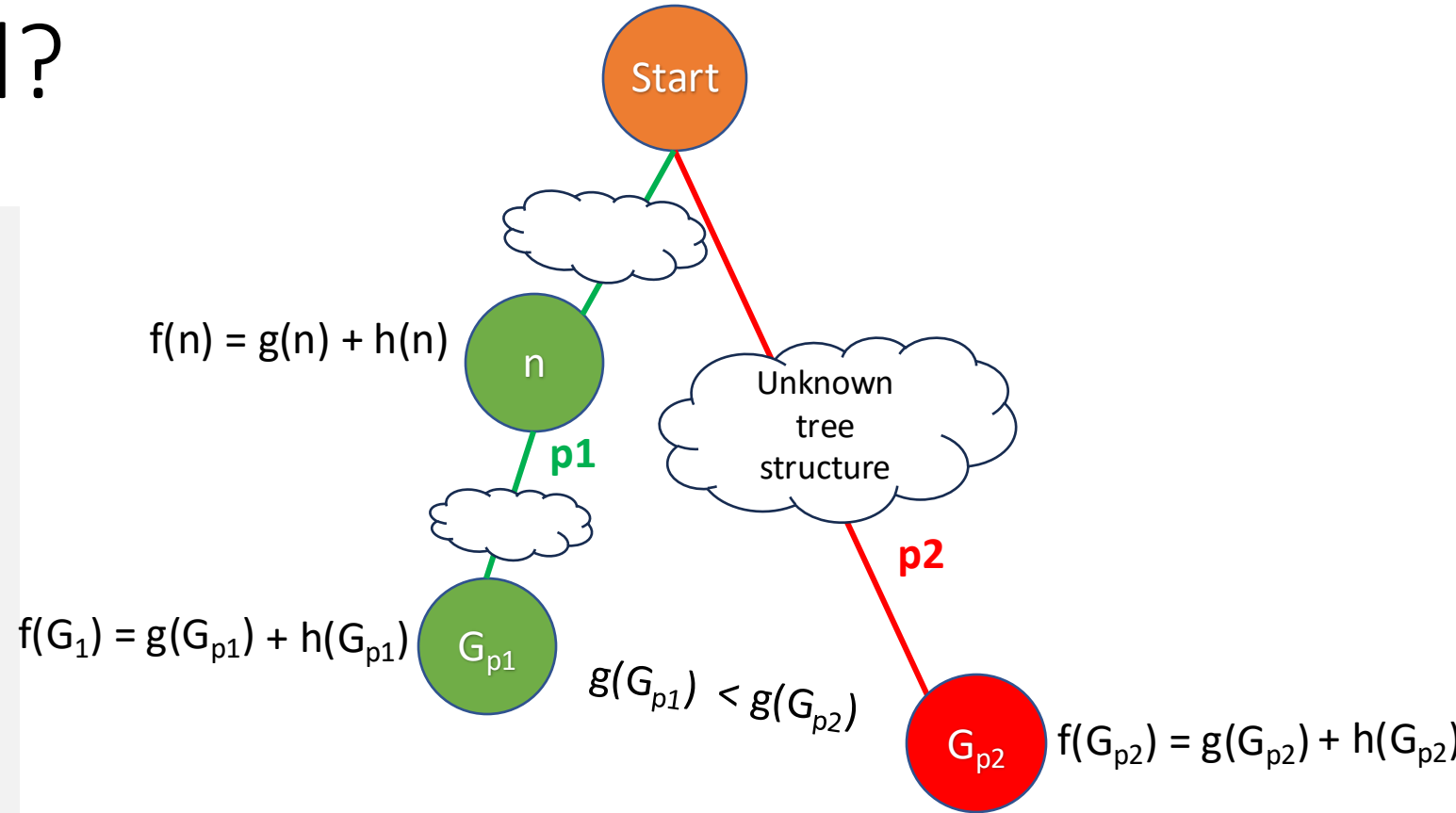
Is it really optimal?

The problem:

1. The optimal path is G_{p1}
2. If we have generated a bad (longer) path G_{p2}
3. Let us assume n is on the real shortest path

Is it possible to expand G_{p2} before n ?

Admissible means that the heuristic never overestimates the remaining distance to the target: $h(n) \leq g(G_{p1}) - g(n)$



$$\begin{array}{lcl}
 & f(n) & \text{vs} \quad f(G_{p2})? \\
 f(n) = g(n) + h(n) & \text{vs} & g(G_{p2}) + h(G_{p2}) = f(G_{p2}) \\
 \text{Since } h(G_{p2}) = 0, & g(n) + h(n) & \text{vs} \quad g(G_{p2}) \\
 & g(n) + h(n) \leq g(G_{p1}) < g(G_{p2}) \\
 f(n) = g(n) + h(n) & \leq & g(G_{p1}) < g(G_{p2}) = f(G_{p2})
 \end{array}$$

Analysing A*

Complete?

- Yes (unless there are infinite 'closer nodes')

Time Complexity

- $O((b^\epsilon)^d)$
 - b: branching factor
 - $\epsilon = (h^* - h) / h^*$: relative error
 - h^* : actual cost to the optimal solution
 - d: optimal solution depth

Space Complexity

- Keeps all nodes in memory

Optimal?

- Yes

Some further observations about h^*

What happens when $h(n)$ is always set to zero.

I.e., the estimate is always zero.

Is this admissible?

What does the algorithm look like?

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

What happens when $h(n) = h^*(n)$

I.e., the heuristic is magically correct?

Is this admissible?

What does this algorithm look like?

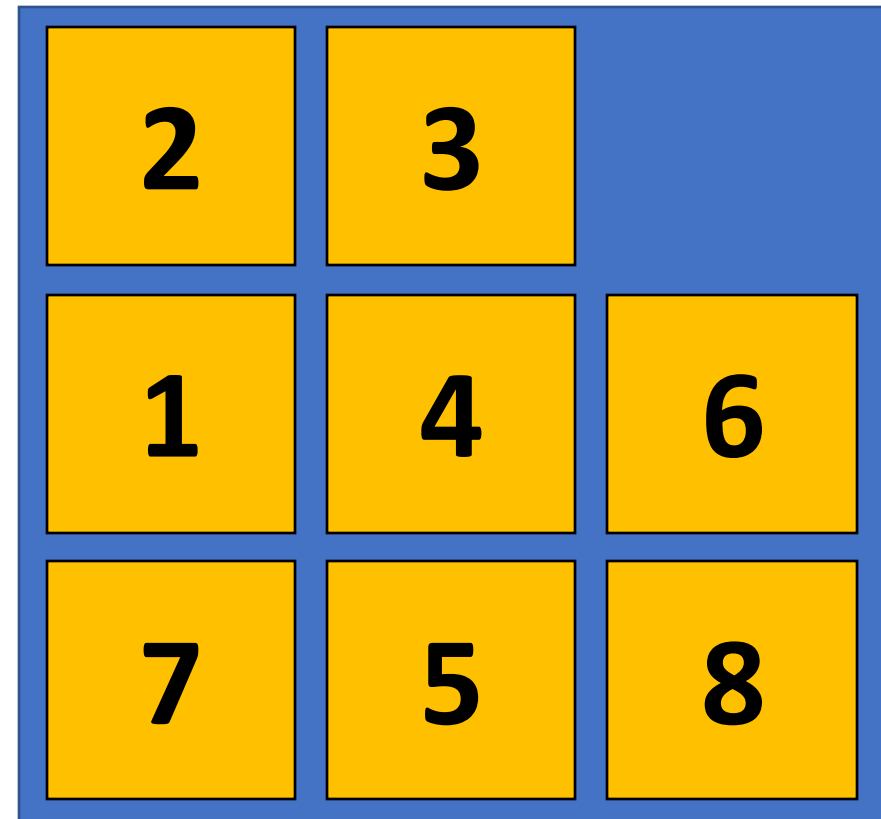
$$f(n) = g(n) + h^*(n)$$

Making an Admissible heuristic

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total **Manhattan** distance

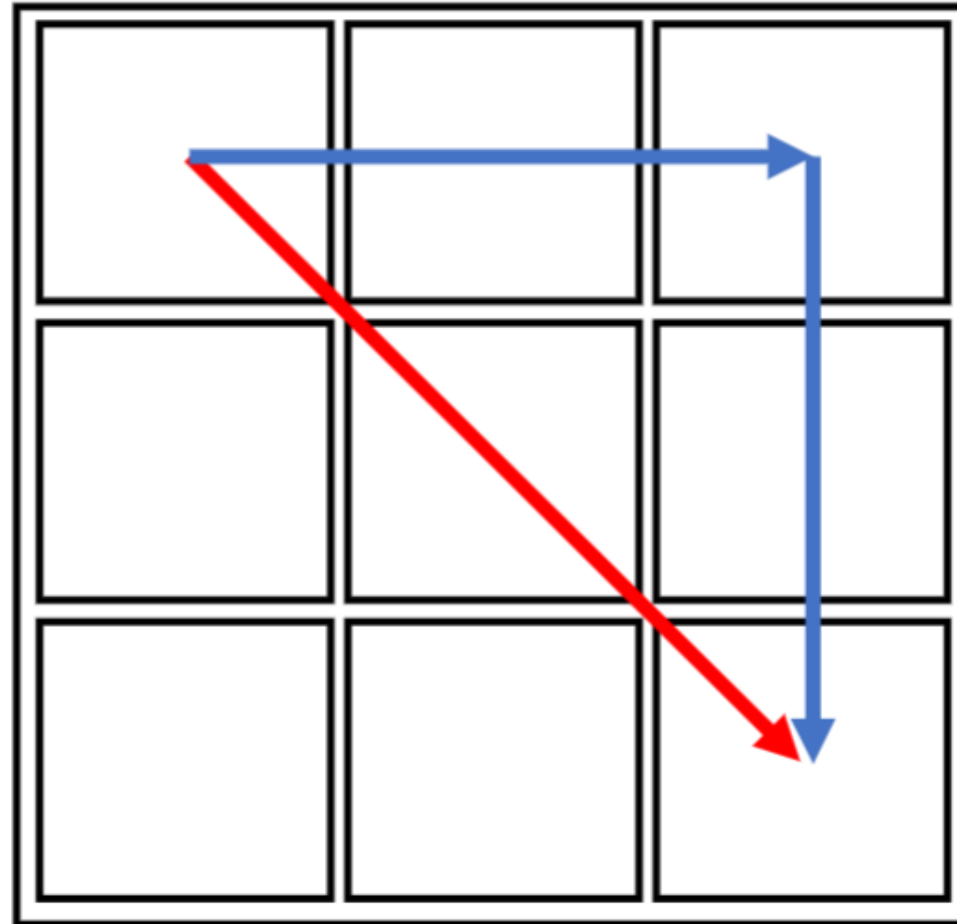
- (total number of squares from desired location no diagonal moves)



Manhattan Distance

Euclidean Distance
(straight line distance)
 $= \text{sqrt}(2) * 2$

Hamming distance
(number of misplaced tiles)
 $= 2$



Manhattan Distance (horizontal + vertical distance)
 $= 2 + 2$

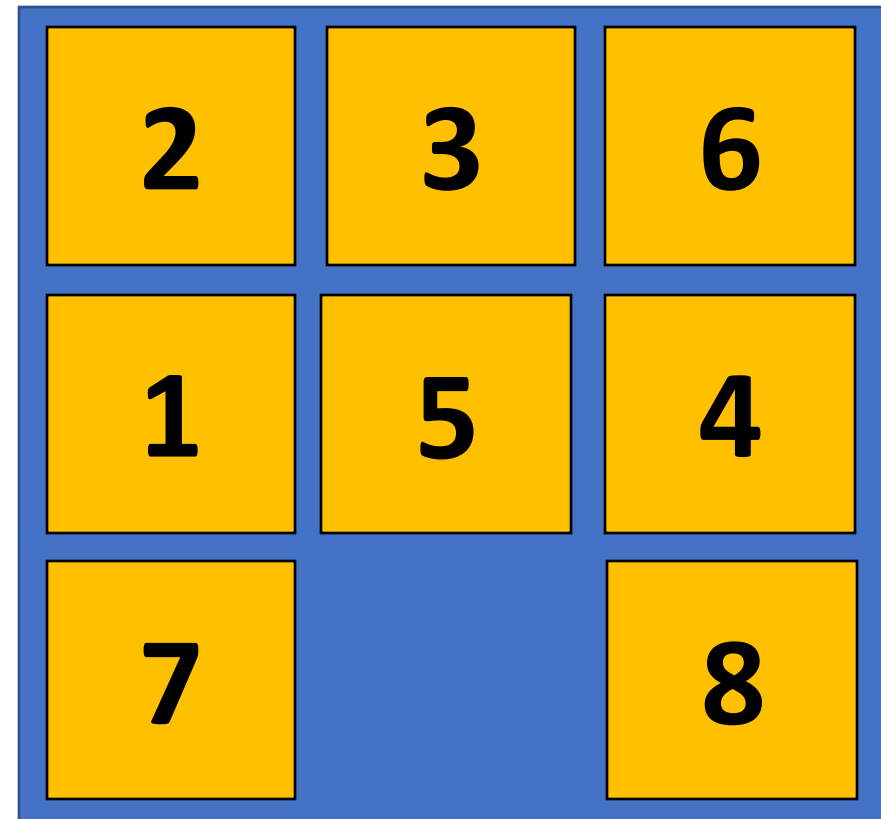
Making an Admissible heuristic

$h_1(n)$ = number of misplaced tiles
(hamming distance)

$h_2(n)$ = total **Manhattan** distance

-(total number of squares from desired location no diagonal moves)

	1	2	3	4	5	6	7	8	Total
$h_1(n)$	1	1	1	1	0	1	0	1	6
$h_2(n)$	1	1	1	2	0	1	0	1	7
$h^*(n)$					+2				9



Are these heuristics admissible?

	1	2	3	4	5	6	7	8	Total
$h_1(n)$	1	1	1	1	0	1	0	1	6
$h_2(n)$	1	1	1	2	0	1	0	1	7
$h^*(n)$					+2				9

Which is better?

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

Analysing A*

Complete?

- Yes (unless there are infinite 'closer nodes')

Time Complexity

- $O((b^\epsilon)^d)$
 - b : branching factor
 - $\epsilon = (h^* - h) / h^*$: relative error
 - h^* : actual cost to the optimal solution
 - d : optimal solution depth

Space Complexity

- Keeps all nodes in memory

Optimal?

- Yes

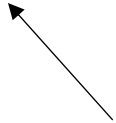
Intuitions about heuristics

What happens when your heuristic... is always zero?

What happens when your heuristic... is always 'correct' (i.e., magic/oracle)?

What happens when you don't take existing path into your calculation?

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



Your new BFF (until you forget)

Artificial Intelligence

Lecture 04: Constraint Satisfaction Problems
(AIMA C6)

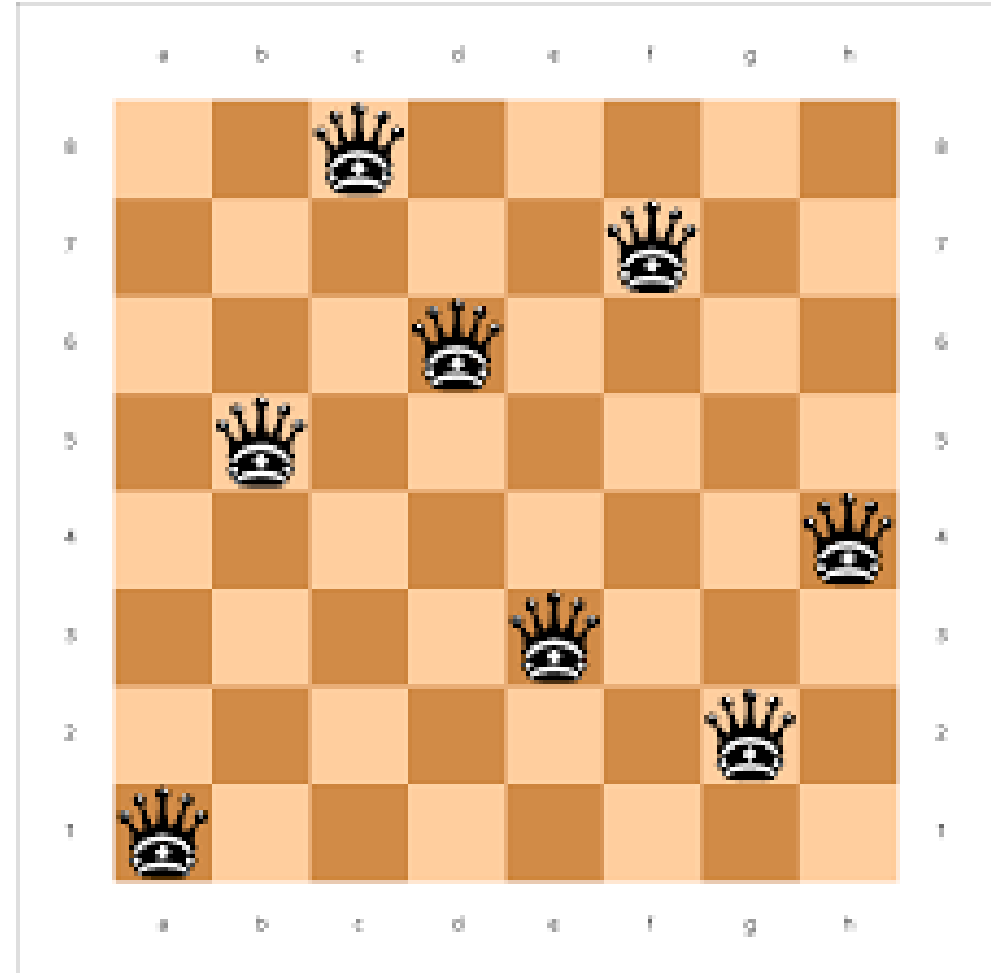
Lecture Summary

- Constraint Satisfaction Problem
- Backtracking Search (next lecture)

Constraints

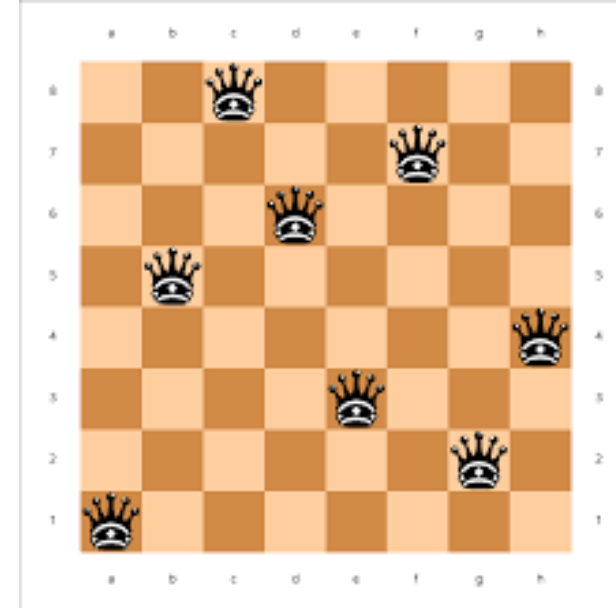
8 Queens

- Queens may not occupy the same row, column or diagonal
- How to place eight queens?



Constraint Problems

- If all constraints are satisfied, that is considered an optimal solution
- If multiple solutions exist, all correct solutions are considered equally valid



Map Colouring Problem

The Problem:

- There are regions present on a 2-dimensional surface.
- Each region may not share an adjacent edge with another region of the same colour.



Send More Money

	C_4	C_3	C_2	C_1
	S	E	N	D
+	M	O	R	E
<hr/>				
M	O	N	E	Y

Task: Make this 'sum' make sense
Each letter represents a unique number.

$$\begin{aligned}
 &1000S + 100E + 10N + 1D \\
 &+ 1000M + 100O + 10R + 1E \\
 &= 10000M + 1000O + 100N + 10E + 1Y
 \end{aligned}$$

variables domain

$$\forall S, E, N, D, M, O, R, Y \in \{0, 1, \dots, 9\}$$

constraints

$$\begin{aligned}
 1000S + 91E - 90N + 1D - 9000M - 900O + 10R - 1Y &= 0 \\
 S &\neq 0 \\
 M &\neq 0 \\
 M &= 1
 \end{aligned}$$

$$S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2$$

$$9000 + 91 \cdot 5 - 90 \cdot 6 + 7 - 9000 - 900 \cdot 0 + 10 \cdot 8 - 2 = 0!$$

Size of the naive search space: $O(d^n)$

Domain size: $d = 10$

Number of variables: $n = 8$

Varieties of Constraints

Unary:

- Involves a single variable
- WA \neq blue

Binary Constraints:

- Involves pairs of variables
- WA \neq QLD

Higher Order Constraints

- Involve 3 or more variables

Preferences:

- Sometimes a constraint isn't a hard constraint, rather an optimisation
- E.g., maybe **red** is better than **blue**?
- Can be represented as a cost

Some types of CSPs

Discrete variables

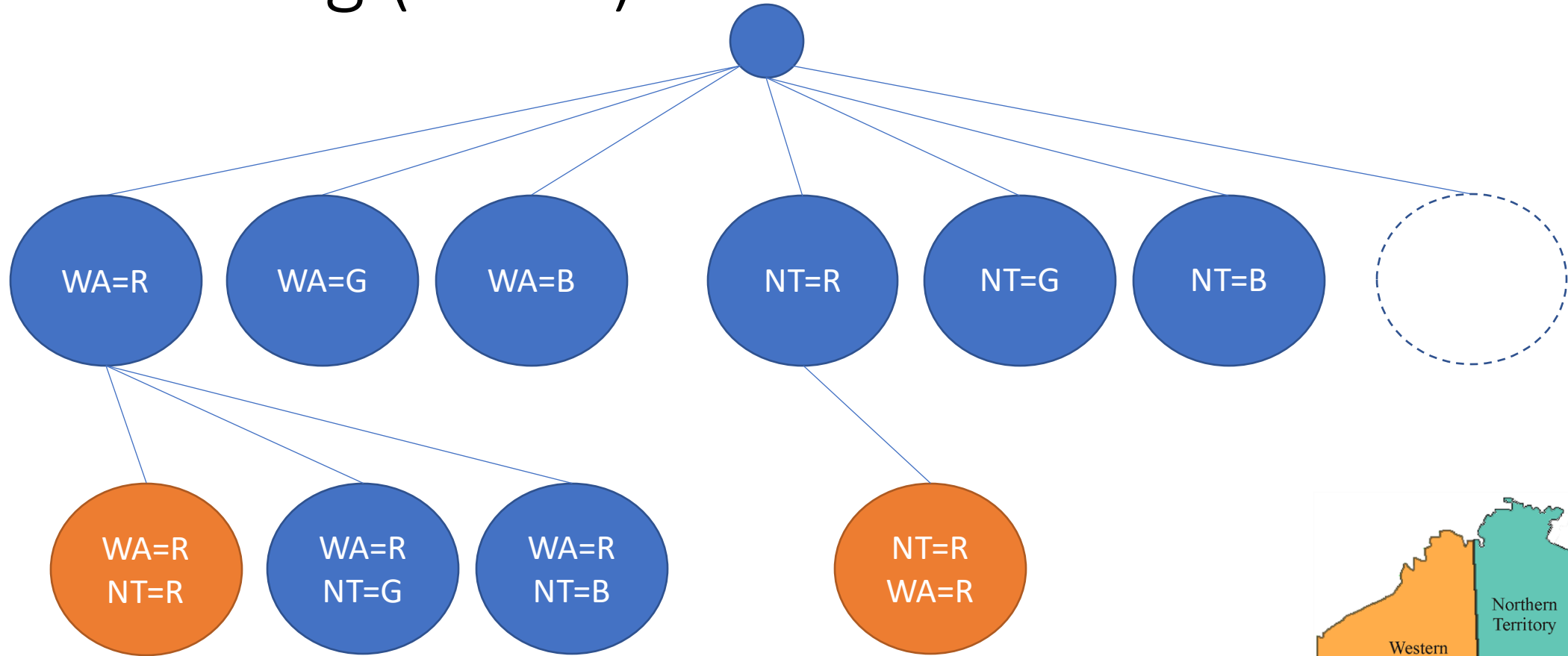
- Finite domains;
 - Size of the naive search space: $O(d^n)$
 - Boolean CSPs, Boolean satisfiability (NP-Complete problem)
- Infinite domains; (strings, integers)
 - Job scheduling (variables are the start/end days for each job)
 - Need a **constraint language** ($\text{startJob}_1 + 5 < \text{startJob}_3$ etc)

Continuous variables

- Start end times for Hubble Telescope observations
- Linear constraints are solvable in polynomial time via Linear Programming methods

Find a vector \mathbf{x}
that maximizes $\mathbf{c}^T \mathbf{x}$
subject to $A\mathbf{x} \leq \mathbf{b}$
and $\mathbf{x} \geq \mathbf{0}$.

Searching (Naïve)



Searching (Naïve)

The logic:

Initial: $\{\}$ (empty)

Successor: Add an unassigned value, check it does not conflict

Goal: Are we done?

Some issues:

- This approach suffers from repetition
- The branching factor is large ($7 * 3$)
- In general $(n - 1) * d$ at depth = 1
- Leaves = $n!d^n$ (oh dear)

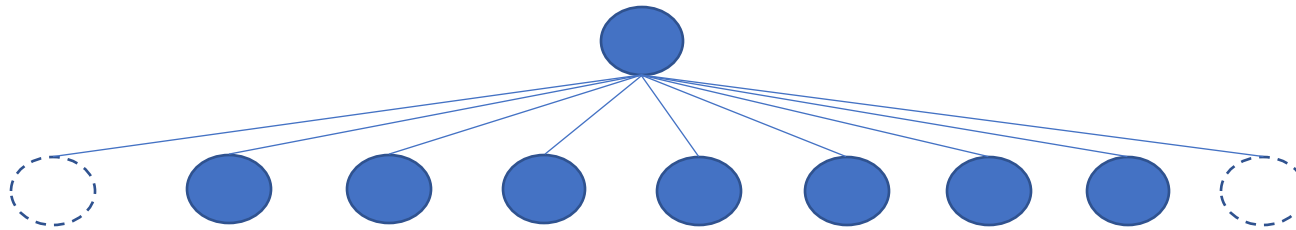
$7*3$

$6*3$

...

$1*3$

All solutions appear at depth n with n variables



Domain size: $d = 3$

Number of variables: $n = 7$

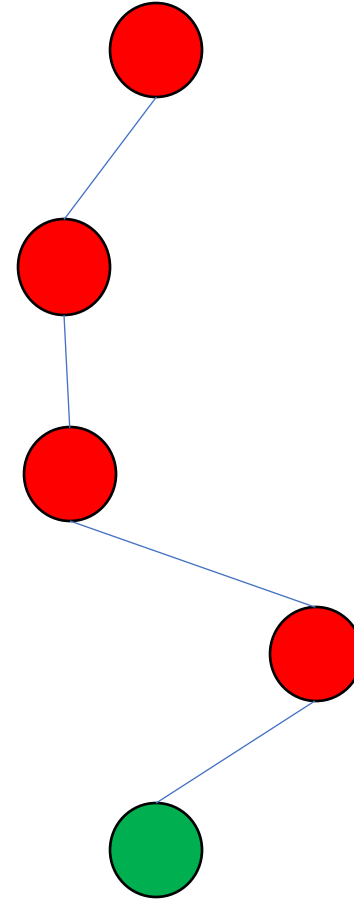
Which search would we start with?

BFS or DFS?

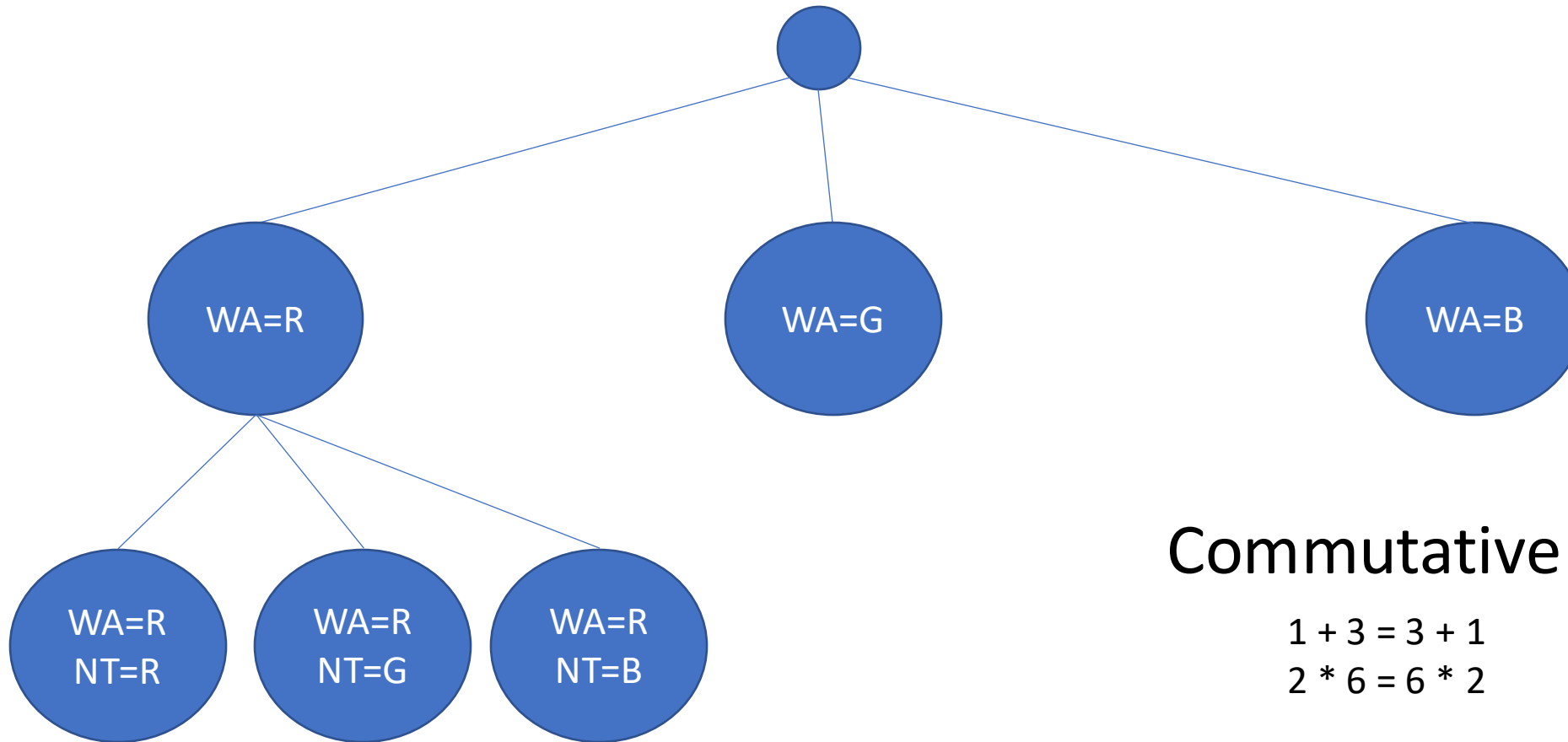
- DFS

Why?

- None of the interim states are solutions



Searching (Optimising)



WA = R, NT = B is the same as **NT = B, WA = R**

Commutative!!

$$1 + 3 = 3 + 1$$

$$2 * 6 = 6 * 2$$

$$2 ^ 6 \neq 6 ^ 2$$

Searching (Optimising)

Approach #1

Pick a random state and expand from there

- BF is $7 \times 3 = 21$.

Approach #2

Pick a specific state, knowing that all states will eventually be picked.

- BF goes from 21 to 3.

Searching with Backtracking

Backtracking

- In these examples we go forward... until a constraint is violated
- Then we back track to a new branch and then continue...

5	3	1	2	7	6	8	9	4
6	2	4	1	9	5	2		
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
			8				7	9

