Introduction to Artificial Intelligence & Data Science

Exercise 1

DI Dr Henrik Schulz

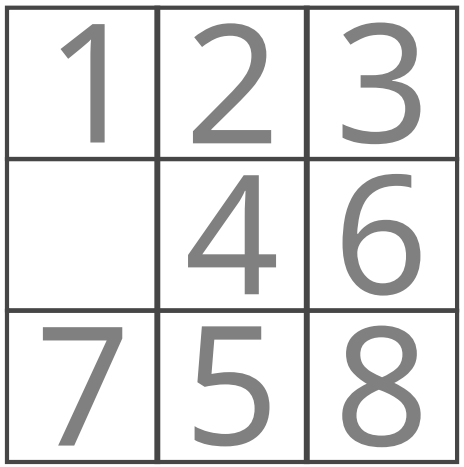
University of Applied Sciences
Computer Science and Digital Communication

November 25, 2024

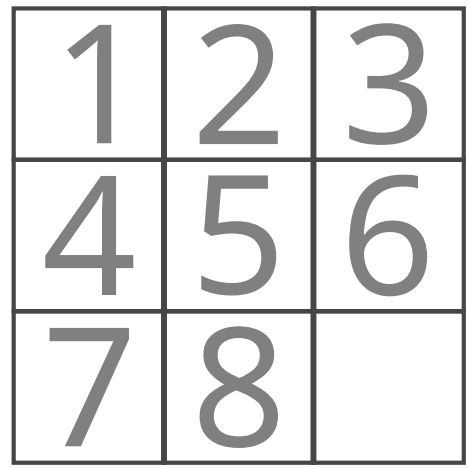
Planung der Übungstermine

ILV	Modus	UE	Thema	Aufgabe/ Datensatz	Abgabe- termin	
1	Online	2	A-Search	8-Puzzle	Einführung	
	Online	2	A-Search	8-Puzzle	Praktische Übung unter Anleitung	1.12.2024
2		2	A-Search	8-Puzzle	Gruppen- präsentation	
		2	ML (RandomForest)/ DL (CNN)	CIFAR	Einführung	8.12.2024
3		2	CNN Optimierung	MNIST	Einführung	
		2	CNN Optimierung	MNIST	Praktische Übung unter Anleitung	31.12.2024
4	Online	2	MLP Backpropagation			
	Online	2	MLP XOR		Einführung	
5		2	CNN Optimierung	MNIST	Gruppen- präsentation	
		2	MLP XOR	XOR	Gruppen- präsentation	

A*-search example - 8-puzzle



start state



goal state

Exercise: A*-search example - 8-puzzle

- from random state generate goal of ordered puzzle elements
- use 2 different heuristics
- provide algorithm complexity
- implementation in Python or Java
- measure memory effort (number of tree nodes) and runtime for 100 random start states for each heuristic
- estimate mean and variance of these measures

Exercise: A*-search example - 8-puzzle

1. check for solvability
2. implement two heuristics: Hamming and Manhattan
3. compare the two heuristics (using 100 random searches each):
   - memory usage
   - computational costs
4. comment programming code
   - inputs and output variables
   - classes, member functions and variables
   - computational costs
5. don't just copy/paste code from the internet
   - provide your own design class/data structures
   - comment the programming source code

Exercise: A*-search example - 8-puzzle

- identify the elements - data structure and its algorithms - of the overall algorithms
- implement the data structures with member functions and variables, test the implementation, comment

Exercise: 8-puzzle task documentation

1. short task description
2. software architecture diagram
3. short descriptions of modules, classes and interfaces, variables
4. explain fundamental design decisions
5. discussion and conclusions - describe ...
   - experimental observations and experience
   - provide table with complexity comparisons of different heuristics
   - alternatives and improvements

Exercise schedule: 8-puzzle task

1. documented source code (.py, jupyter notebook)
2. documentation
3. submission until 1/12/2024
4. presentation in groups of 3-5 next exercise

Uniform cost-search

uses the cost of the path to a node to determine its desirability (Dijkstra graph search)

$g(n) =$ cost of path from start node to current node

only determines costs how expensive the path was to get to the final node
$\rightarrow$ insufficient if search aims at achieving the goal with minimal cost

estimate of the cost of reaching the goal from here desirable

Greedy search

uses the estimated cost from a node to the goal to determine its desirability

heuristic function estimates distance remaining to a goal

$h(n)$ = estimated cost of path from node to the goal

heuristic - technique to improve average-case performance

good heuristic function for the route-finding problem: straight-line distance to the goal

A*-search

combines uniform-cost search and greedy search by summing their evaluations

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

$f(n)$ =actual distance so far + estimated distance remaining

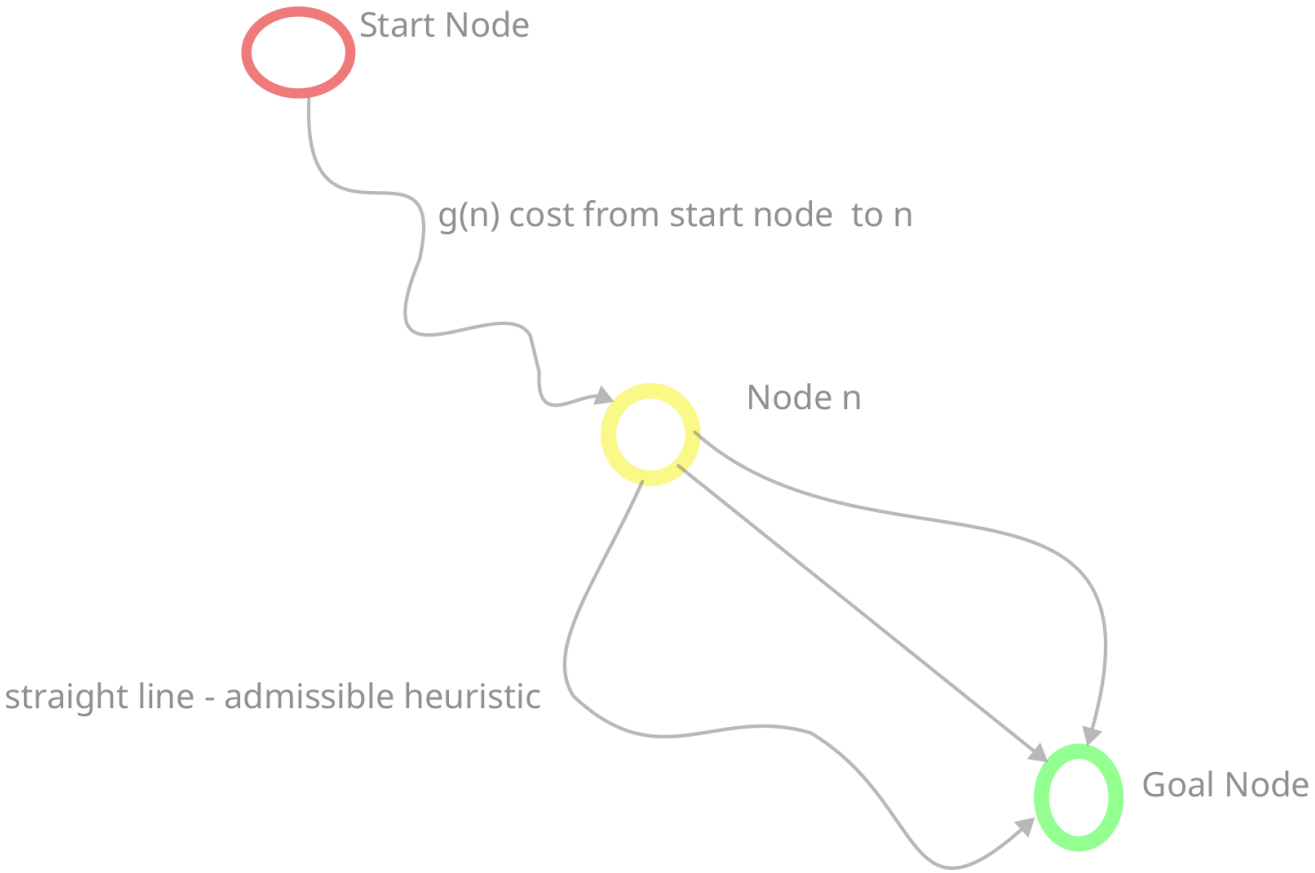
Admissible Heuristics

admissible heuristic - never overestimates the cost to reach the goal

optimistic inherently $\rightarrow$ heuristics assume cost of a solution is less than it actually is

$\rightarrow$ A* h(n) admissible also $f(n) = g(n) + h(n)$ never overestimates the true costs of a solution along the current path through n

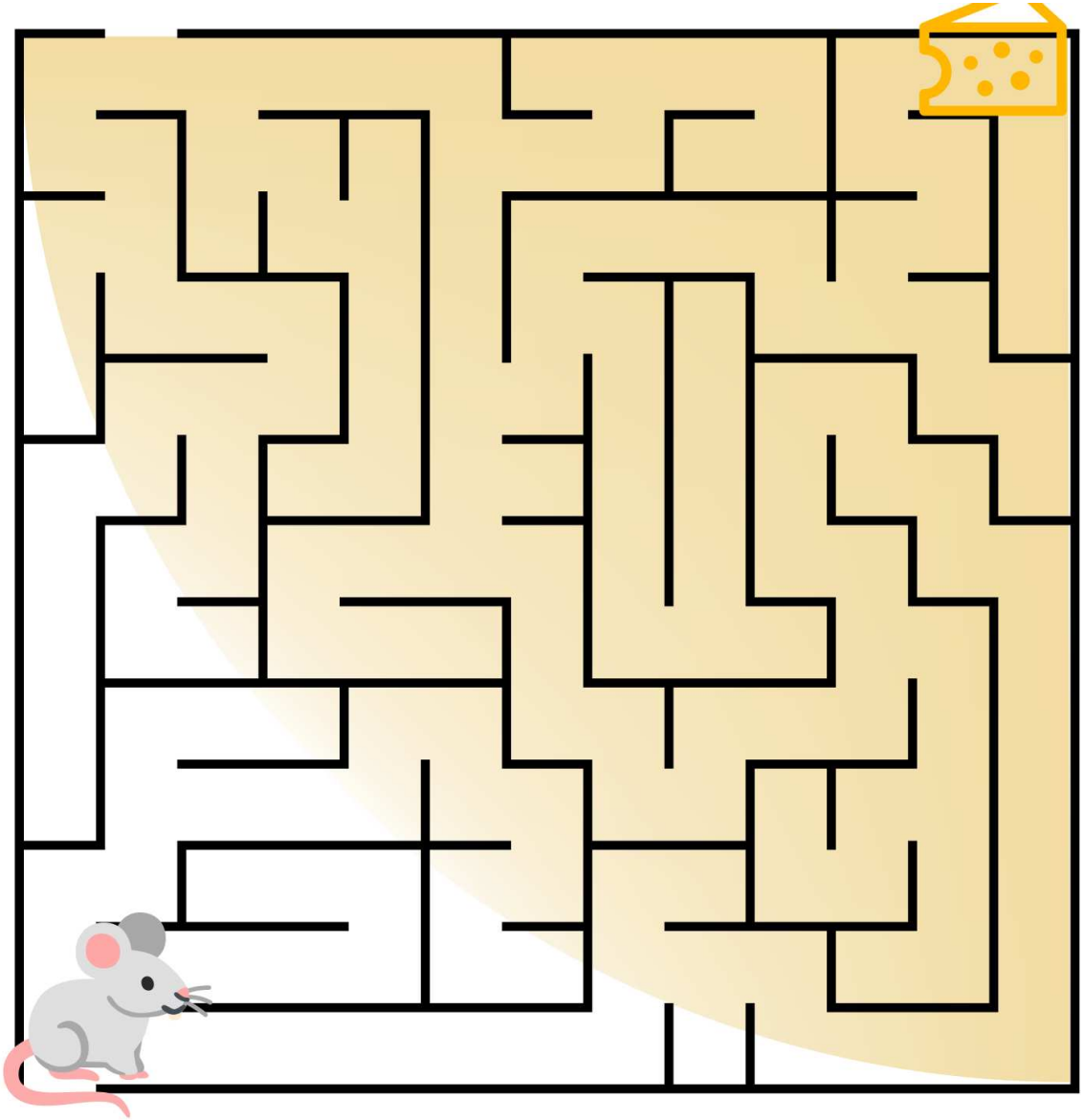
Admissible Heuristics



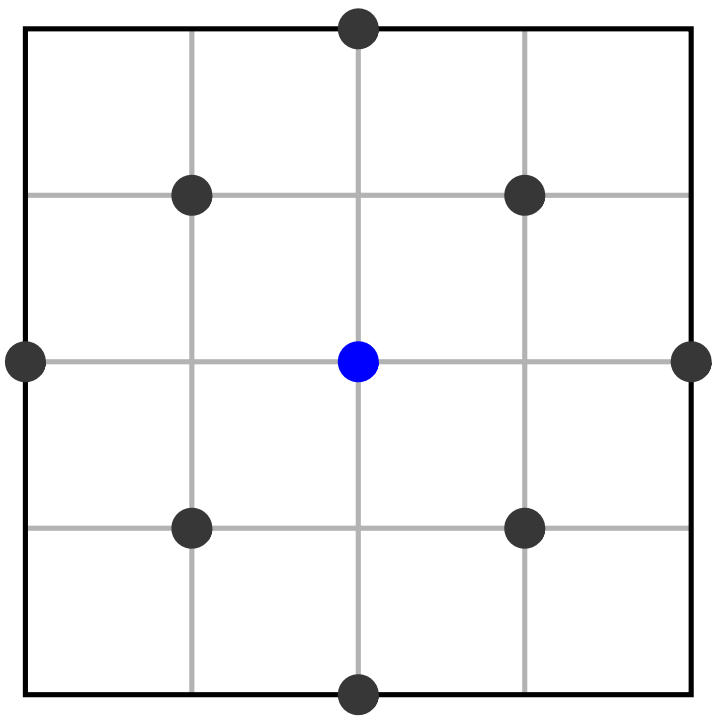
straight line distance is admissible $\Leftrightarrow$ shortest path between two points is straight line

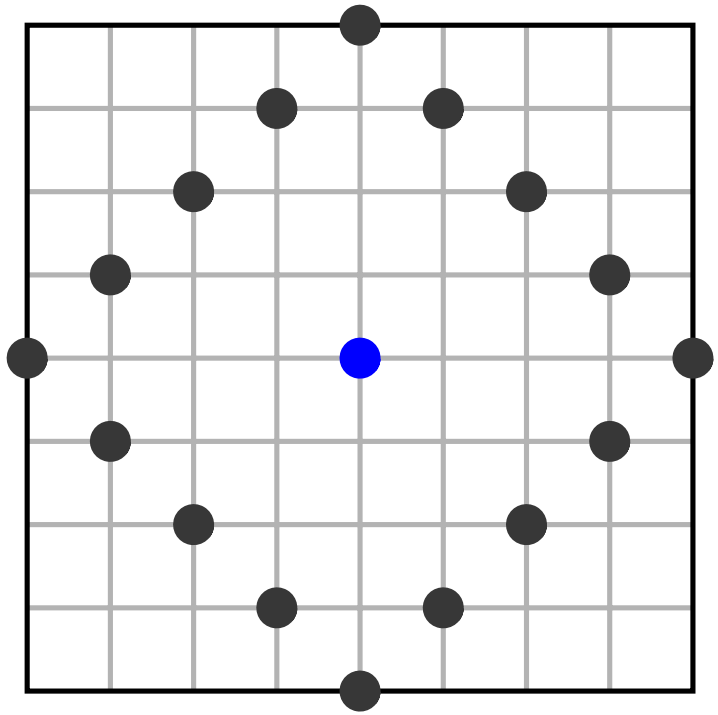
$\rightarrow$ straight line cannot be overestimated

Heuristics



Manhattan distance



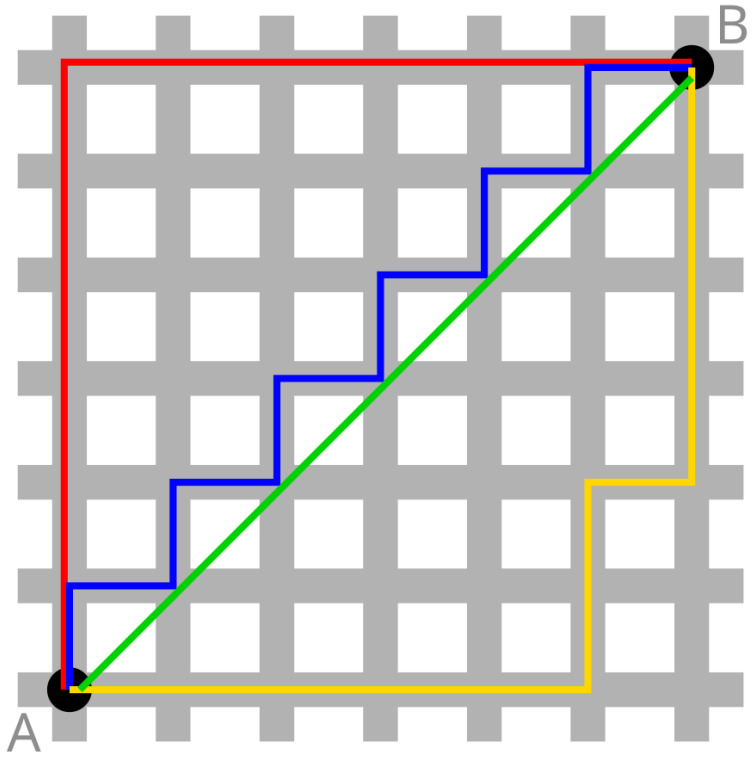


Reference: Manhattan Distance

Manhattan distance

Manhattan/city/taxi distance/metric

$$d(A, B) = \sum_i |A_i - B_i|$$



blue, red, yellow Manhattan distance, green Euclidean distance between two points A,B

admissible heuristic - since in every move, one tile can only move closer to its goal by one step

Manhattan distance

7	2	4
5		6
8	3	1

start state

1	2	3
4	5	6
7	8	

goal state

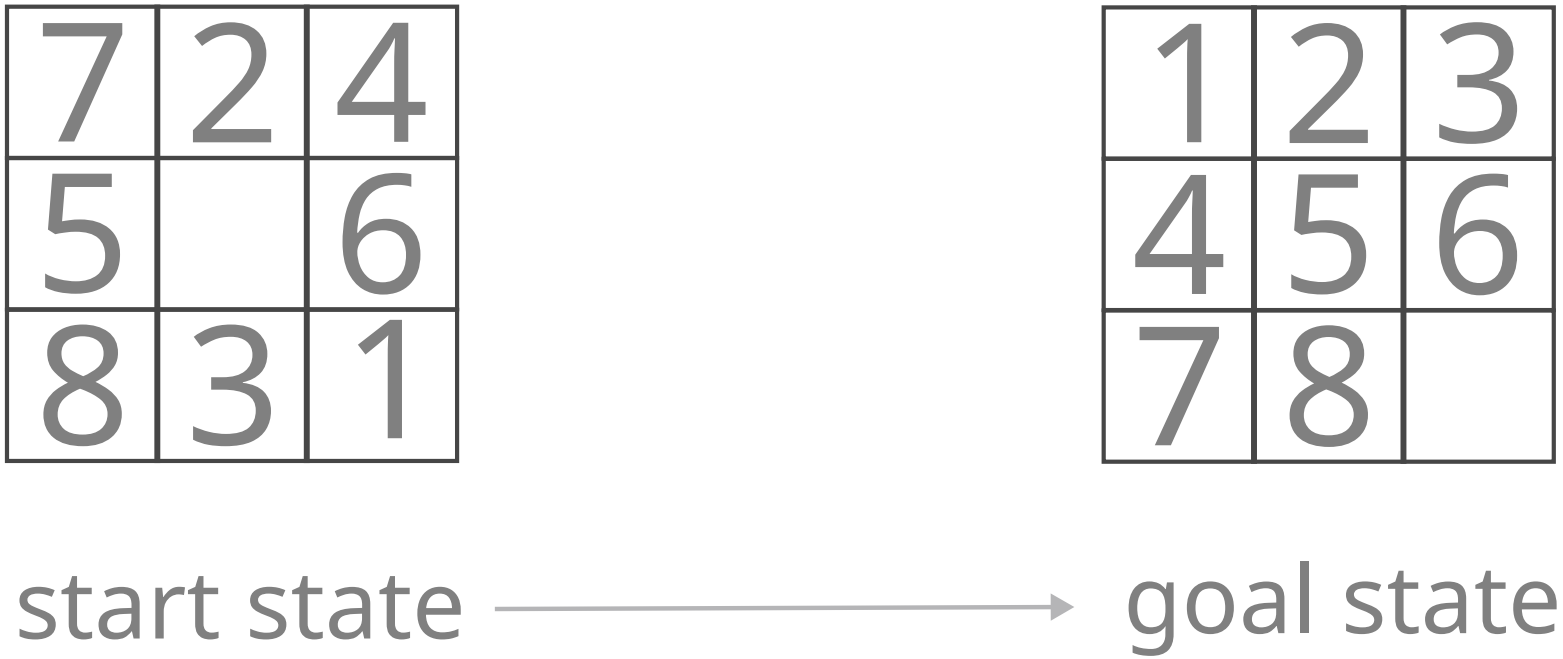
Sum of Manhattan distances of the tiles from their goal positions

$h = 2 + 0 + 3 + 1 + 0 + 1 + 3 + 4$

admissible heuristic function - total cost - number of moves required to reach the goal state will be greater than or equal to the Manhattan distance because any move will only take one tile - one step - closer to the goal

Hamming distance

Hamming distance/misplaced tiles

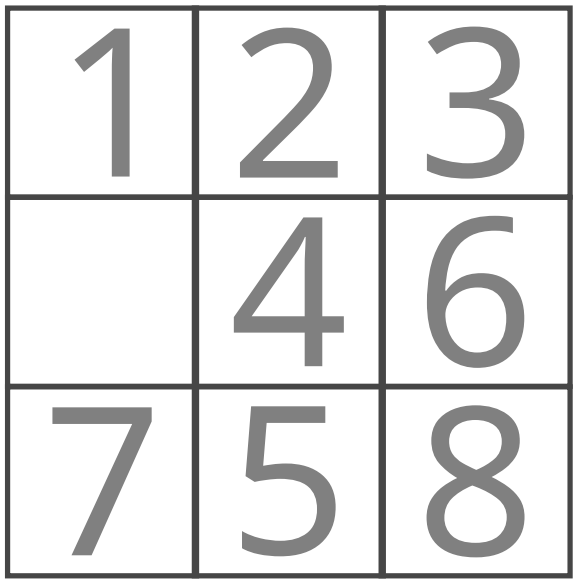


Above example h = 6

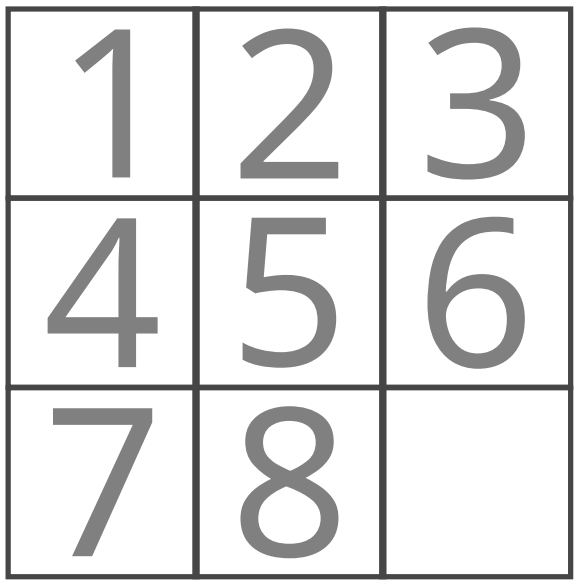
Admissible heuristics criterion holds
→ number of steps required to reach the goal definitely higher or equal to the number of misplaced tiles

each misplaced tile needs to be moved at least once

A*-search example - 8-puzzle

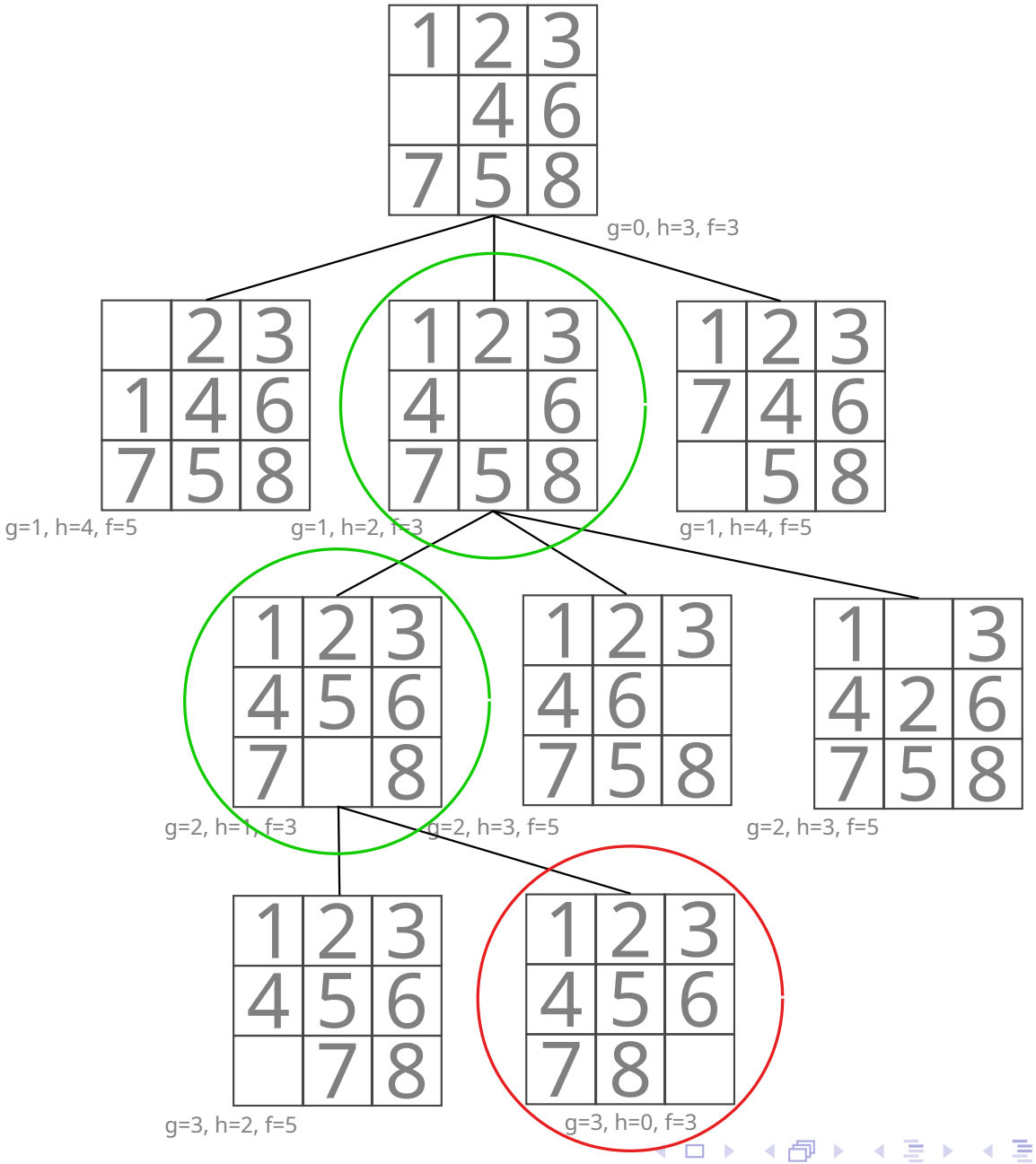


start state



goal state

A*-search example - 8-puzzle



A*-search example - 8-puzzle

Apart from opening up the search space as a tree:

What are other essentials to consider to make the algorithm work ?

A*-search example - 8-puzzle

Apart from opening up the search space as a tree:

What are other essentials to consider to make the algorithm work ?

1. open list - the states yet to be explored (priority queue of states, each state having an associated cost f)
2. closed list - the states already been fully explored
3. parent mapping - keep track of parent node to reconstruct the solution once the goal is found

Exercise Lets start: 8-puzzle task

1. recall the algorithm
2. note its fundamental concepts
3. derive data structures, methods and variables
4. formulate and illustrate a design
5. implement, test and document the functions, classes, members
6. run and document the experiments as discussed