Exercise 2

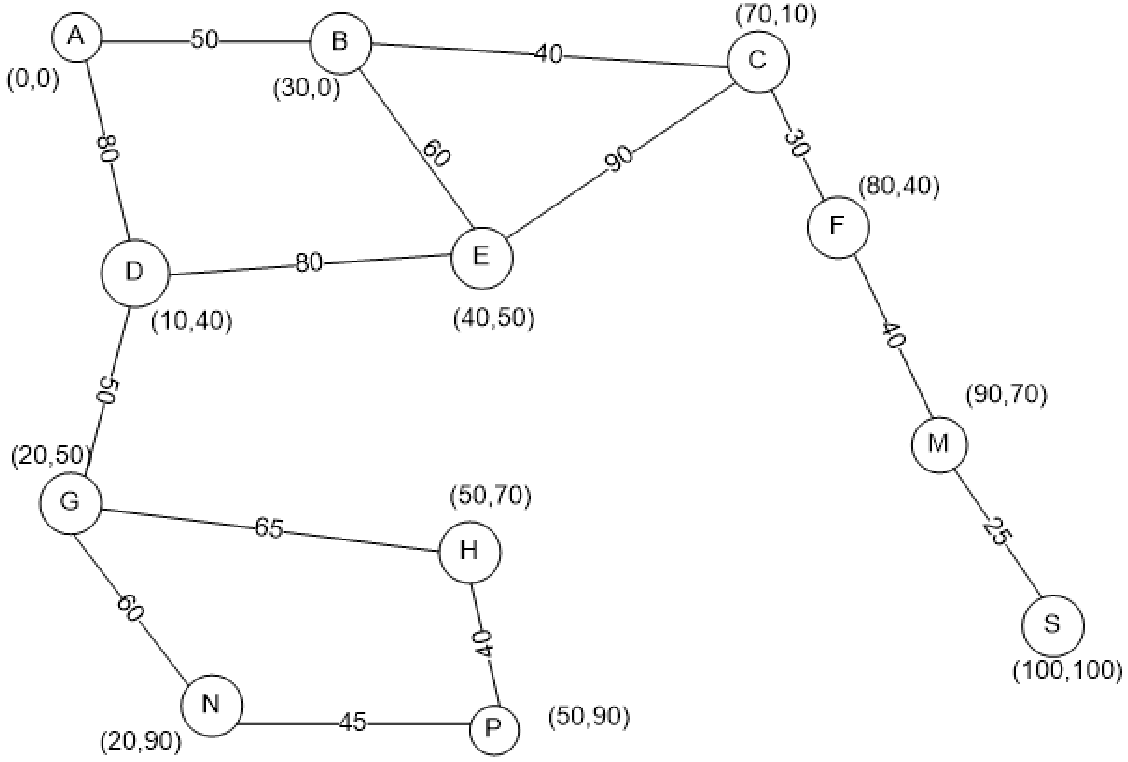
Exercise 2: Problem formulation and heuristics

Suppose two friends live in different cities.

In each turn: move both friends to an adjacent city (time = distance)

The friend who arrives first must wait until the other arrives before the next turn can begin.

We want the two friends to meet as soon as possible.



Exercise 2: Problem formulation and heuristics

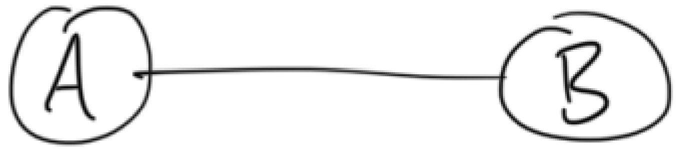
(a) Write a detailed formulation for this search problem

- State space: $\{(i,j) \quad \forall\ (i,j) \in V \times V\}$
- Successor function: $(i,j) \rightarrow \{(x,y) \quad with \quad (i,x) \in E,\ (j,y) \in E\}$
- Goal: $\{(i,i)\ \ \forall\ i \in V\}$
- Step cost function: $\max(\ d(i,x)\ ,\ d(j,y)\ )$

Exercise 2: Problem formulation and heuristics

(b) Are there any fully connected (not an island) maps for which there is no solution?

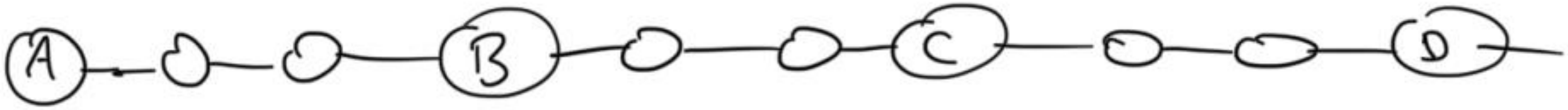
Yes, since both have to move in each turn



with (A,B) being the starting state

Exercise 2: Problem formulation and heuristics

or more generally: any chain with odd spacing



Starting states: (A,B), (A,D), (B,C), …

Exercise 2: Problem formulation and heuristics

(c) Let $D(i,j)$ be the aerial distance between city i and j. Which of the following heuristic functions are admissible?

- $D(i,j)$
- $2 * D(i,j)$
- $D(i,j) / 2$

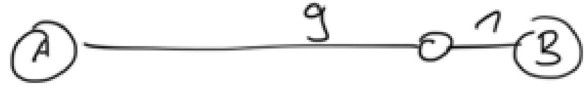
Reminder: a heuristic function is admissible iff:

$\forall n$: $h(n) \leq h^*(n)$ with $h^*(n)$ being the real costs

I.e.: A heuristic function shall <u>not overestimate</u> the costs

Exercise 2: Problem formulation and heuristics

(c) Let $D(i,j)$ be the aerial line between city i and j. Which of the following heuristic functions are admissible?

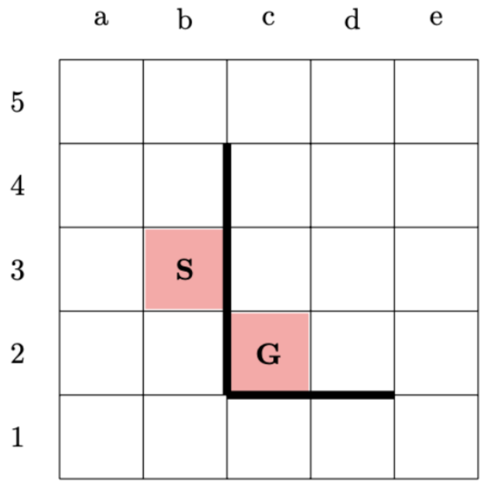
- $D(i,j)$ → not admissible. Counterexample: 

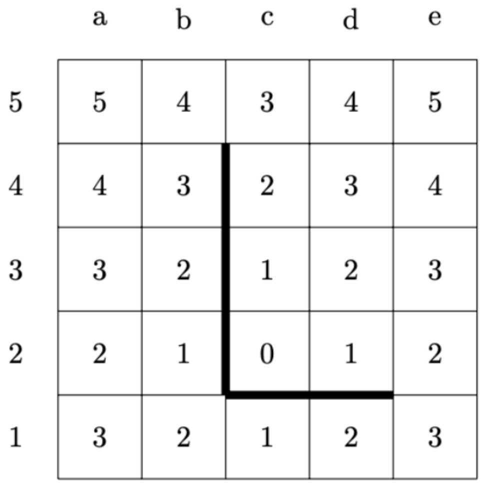
- $2 * D(i,j)$ → not admissible (even worse)
- $D(i,j) / 2$ → admissible (heuristic shall not overestimate)

Exercise 3

Exercise 3: Application of A*

A mobile robot wants to determine the shortest path from cell S (start) to cell G (goal). The robot can move horizontally and vertically, as long as it is not separated from the current cell by a wall (thick black line). Each movement has a uniform cost of 1. The left figure shows the initial state, the right figure the heuristic values.

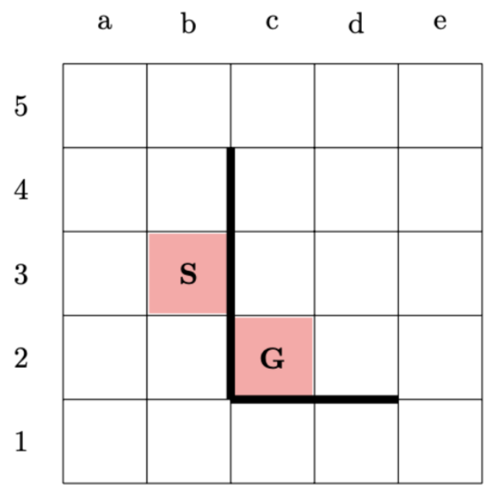


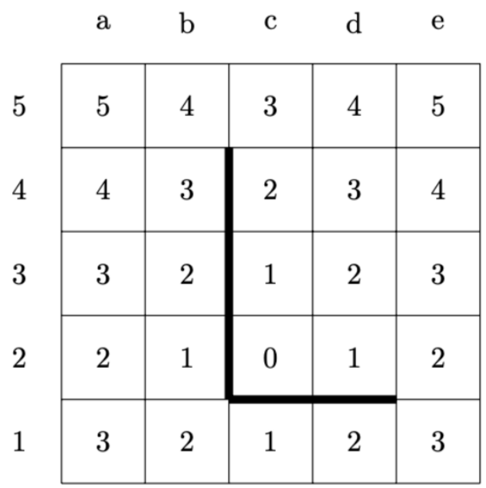


Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

Node	f = g + h	expanded





Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

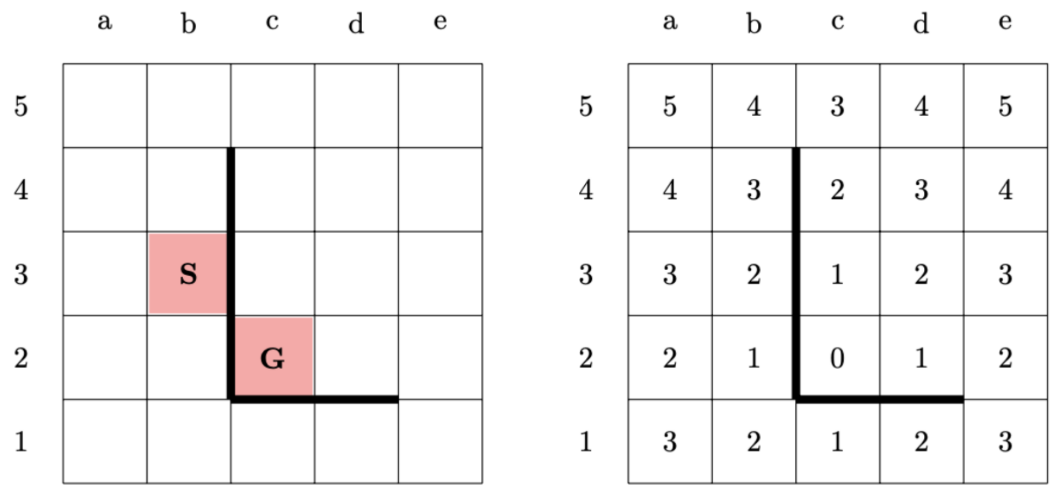
Node	f = g + h	expanded
B3		

	a	b	c	d	e
5					
4					
3		**S**			
2			**G**		
1					

	a	b	c	d	e
5	5	4	3	4	5
4	4	3	2	3	4
3	3	2	1	2	3
2	2	1	0	1	2
1	3	2	1	2	3

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

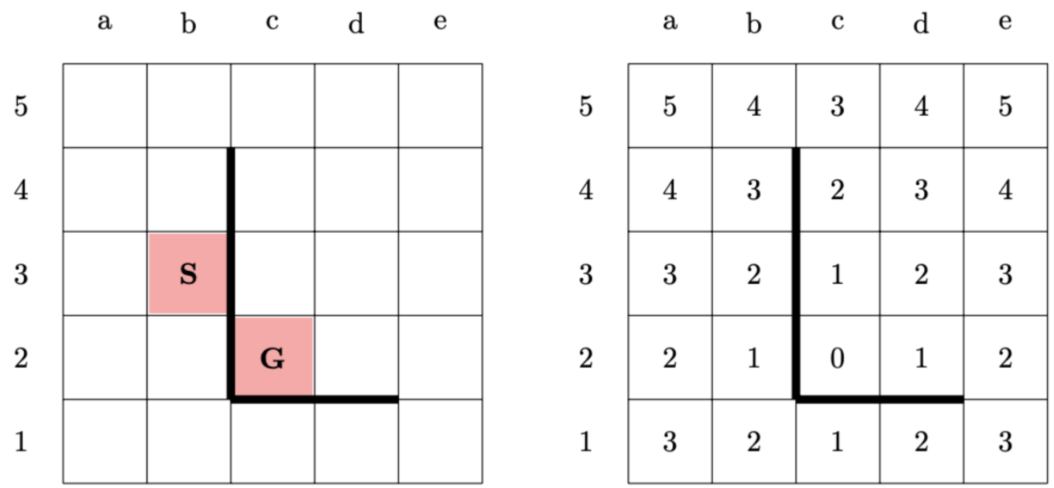


Node	f = g + h	expanded
B3	0+2=2	

Exercise 3: Application of A*

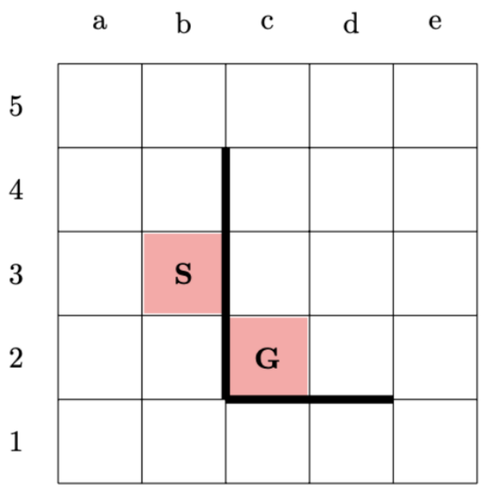
(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

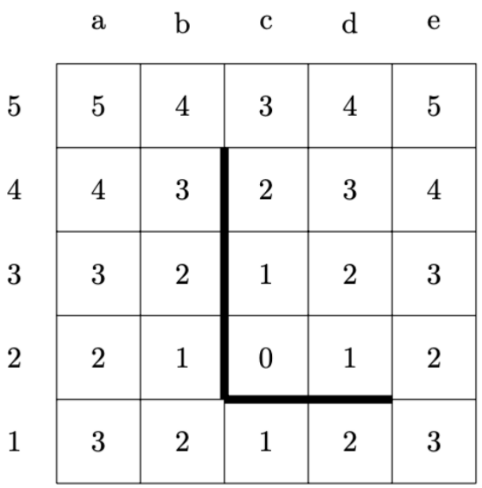
Node	f = g + h	expanded
B3	0+2=2	✓
B4		
A3		
B2		



Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

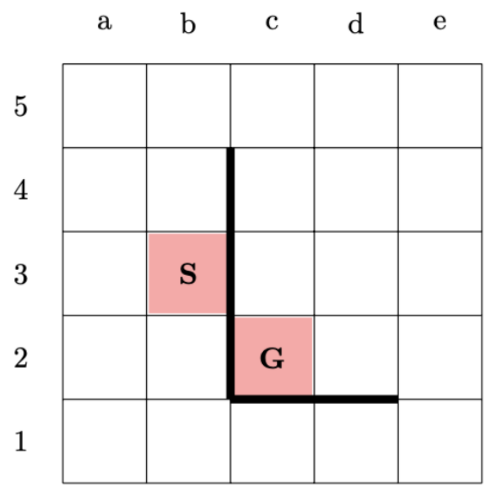


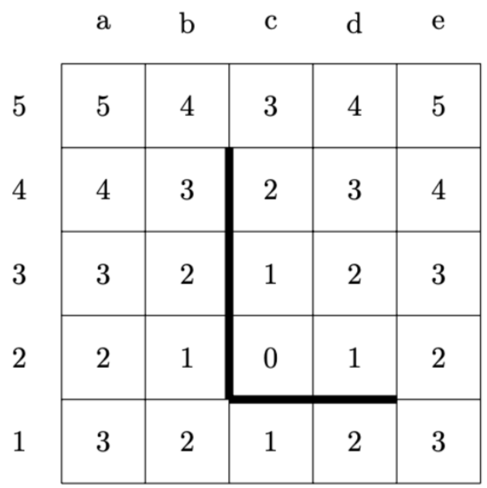


Node	f = g + h	expanded
B3	0+2=2	✓
B4	1+3=4	
A3	1+3=4	
B2	1+1=2	✓
A2	2+2=4	
B1	2+2=4	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

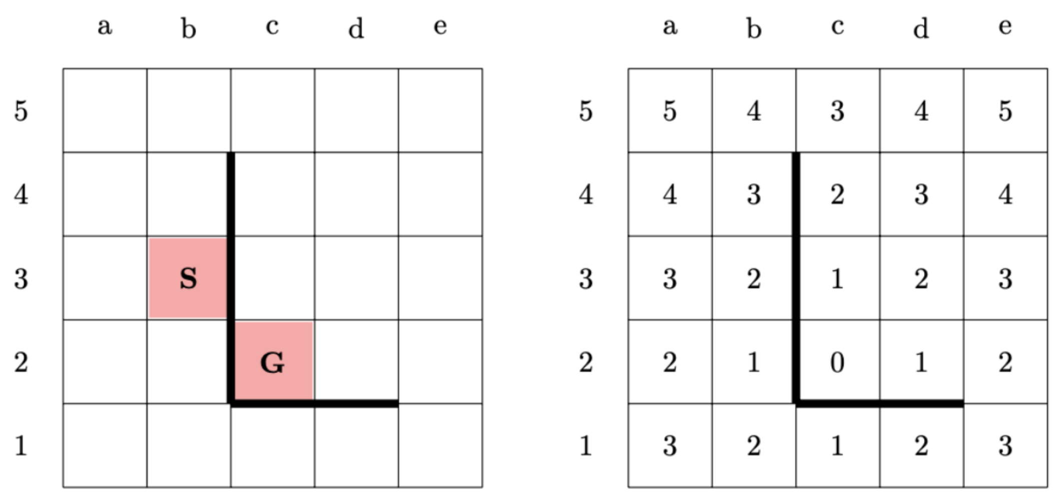




Node	f = g + h	expanded
B3	0+2=2	✓
B4	1+3=4	✓
A3	1+3=4	
B2	1+1=2	✓
A2	2+2=4	
B1	2+2=4	
B5	2+4=6	
A4	2+4=6	

Exercise 3: Application of A*

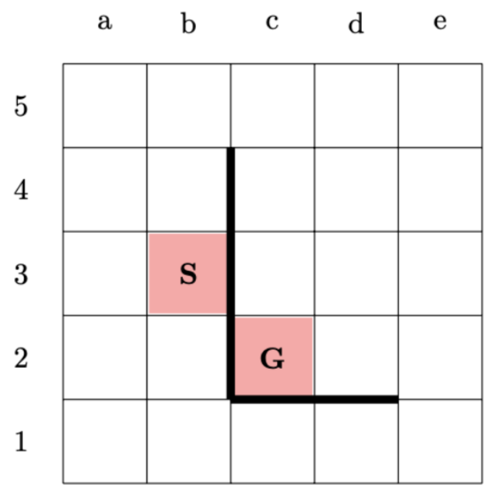
(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

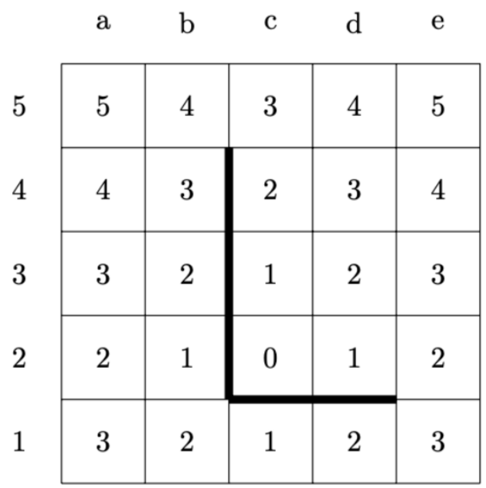


Node	f = g + h	expanded
B3	0+2=2	✓
B4	1+3=4	✓
A3	1+3=4	✓
B2	1+1=2	✓
A2	2+2=4	
B1	2+2=4	
B5	2+4=6	
A4	2+4=6	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

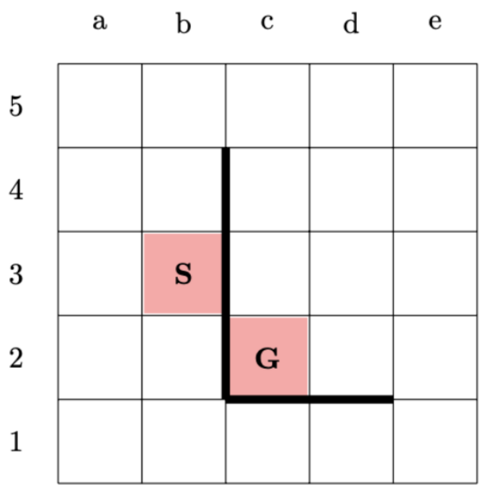


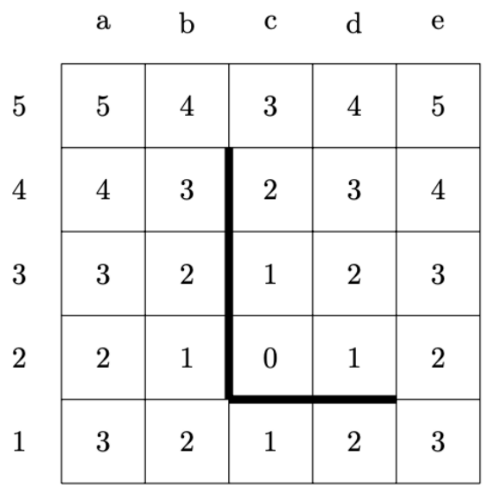


Node	f = g + h	expanded
B3	0+2=2	✓
B4	1+3=4	✓
A3	1+3=4	✓
B2	1+1=2	✓
A2	2+2=4	
B1	2+2=4	✓
B5	2+4=6	
A4	2+4=6	
A1	3+3=6	
C1	3+1=4	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

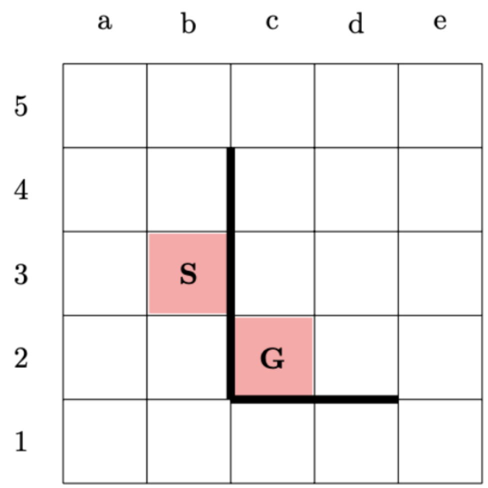




Node	f = g + h	expanded
B3	0+2=2	✓
B4	1+3=4	✓
A3	1+3=4	✓
B2	1+1=2	✓
A2	2+2=4	✓
B1	2+2=4	✓
B5	2+4=6	
A4	2+4=6	
A1	3+3=6	
C1	3+1=4	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

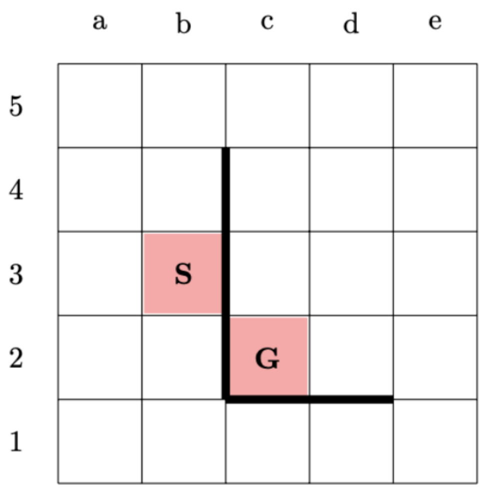


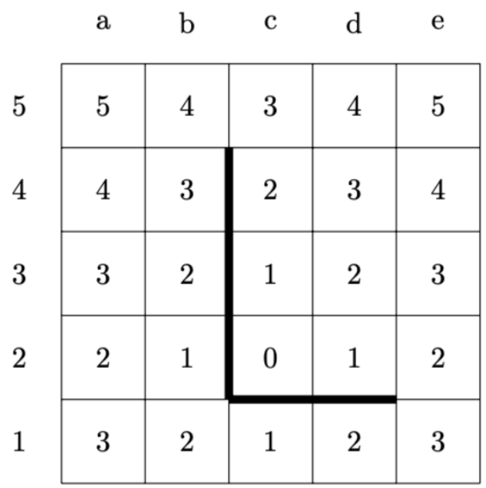
	a	b	c	d	e
5	5	4	3	4	5
4	4	3	2	3	4
3	3	2	1	2	3
2	2	1	0	1	2
1	3	2	1	2	3

Node	f = g + h	expanded
B3	0+2=2	✓
B4	1+3=4	✓
A3	1+3=4	✓
B2	1+1=2	✓
A2	2+2=4	✓
B1	2+2=4	✓
B5	2+4=6	
A4	2+4=6	
A1	3+3=6	
C1	3+1=4	✓
D1	4+2=6	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

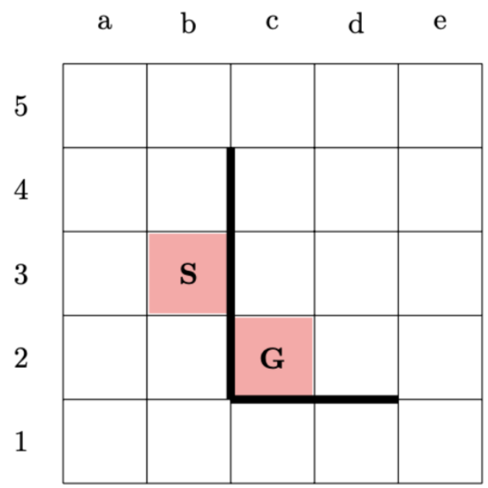


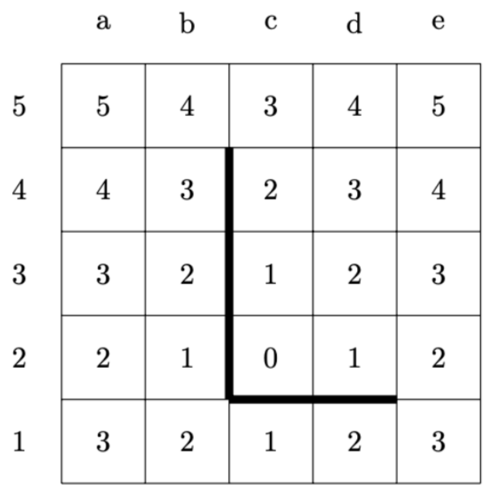


Node	f = g + h	expanded
…	…	…
B5	2+4=6	
A4	2+4=6	
A1	3+3=6	
C1	3+1=4	✓
D1	4+2=6	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

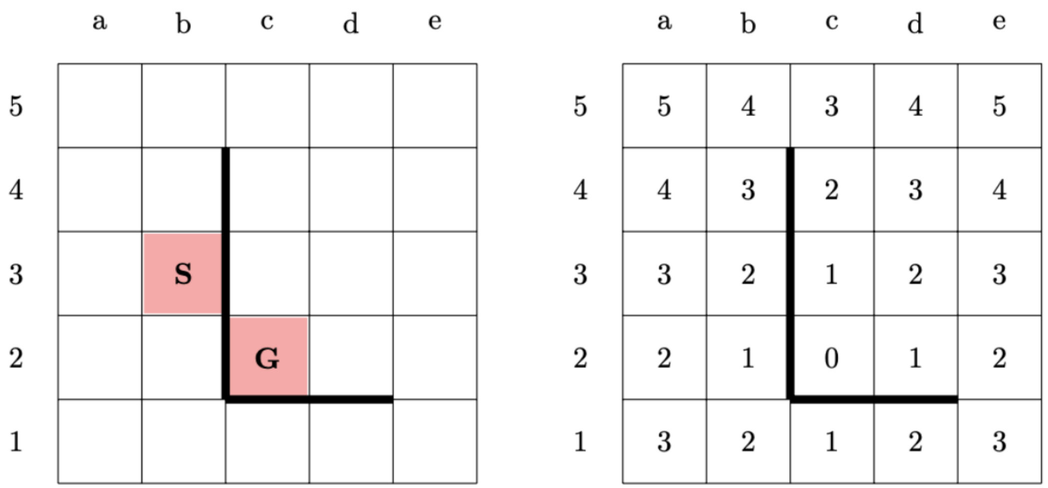




Node	f = g + h	expanded
…	…	…
B5	2+4=6	✓
A4	2+4=6	
A1	3+3=6	
C1	3+1=4	✓
D1	4+2=6	
A5	3+5=8	
C5	3+3=6	

Exercise 3: Application of A*

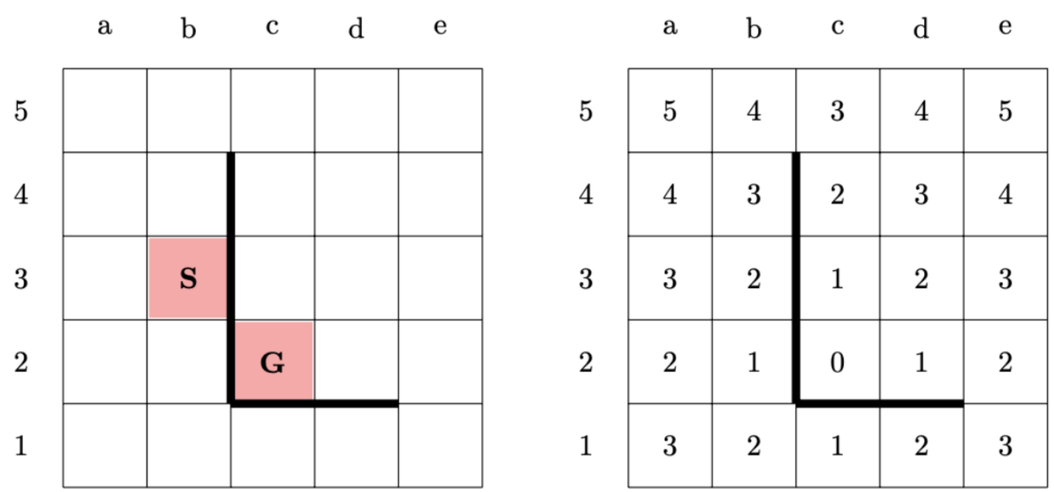
(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*



Node	f = g + h	expanded
…	…	…
B5	2+4=6	✓
A4	2+4=6	✓
A1	3+3=6	
C1	3+1=4	✓
D1	4+2=6	
A5	3+5=8	
C5	3+3=6	

Exercise 3: Application of A*

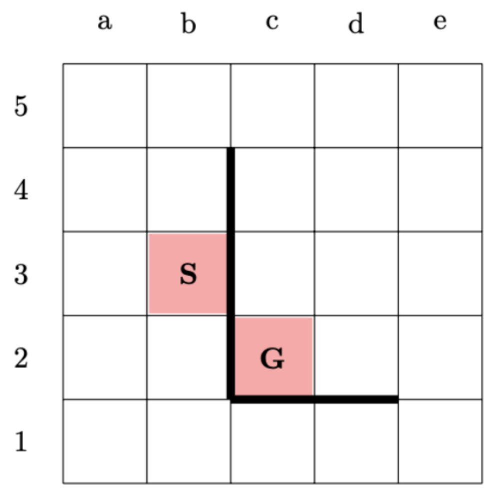
(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

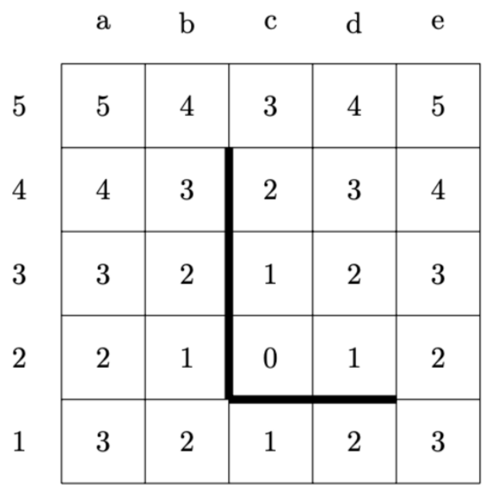


Node	f = g + h	expanded
…	…	…
B5	2+4=6	✓
A4	2+4=6	✓
A1	3+3=6	✓
C1	3+1=4	✓
D1	4+2=6	
A5	3+5=8	
C5	3+3=6	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

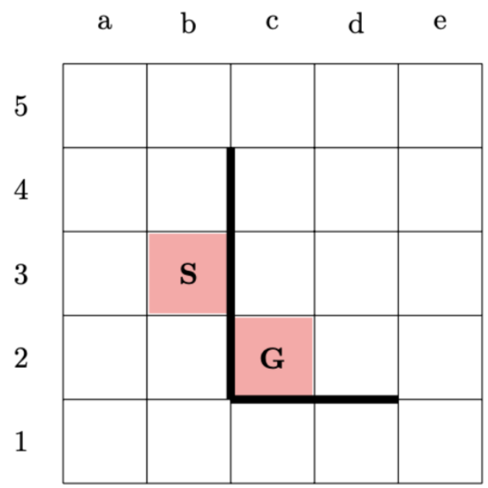




Node	f = g + h	expanded
…	…	…
B5	2+4=6	✓
A4	2+4=6	✓
A1	3+3=6	✓
C1	3+1=4	✓
D1	4+2=6	✓
A5	3+5=8	
C5	3+3=6	
E1	5+3=8	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

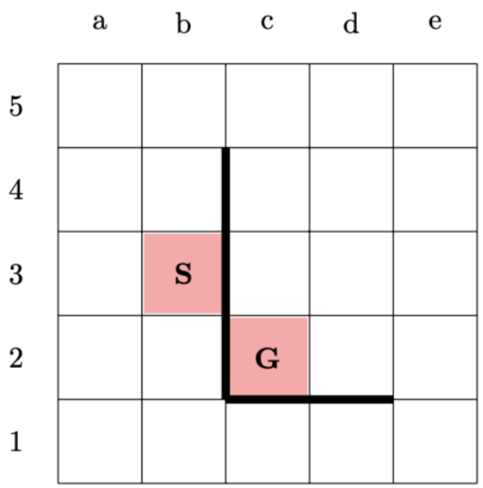


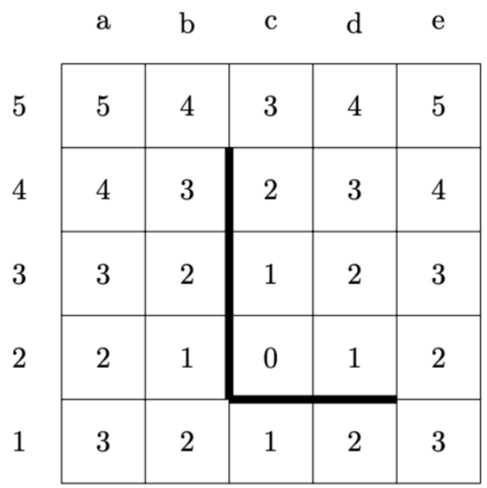
	a	b	c	d	e
5	5	4	3	4	5
4	4	3	2	3	4
3	3	2	1	2	3
2	2	1	0	1	2
1	3	2	1	2	3

Node	f = g + h	expanded
…	…	…
B5	2+4=6	✓
A4	2+4=6	✓
A1	3+3=6	✓
C1	3+1=4	✓
D1	4+2=6	✓
A5	3+5=8	✓
C5	3+3=6	
E1	5+3=8	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

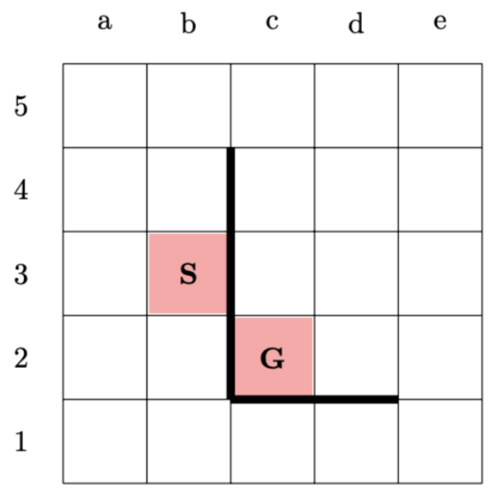


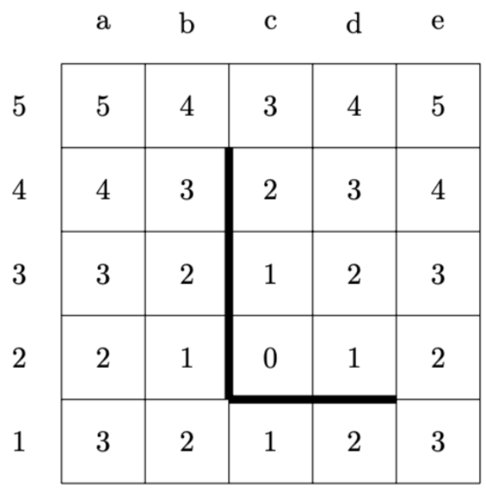


Node	f = g + h	expanded
…	…	…
B5	2+4=6	✓
A4	2+4=6	✓
A1	3+3=6	✓
C1	3+1=4	✓
D1	4+2=6	✓
A5	3+5=8	✓
C5	3+3=6	✓
E1	5+3=8	
D5	4+4=8	
C4	4+2=6	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

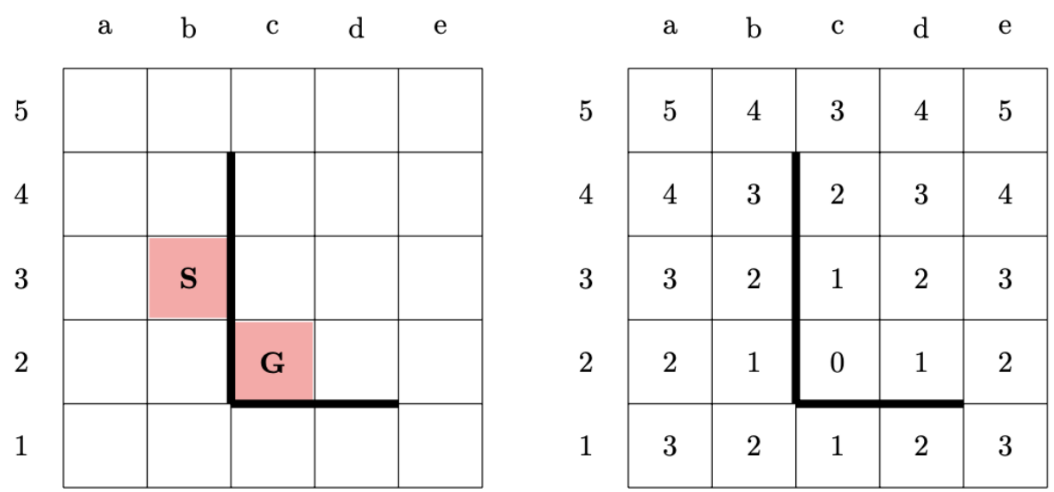




Node	f = g + h	expanded
…	…	…
E1	5+3=8	
D5	4+4=8	
C4	4+2=6	

Exercise 3: Application of A*

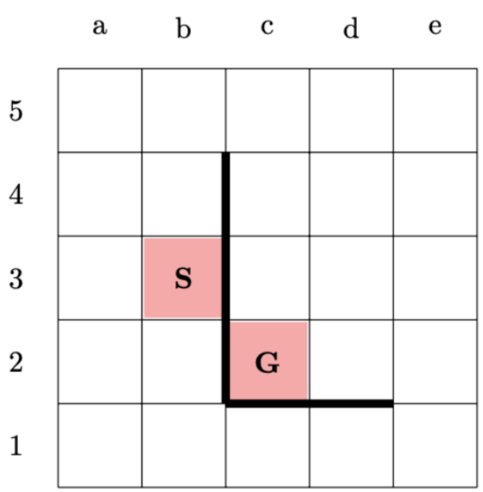
(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

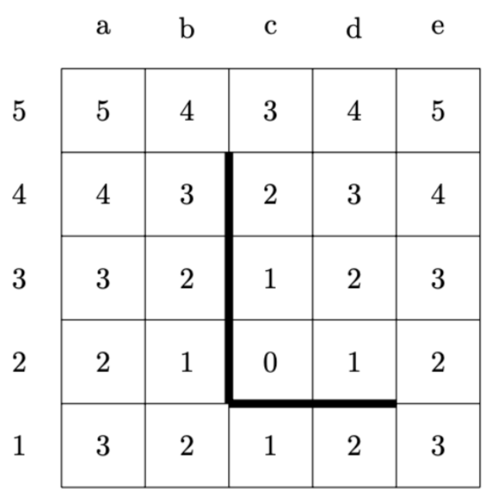


Node	f = g + h	expanded
…	…	…
E1	5+3=8	
D5	4+4=8	
C4	4+2=6	✓
D4	5+3=8	
C3	5+1=6	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*





Node	f = g + h	expanded
…	…	…
E1	5+3=8	
D5	4+4=8	
C4	4+2=6	✓
D4	5+3=8	
C3	5+1=6	✓
D3	6+2=8	
C2	6+0=6	

Exercise 3: Application of A*

(a) Perform an A search. Enter the f-values of all generated nodes into the corresponding cells in the left figure. All other cells should remain empty.*

	a	b	c	d	e
5	8	6	6	8	
4	6	4	6	8	
3	4	**S**	6	8	
2	4	2	**G**		
1	6	4	4	6	8

	a	b	c	d	e
5	5	4	3	4	5
4	4	3	2	3	4
3	3	2	1	2	3
2	2	1	0	1	2
1	3	2	1	2	3

Node	f = g + h	expanded
…	…	…
E1	5+3=8	
D5	4+4=8	
C4	4+2=6	✓
D4	5+3=8	
C3	5+1=6	✓
D3	6+2=8	
C2	6+0=6	

Exercise 3: Application of A*

	a	b	c	d	e
5	5	4	3	4	5
4	4	3	2	3	4
3	3	2	1	2	3
2	2	1	0	1	2
1	3	2	1	2	3

(b) is the heuristic admissible?

Yes, since it never overestimates

Can someone explain why a heuristic shall not overestimate?

(c) How many nodes must A expand when h*(n) is used as a heuristic, where h*(n) is the actual cost of an optimal path from cell n to goal G?*

- B3 (start), then follow shortest path
- So 6, if target is not expanded: B3, B4, B5, C5, C4, C3

Exercise 4

Exercise 4: Application of A*

The 8-puzzle consists of 9 fields (3×3), on which 8 movable tiles are arranged; thus, one field remains free. Tiles that lie next to the free field can be moved into it. The goal is to arrange the tiles so that the following target state is reached:

1	2	3
4	5	6
7	8	

Run the A algorithm to solve the 8-puzzle. Use the following start state for this:*

1	2	3
	4	6
7	5	8

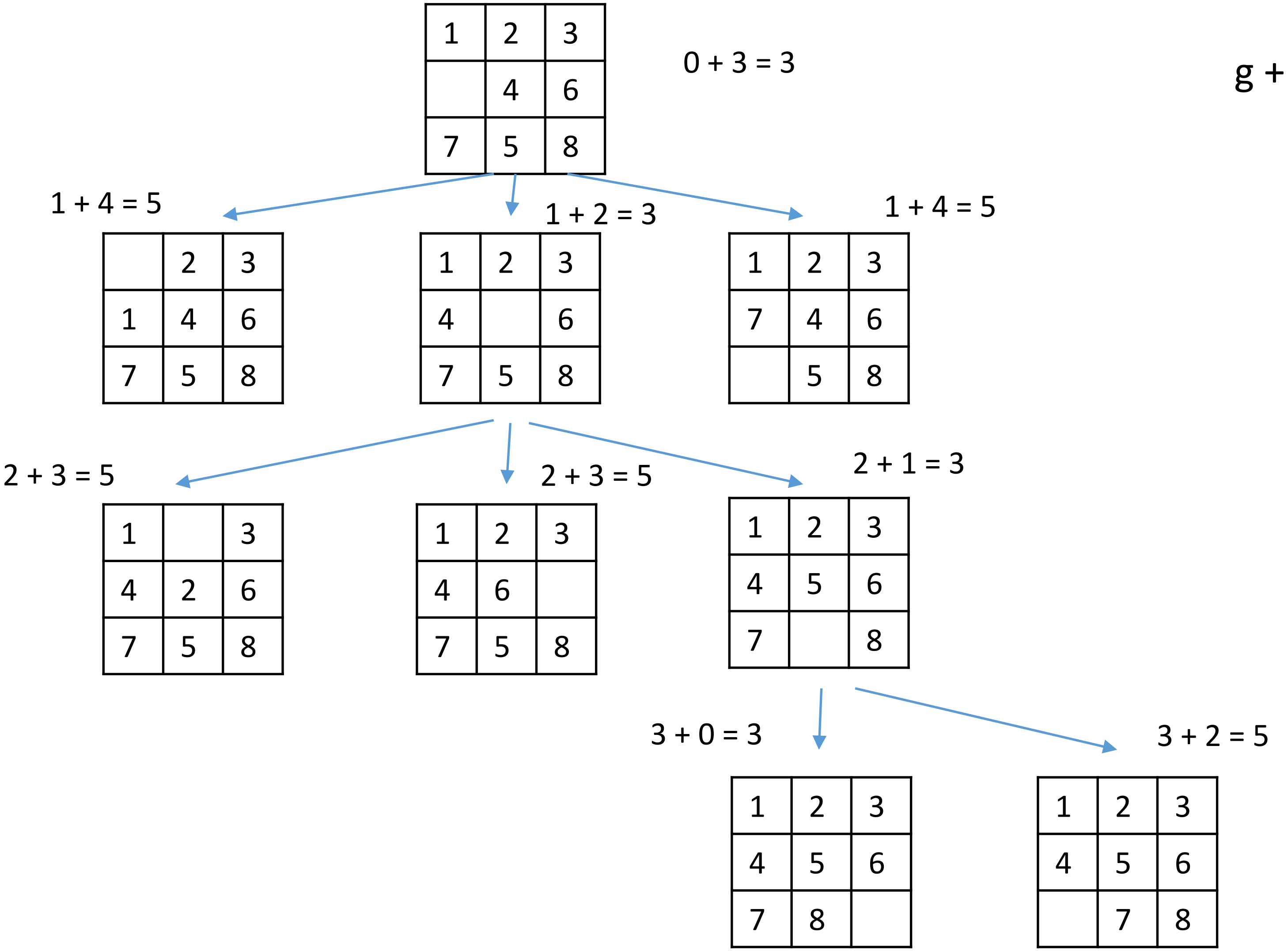
Exercise 4: Application of A*

(a) Draw the search tree with the states visited by the algorithm as nodes. Also enter the corresponding f, g and h values for each node. The heuristic to be used is the number of misplaced tiles ("misplaced tiles“ heuristic). So for the initial state, the heuristic would estimate a cost of 3 (tiles 4, 5 and 8 are misplaced). Moving a tile has an actual cost of 1. We assume that the algorithm remembers states that have already been visited and does not visit them again.

1	2	3
4	5	6
7	8	

1	2	3
	4	6
7	5	8

Exercise 4: Application of A*



g + h = f

Exercise 4: Application of A*

(b) What is the maximum cost that the heuristic from (a) can estimate in the 8-puzzle?

All 8 tiles wrong → 8

Exercise 4: Application of A*

(c) Another heuristic that can be used in the 8-puzzle is the summed Manhattan-Distance of all misplaced tiles to each tile's destination.

(i) Is there a state for which this heuristic estimates a lower value than the "Misplaced Tiles“ heuristic?

- No
- A misplaced tile automatically has Manhattan distance of at least 1
- → Summed MD can not be lower then the number of misplaced tiles

Exercise 4: Application of A*

(c) Another heuristic that can be used in the 8-puzzle is the summed Manhattan-Distance of all misplaced tiles to each tile's destination.

(ii) Give an example of a state where this heuristic estimates a higher cost than the "Misplaced Tiles" heuristic

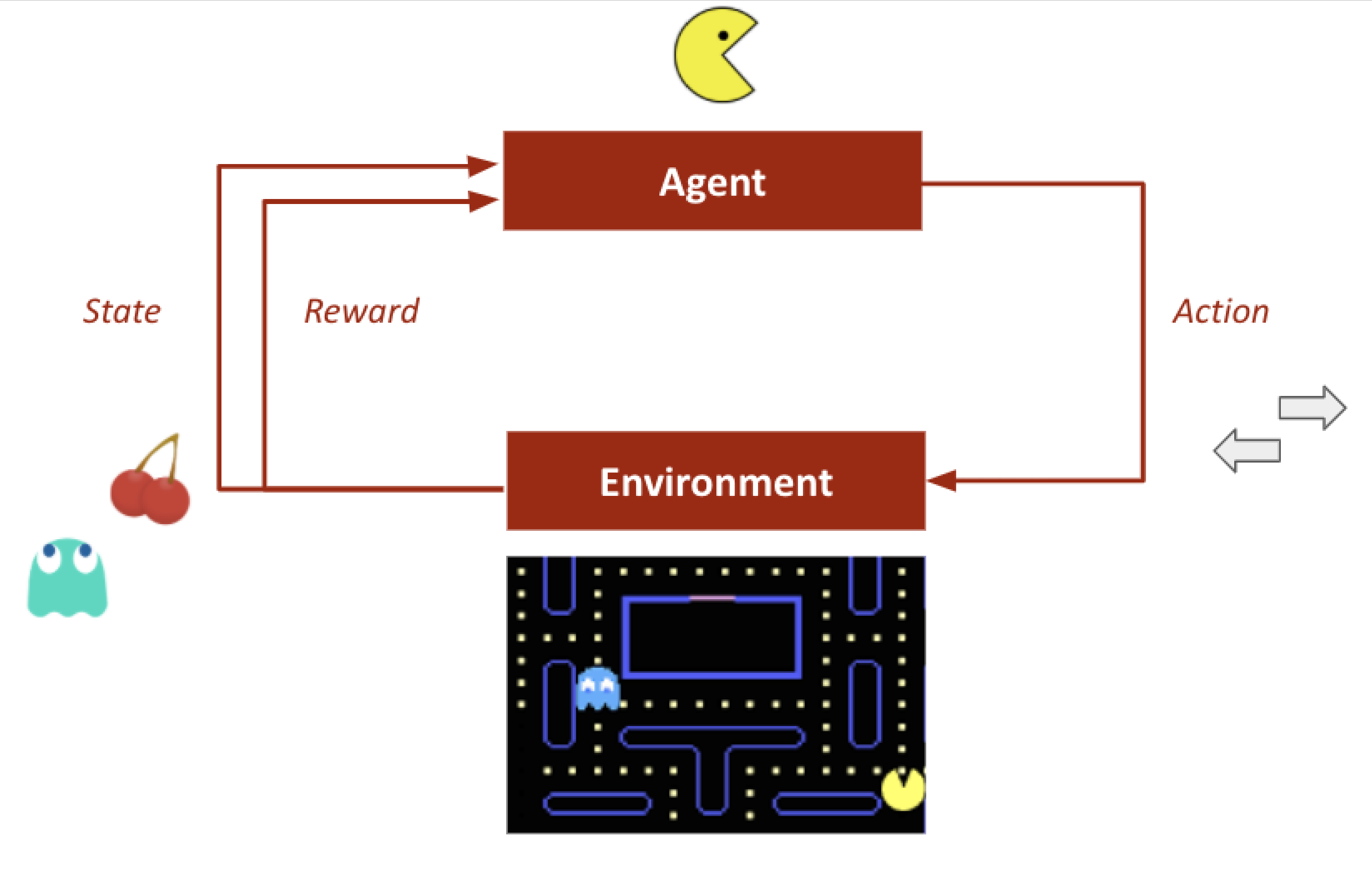
→ At least one misplaced tile must have MD > 1

3	2	1
6	5	4
	8	7

2+0+2+
2+0+2+
2+0 → 10 > 5

Intermezzo

Intermezzo: Advanced goal reaching algorithms



Intermezzo: Advanced goal reaching algorithms

- *Markov Decision Process (MDP):* $< S, A, T, R, \gamma >$

- *S = Set of all possible states*
- *A = Set of all possible actions the agent can perform*
- $T: S \times A \rightarrow S$ *state transition function (if deterministic → MDP, if probabilistic → POMDP)*
- $R: S \times A \rightarrow \mathbb{R}$ *reward function*
- $\gamma \in [0,1]$ *= discount factor (prefer instantaneous rewards over delayed rewards)*

Intermezzo: Advanced goal reaching algorithms

- *Markov Decision Process (MDP):* $< S, A, T, R, \gamma >$
- *Policy*: mapping of states to actions

$$\pi(s_t) = a_t$$

- Learn a policy which maximizes expected cumulative reward

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \left(\sum_t \gamma^t r(s_t) \mid a_t \sim \pi(s_t) \right)$$

Intermezzo: Advanced goal reaching algorithms

Goal reaching domain:

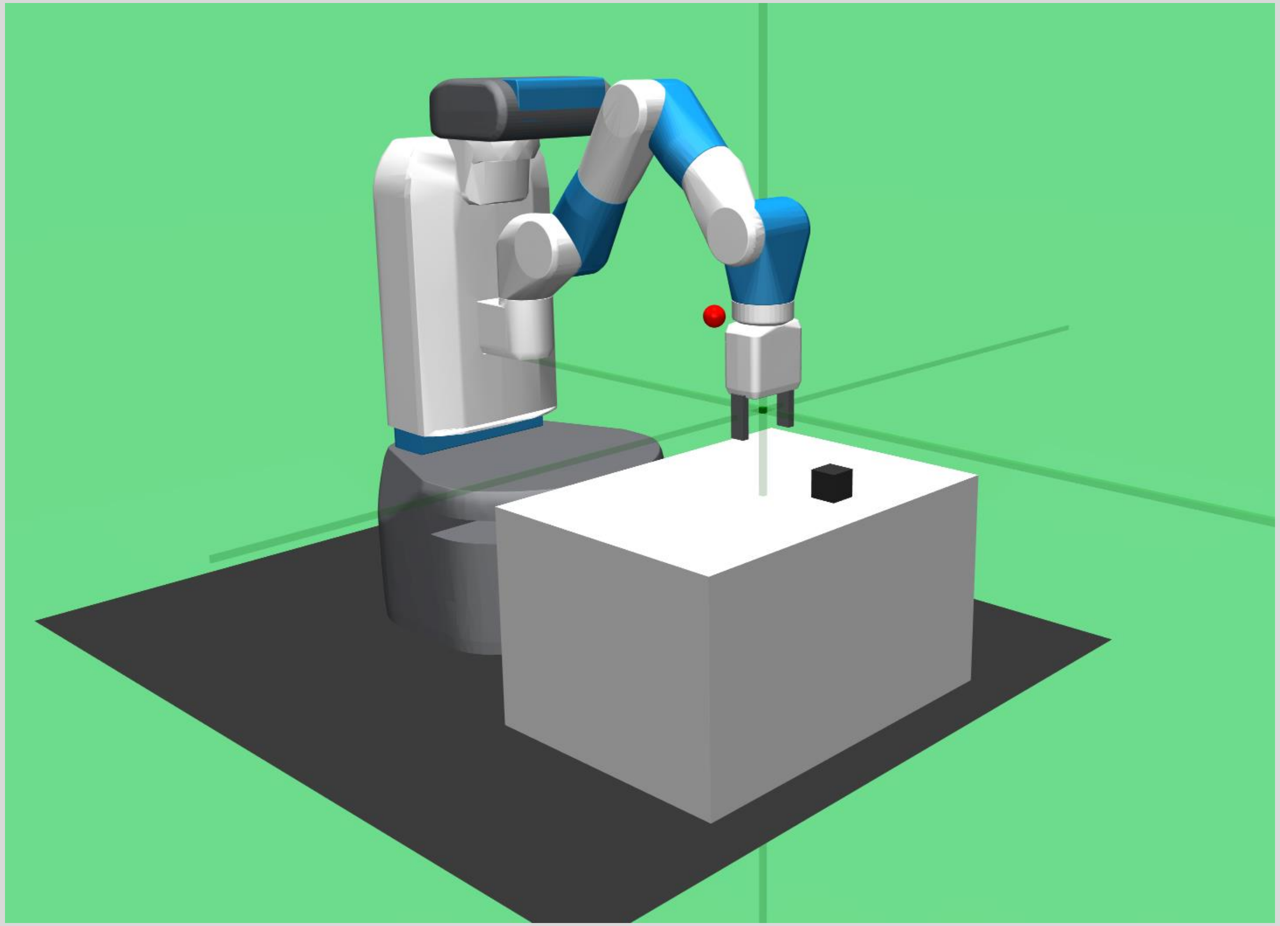
- *Agent is given a goal g_d (desired goal) at the beginning of every episode which it shall achieve*

$$\pi(s_t, g_d) = a_t$$

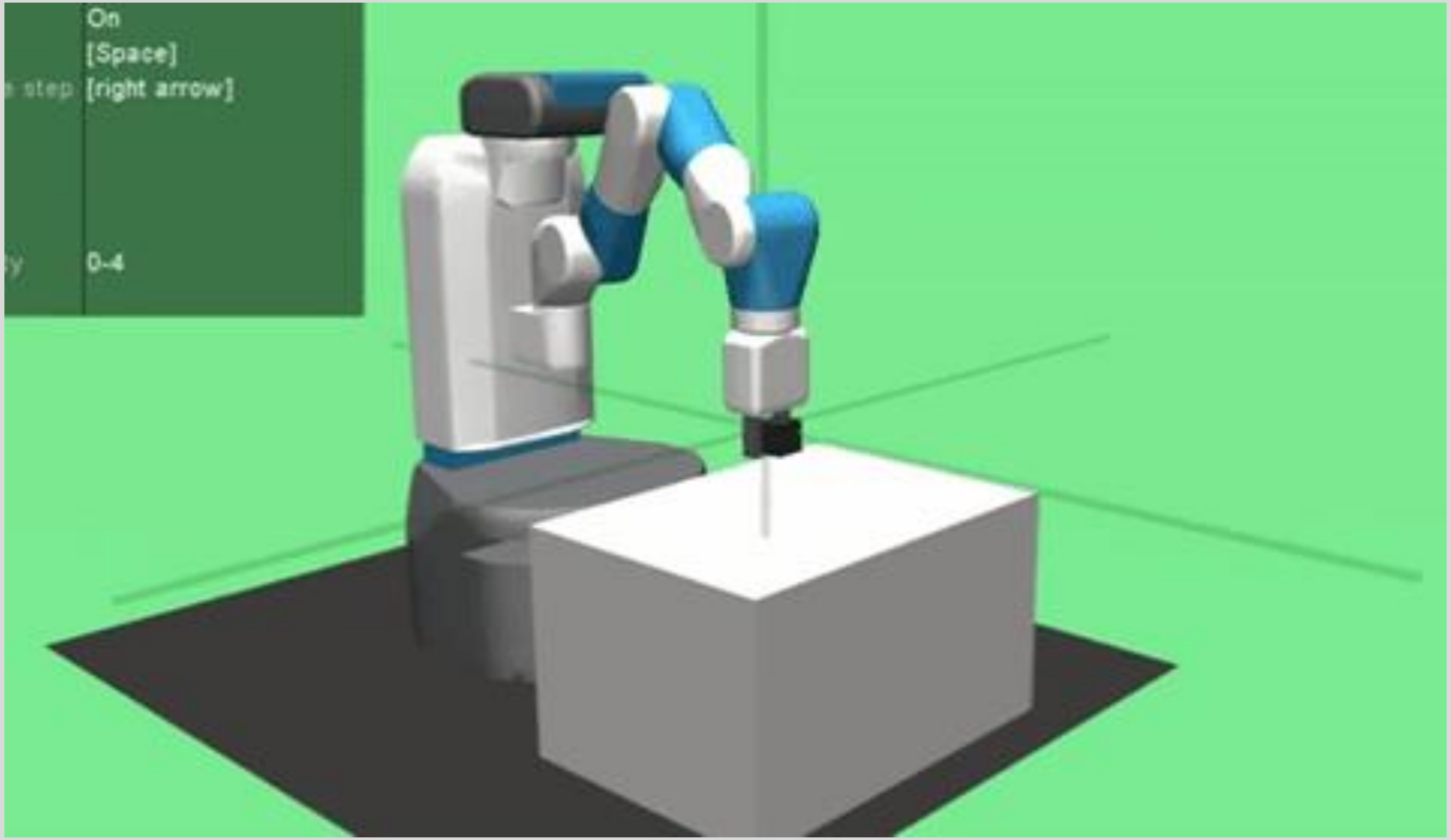
Problem: sparse rewards. Example:

$$r(s,g) = \begin{cases} 1 & \text{if s == g} \\ 0 & \text{else} \end{cases}$$

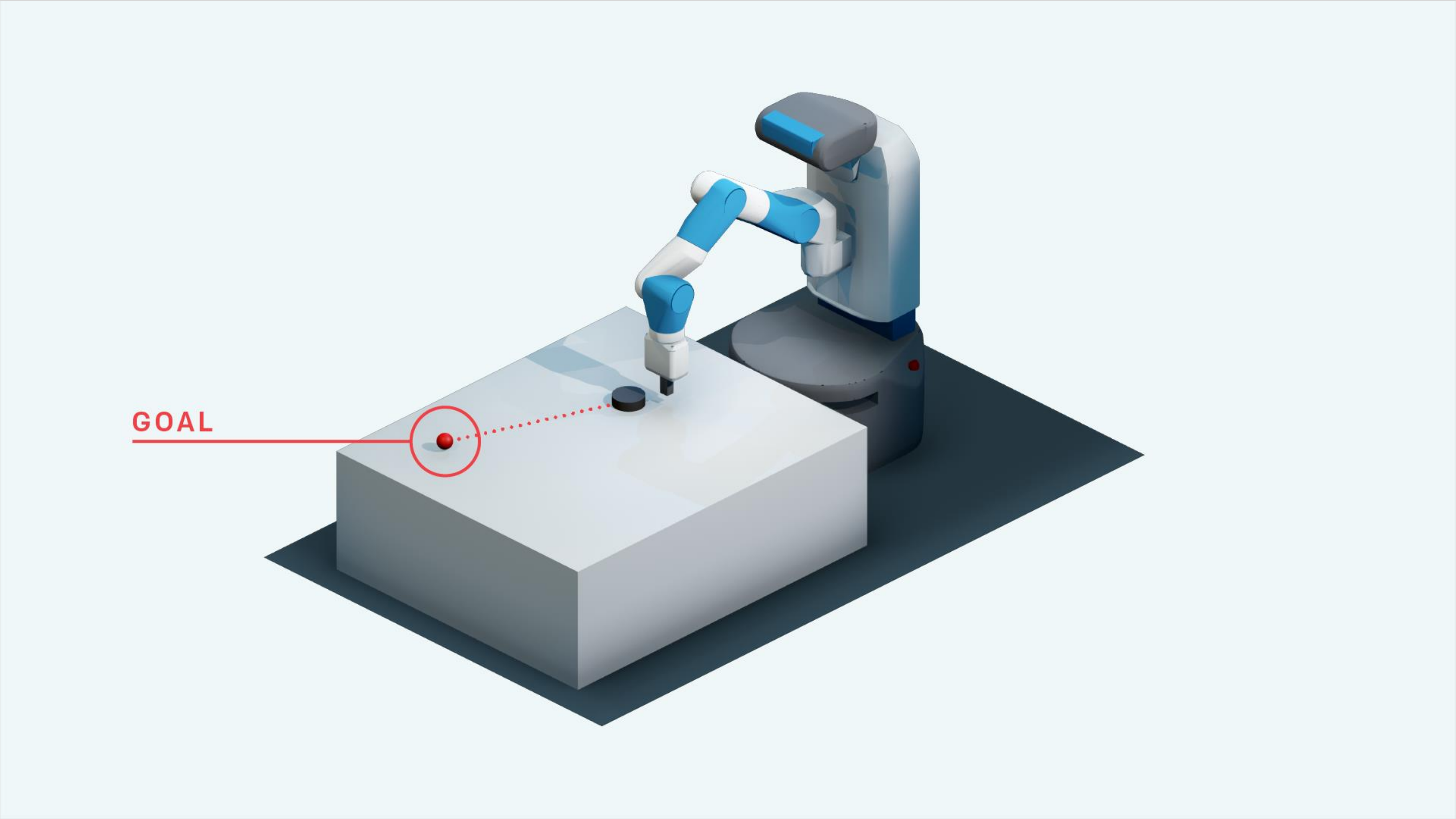
Intermezzo: Advanced goal reaching algorithms



Intermezzo: Advanced goal reaching algorithms

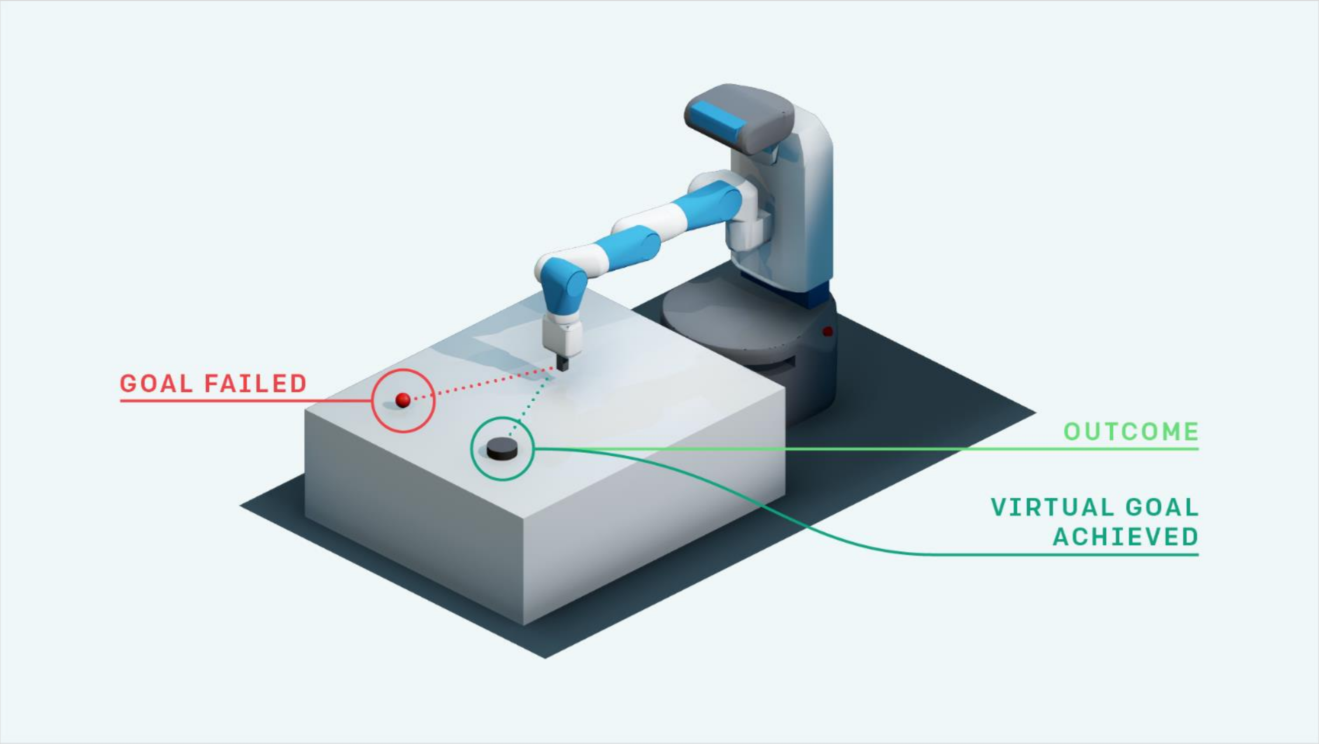


Intermezzo: Advanced goal reaching algorithms



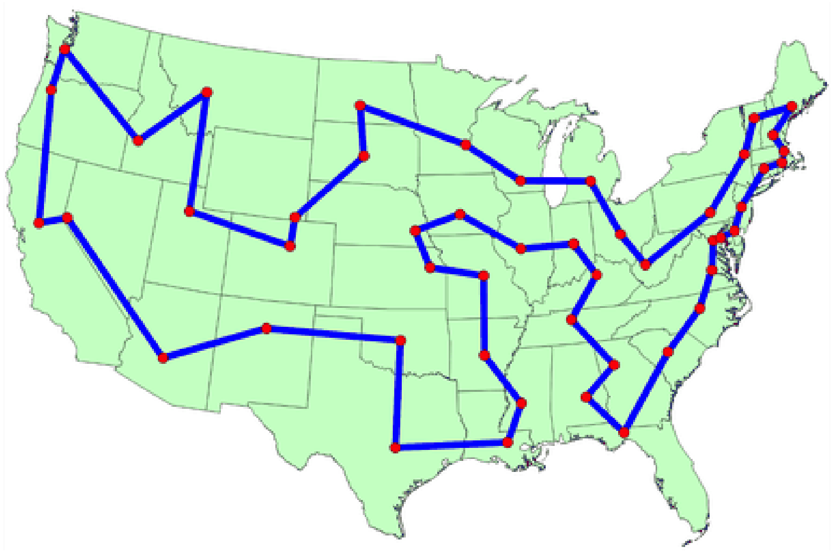
Intermezzo: Advanced goal reaching algorithms

- Sparse rewards → very few training signals → very slow training in RL
- Solution: Hindsight Experience Replay (HER)
  - Even if you fail to reach the actual goal, you still learned how you reach the actually achieved goal



Exercise 5

Exercise 5



Traveling Salesman Problem:

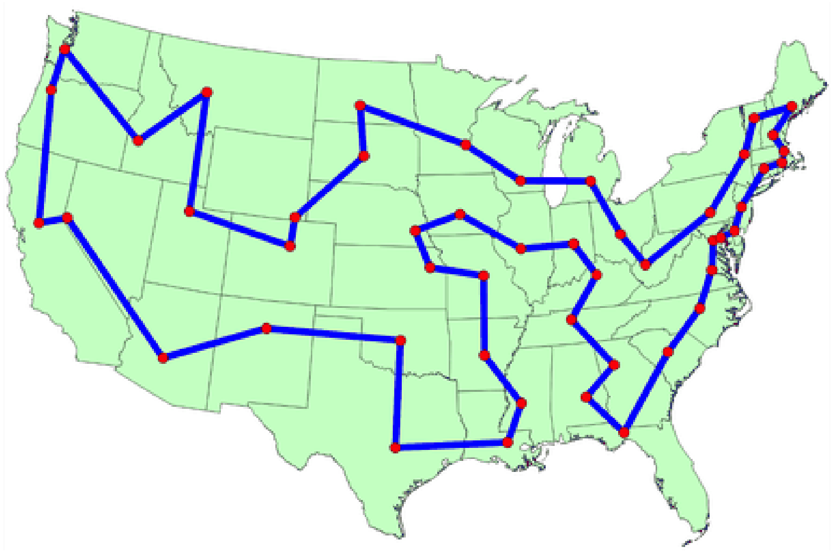
- Visit every city
- Finish the tour in the same city you started from
- Minimize the length of the tour

How can a problem solution (a state) be represented for these problems?

For a problem with N cities we need an array x of length N.

$x[i]$ = the city to be visited in step i

Exercise 5



Traveling Salesman Problem:

- Visit every city
- Finish the tour in the same city you started from
- Minimize the length of the tour

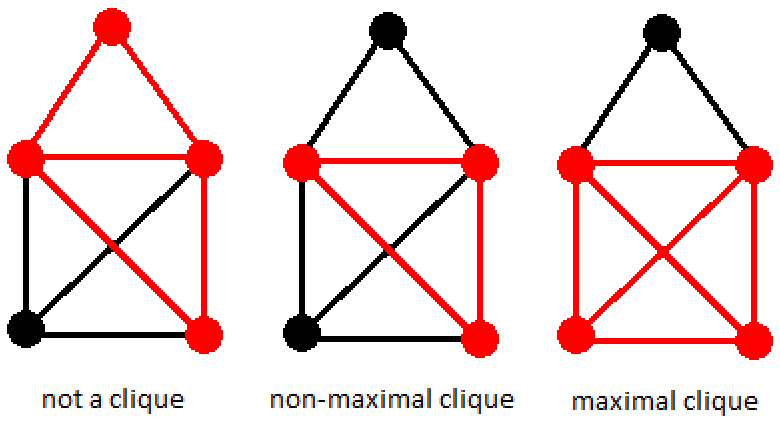
How can a neighboring state be constructed given another state?

Given state x

set of neighbor states:

$$N(x) = \{x \; with \; x[i] \; and \; x[j] \; swapped \mid \forall \, i \in [1, N] \; \forall \, j \in [1, N]: i \neq j\}$$

Exercise 5



Maximum Clique:

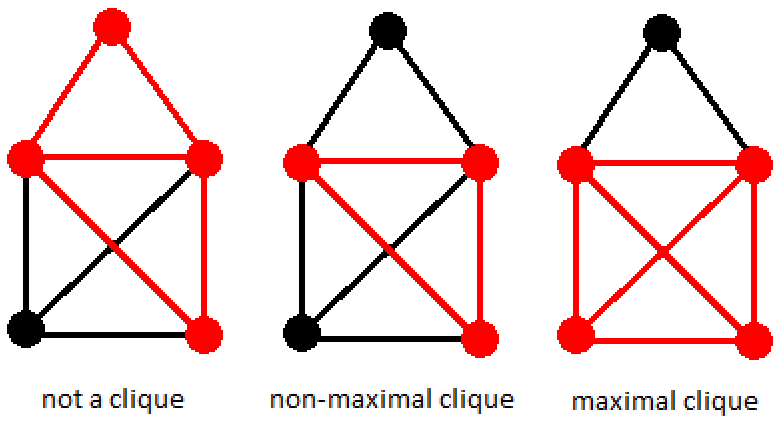
- Given an undirected graph
- Determine the largest set of nodes
  with all nodes being pairwise connected by an edge

How can a problem solution (a state) be represented for these problems?

For a problem with N vertices we need a binary array x of length N.

$x[i] = 1 \;\Leftrightarrow\;$ node i is part of the clique

Exercise 5



Maximum Clique:

- Given an undirected graph
- Determine the largest set of nodes
  with all nodes being pairwise connected by an edge

How can a neighboring state be constructed given another state?

Given state x

set of neighbor states:

$$N(x) = \{x\ with\ flipped\ x[i]\ \forall\ i \in [1, N]\}$$

Exercise 6

Exercise 6

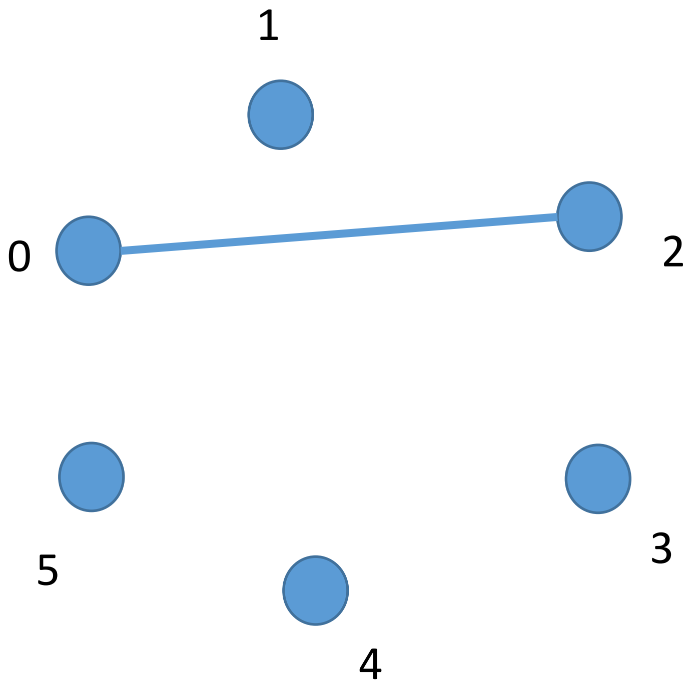
Consider the following (undirected) graph represented by an adjacency matrix:

0	0	1	0	1	1
0	0	0	1	1	1
1	0	0	0	1	1
0	1	0	0	1	1
1	1	1	1	0	1
1	1	1	1	1	0

a) Draw the graph

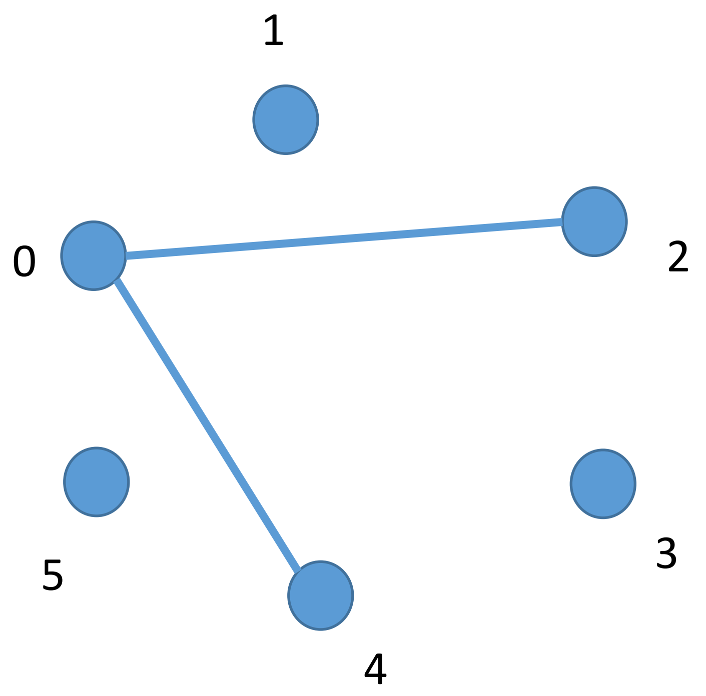
Exercise 6

	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	1	1	1
2	1	0	0	0	1	1
3	0	1	0	0	1	1
4	1	1	1	1	0	1
5	1	1	1	1	1	0



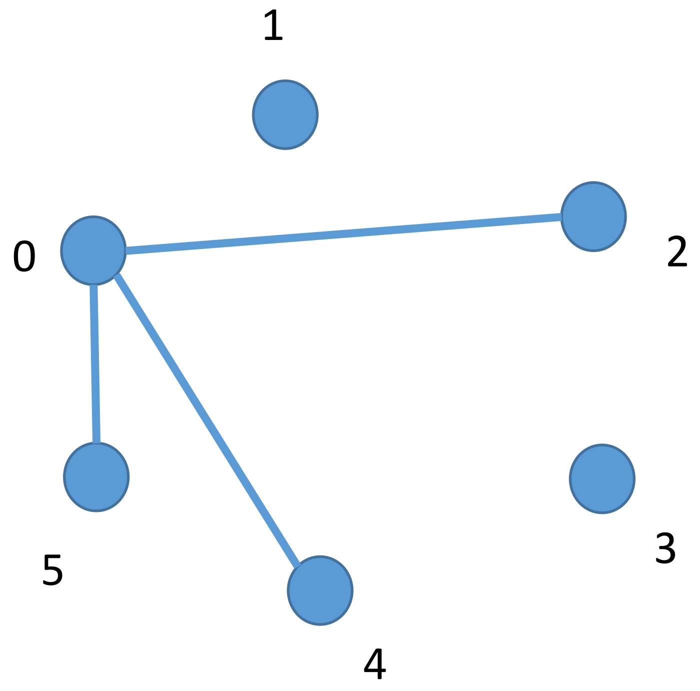
Exercise 6

	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	1	1	1
2	1	0	0	0	1	1
3	0	1	0	0	1	1
4	1	1	1	1	0	1
5	1	1	1	1	1	0



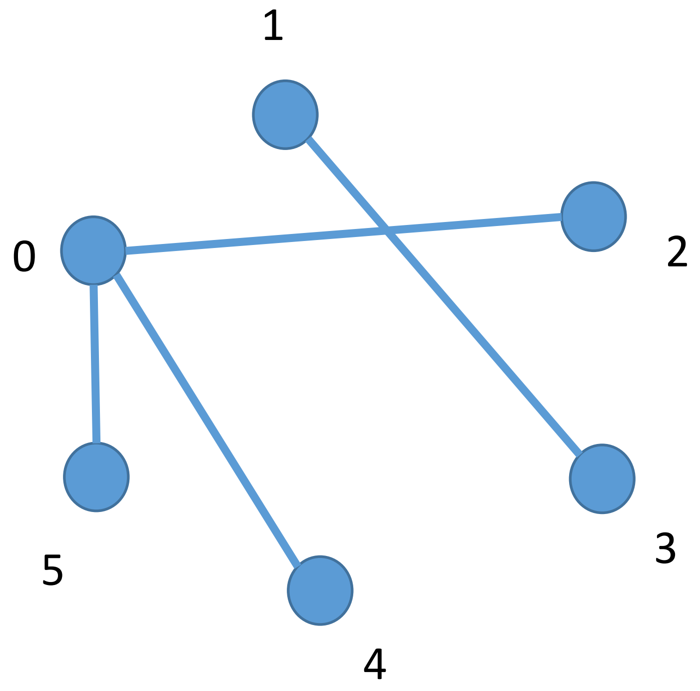
Exercise 6

	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	1	1	1
2	1	0	0	0	1	1
3	0	1	0	0	1	1
4	1	1	1	1	0	1
5	1	1	1	1	1	0



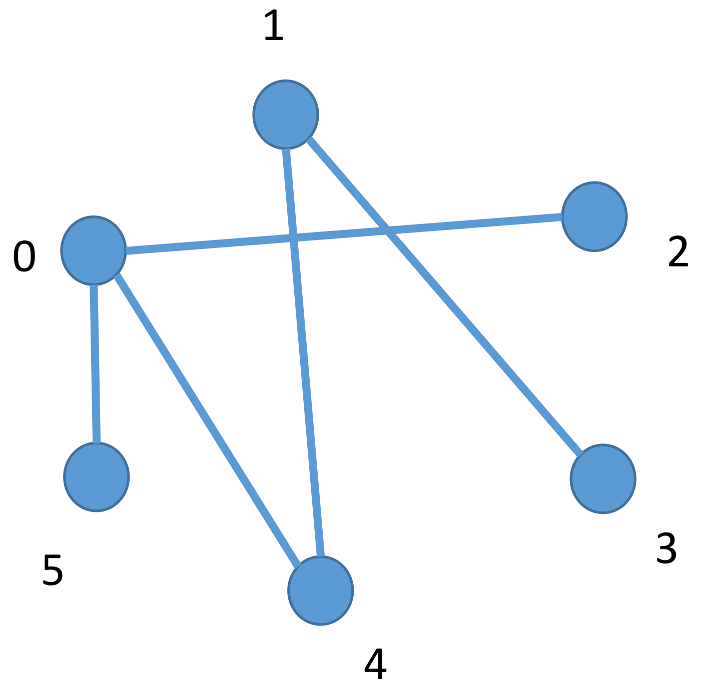
Exercise 6

	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	1	1	1
2	1	0	0	0	1	1
3	0	1	0	0	1	1
4	1	1	1	1	0	1
5	1	1	1	1	1	0



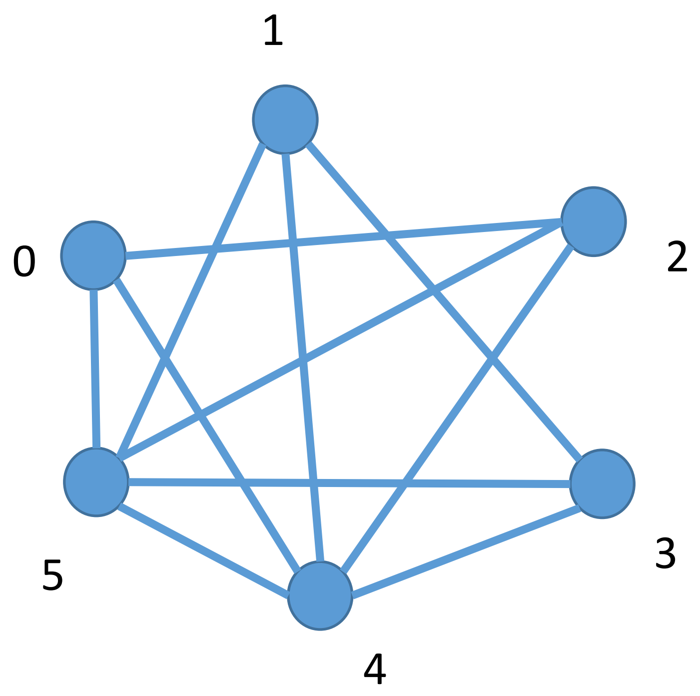
Exercise 6

	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	1	1	1
2	1	0	0	0	1	1
3	0	1	0	0	1	1
4	1	1	1	1	0	1
5	1	1	1	1	1	0



Exercise 6

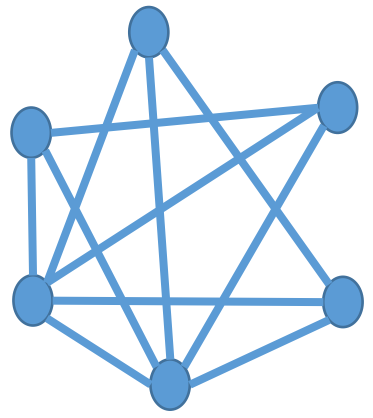
$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
2 & 1 & 0 & 0 & 0 & 1 & 1 \\
3 & 0 & 1 & 0 & 0 & 1 & 1 \\
4 & 1 & 1 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
$$



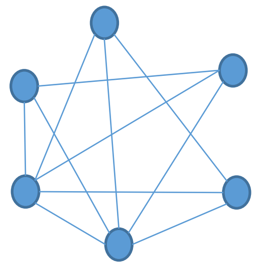
Exercise 6

b) Write the objective function for Maximum Clique and the graph G in Python.

```
def objective_maxClique(x, V, E):
        """
        x: solution candidate
        V: number of vertices in graph G
        E: list containing all edges of graph G
        returns: 0 if x does not represent a clique and sum(x) otherwise
        """
        for i in range(V):
                for j in range(V):
                        if i != j and x[i] and x[j] and (i,j) not in E:
                                return 0
        return sum(x)
```



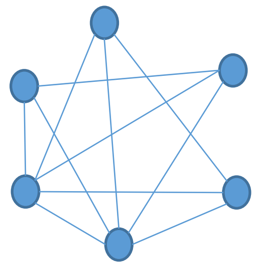
Exercise 6



c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	**Value of** $s_{current}$	**Tabu list**	s_{best}	**Value of** s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2					

Exercise 6

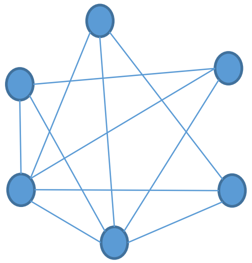


c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	**Value of** $s_{current}$	**Tabu list**	s_{best}	**Value of** s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2					

Neighbors of $s_{current}$	Value
[1,0,0,0,0,0]	1
[0,1,0,0,0,0]	1
[0,0,1,0,0,0]	1
[0,0,0,1,0,0]	1
[0,0,0,0,1,0]	1
[0,0,0,0,0,1]	1

Exercise 6

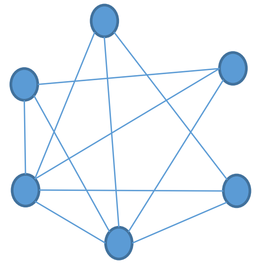


c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$	Tabu list	s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1

Neighbors of $s_{current}$	Value
[1,0,0,0,0,0]	1
[0,1,0,0,0,0]	1
[0,0,1,0,0,0]	1
[0,0,0,1,0,0]	1
[0,0,0,0,1,0]	1
[0,0,0,0,0,1]	1

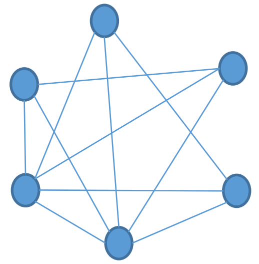
Exercise 6



c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$	Tabu list	s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3					

Exercise 6

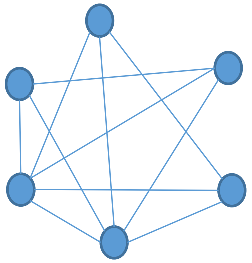


c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	**Value of** $s_{current}$	**Tabu list**	s_{best}	**Value of** s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3					

Neighbors of $s_{current}$	Value
[0,0,0,0,0,0]	tabu
[1,1,0,0,0,0]	0
[1,0,1,0,0,0]	2
[1,0,0,1,0,0]	0
[1,0,0,0,1,0]	2
[1,0,0,0,0,1]	2

Exercise 6

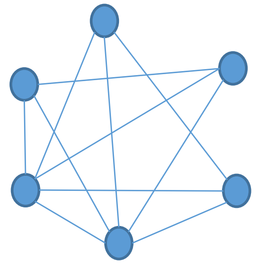


c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$	Tabu list	s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2

Neighbors of $s_{current}$	Value
[0,0,0,0,0,0]	tabu
[1,1,0,0,0,0]	0
[1,0,1,0,0,0]	2
[1,0,0,1,0,0]	0
[1,0,0,0,1,0]	2
[1,0,0,0,0,1]	2

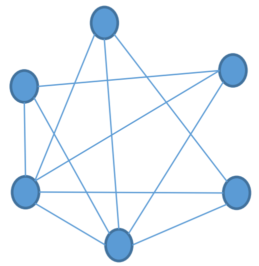
Exercise 6



c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	**Value of** $s_{current}$	**Tabu list**	s_{best}	**Value of** s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4					

Exercise 6

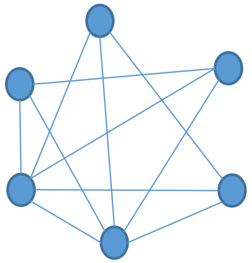


c) Maximize the oracle … starting state [0, 0, 0, 0, 0, 0]. Use a tabu list … ncluding) iteration 8.

Neighbors of $s_{current}$	Value
[0,0,1,0,0,0]	1
[1,1,1,0,0,0]	0
[1,0,0,0,0,0]	tabu
[1,0,1,1,0,0]	0
[1,0,1,0,1,0]	3
[1,0,1,0,0,1]	3

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4					

Exercise 6

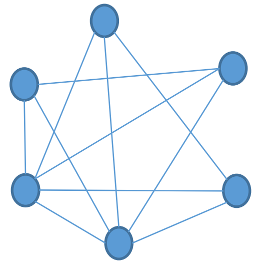


Neighbors of $s_{current}$	Value
[0,0,1,0,0,0]	1
[1,1,1,0,0,0]	0
[1,0,0,0,0,0]	tabu
[1,0,1,1,0,0]	0
[1,0,1,0,1,0]	3
[1,0,1,0,0,1]	3

c) Maximize the oracle … starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s… …ncluding) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3

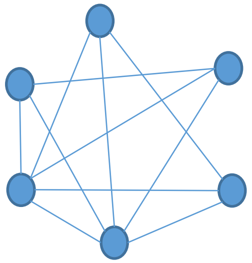
Exercise 6



c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$	Tabu list	s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5					

Exercise 6

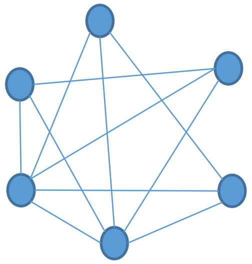


c) Maximize the oracle … starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s… ncluding) iteration 8.

Neighbors of $s_{current}$	Value
[0,0,1,0,1,0]	2
[1,1,1,0,1,0]	0
[1,0,0,0,1,0]	2
[1,0,1,1,1,0]	0
[1,0,1,0,0,0]	tabu
[1,0,1,0,1,1]	4

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5					

Exercise 6

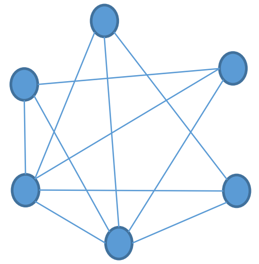


Neighbors of $s_{current}$	Value
[0,0,1,0,1,0]	2
[1,1,1,0,1,0]	0
[1,0,0,0,1,0]	2
[1,0,1,1,1,0]	0
[1,0,1,0,0,0]	tabu
[1,0,1,0,1,1]	4

c) Maximize the oracle starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s ncluding) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4

Exercise 6

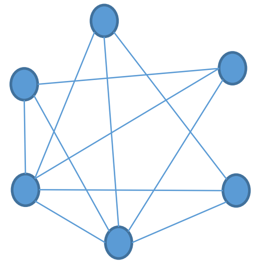


c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	**Value of** $s_{current}$	**Tabu list**	s_{best}	**Value of** s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6					

Exercise 6

Neighbors of $s_{current}$	Value
[0,0,1,0,1,1]	3
[1,1,1,0,1,1]	0
[1,0,0,0,1,1]	3
[1,0,1,1,1,1]	0
[1,0,1,0,0,1]	3
[1,0,1,0,1,0]	tabu

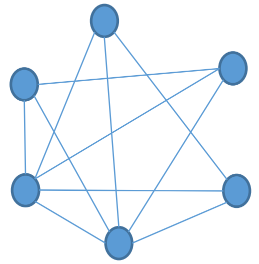


c) Maximize the oracle ... starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s ... ncluding) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6					

Exercise 6

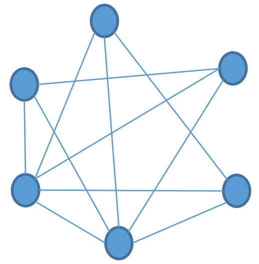
Neighbors of $s_{current}$	Value
[0,0,1,0,1,1]	3
[1,1,1,0,1,1]	0
[1,0,0,0,1,1]	3
[1,0,1,1,1,1]	0
[1,0,1,0,0,1]	3
[1,0,1,0,1,0]	tabu



c) Maximize the oracle … starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s… …ncluding) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4

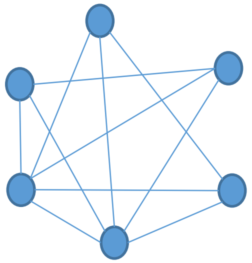
Exercise 6



c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$	Tabu list	s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4
7					

Exercise 6

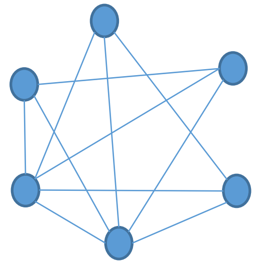


c) Maximize the oracle … starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s… …ncluding) iteration 8.

Neighbors of $s_{current}$	Value
[1,0,1,0,1,1]	tabu
[0,1,1,0,1,1]	0
[0,0,0,0,1,1]	2
[0,0,1,1,1,1]	0
[0,0,1,0,0,1]	2
[0,0,1,0,1,0]	2

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4
7					

Exercise 6

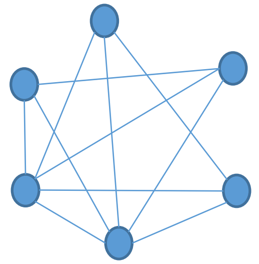


c) Maximize the oracle … starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s… …ncluding) iteration 8.

Neighbors of $s_{current}$	Value
[1,0,1,0,1,1]	tabu
[0,1,1,0,1,1]	0
[0,0,0,0,1,1]	2
[0,0,1,1,1,1]	0
[0,0,1,0,0,1]	2
[0,0,1,0,1,0]	2

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4
7	[0,0,0,0,1,1]	2	[[1,0,1,0,1,1], [0,0,1,0,1,1], [0,0,0,0,1,1]]	[1,0,1,0,1,1]	4

Exercise 6

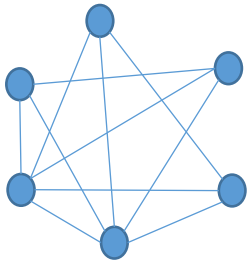


c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$	Tabu list	s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4
7	[0,0,0,0,1,1]	2	[[1,0,1,0,1,1], [0,0,1,0,1,1], [0,0,0,0,1,1]]	[1,0,1,0,1,1]	4
8					

Exercise 6

Neighbors of $s_{current}$	Value
[1,0,0,0,1,1]	3
[0,1,0,0,1,1]	3
[0,0,1,0,1,1]	tabu
[0,0,0,1,1,1]	3
[0,0,0,0,0,1]	1
[0,0,0,0,1,0]	1

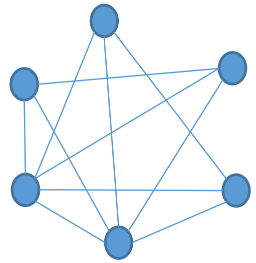


c) Maximize the oracle … starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s… ncluding) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4
7	[0,0,0,0,1,1]	2	[[1,0,1,0,1,1], [0,0,1,0,1,1], [0,0,0,0,1,1]]	[1,0,1,0,1,1]	4
8					

Exercise 6

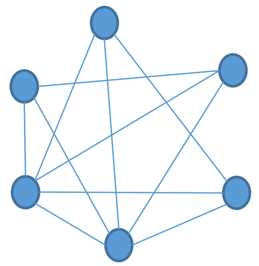
Neighbors of $s_{current}$	Value
[1,0,0,0,1,1]	3
[0,1,0,0,1,1]	3
[0,0,1,0,1,1]	tabu
[0,0,0,1,1,1]	3
[0,0,0,0,0,1]	1
[0,0,0,0,1,0]	1



c) Maximize the oracle starting state [0, 0, 0, 0, 0, 0]. Use a tabu list s ncluding) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$		s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4
7	[0,0,0,0,1,1]	2	[[1,0,1,0,1,1], [0,0,1,0,1,1], [0,0,0,0,1,1]]	[1,0,1,0,1,1]	4
8	[1,0,0,0,1,1]	3	[[0,0,1,0,1,1], [0,0,0,0,1,1], [1,0,0,0,1,1]]	[1,0,1,0,1,1]	4

Exercise 6



c) Maximize the oracle function from (b) using Tabu Search. Use the starting state [0, 0, 0, 0, 0, 0]. Use a tabu list size of 3. Complete the following table until (including) iteration 8.

Iteration	$s_{current}$	Value of $s_{current}$	Tabu list	s_{best}	Value of s_{best}
1	[0,0,0,0,0,0]	0	[[0,0,0,0,0,0]]	[0,0,0,0,0,0]	0
2	[1,0,0,0,0,0]	1	[[0,0,0,0,0,0], [1,0,0,0,0,0]]	[1,0,0,0,0,0]	1
3	[1,0,1,0,0,0]	2	[[0,0,0,0,0,0], [1,0,0,0,0,0], [1,0,1,0,0,0]]	[1,0,1,0,0,0]	2
4	[1,0,1,0,1,0]	3	[[1,0,0,0,0,0], [1,0,1,0,0,0], [1,0,1,0,1,0]]	[1,0,1,0,1,0]	3
5	[1,0,1,0,1,1]	4	[[1,0,1,0,0,0], [1,0,1,0,1,0], [1,0,1,0,1,1]]	[1,0,1,0,1,1]	4
6	[0,0,1,0,1,1]	3	[[1,0,1,0,1,0], [1,0,1,0,1,1], [0,0,1,0,1,1]]	[1,0,1,0,1,1]	4
7	[0,0,0,0,1,1]	2	[[1,0,1,0,1,1], [0,0,1,0,1,1], [0,0,0,0,1,1]]	[1,0,1,0,1,1]	4
8	[1,0,0,0,1,1]	3	[[0,0,1,0,1,1], [0,0,0,0,1,1], [1,0,0,0,1,1]]	[1,0,1,0,1,1]	4

Exercise 7

Exercise 7

Given N *players, each with individual playing strength* s_i*. Example:* $s = [10, 5, 12, 39, 32, \ldots]$

Your task is to divide the N *players into two teams of as equal strength as possible (the problem is also known as number partitioning). The team strength is the sum of all player strengths.*

Implement the Tabu Search algorithm in Python and use it to solve the above presented task.

Exercise 7: Tabu in Python

see PyCharm

No Free Lunch Theorem

No Free Lunch Theorem

Any two optimization algorithms are equivalent when their performance is averaged across all possible problems

No Free Lunch Theorem

Intuitive understanding:

An (uninformed) optimization algorithm is a sequence of oracle evaluations

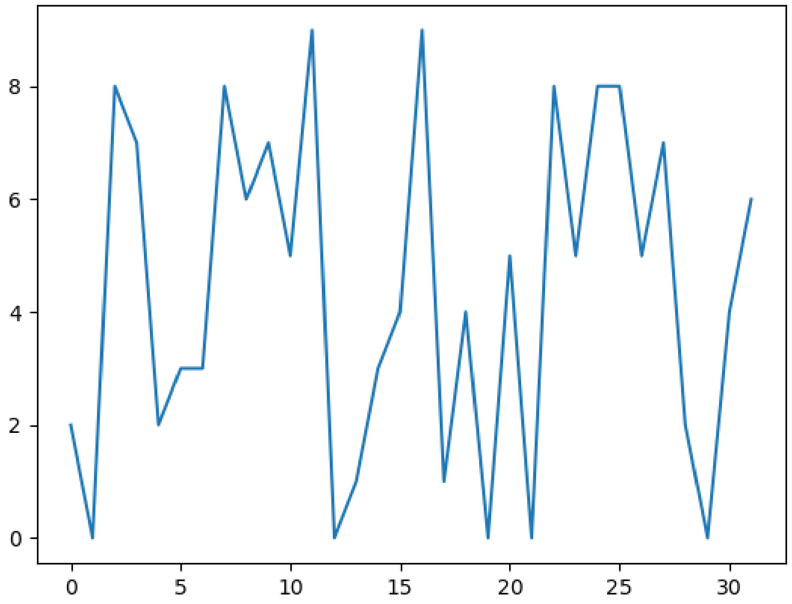
$f(x)$ # function to be optimized

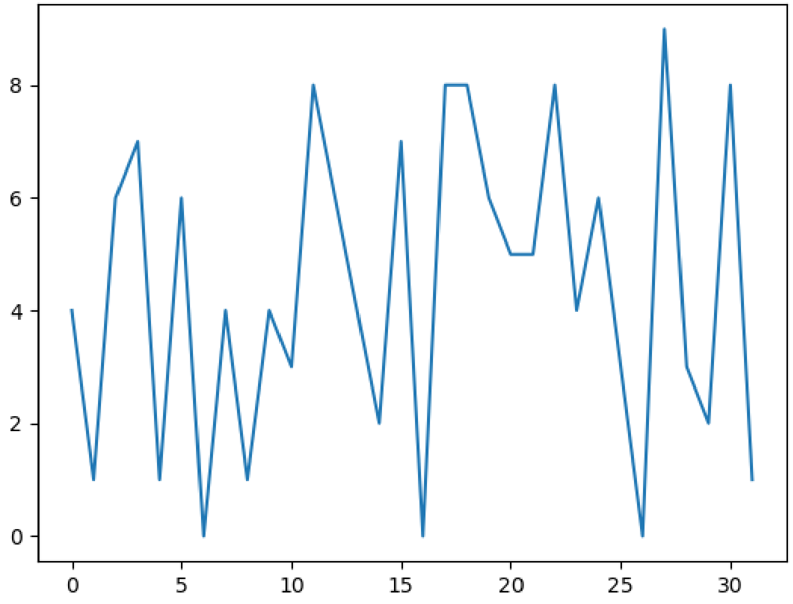
$\mathrm{f}(x_1), \mathrm{f}(x_2), \mathrm{f}(x_3), \dots$

The states x_1, x_2, x_3, … are determined by the optimization algorithm

No Free Lunch Theorem

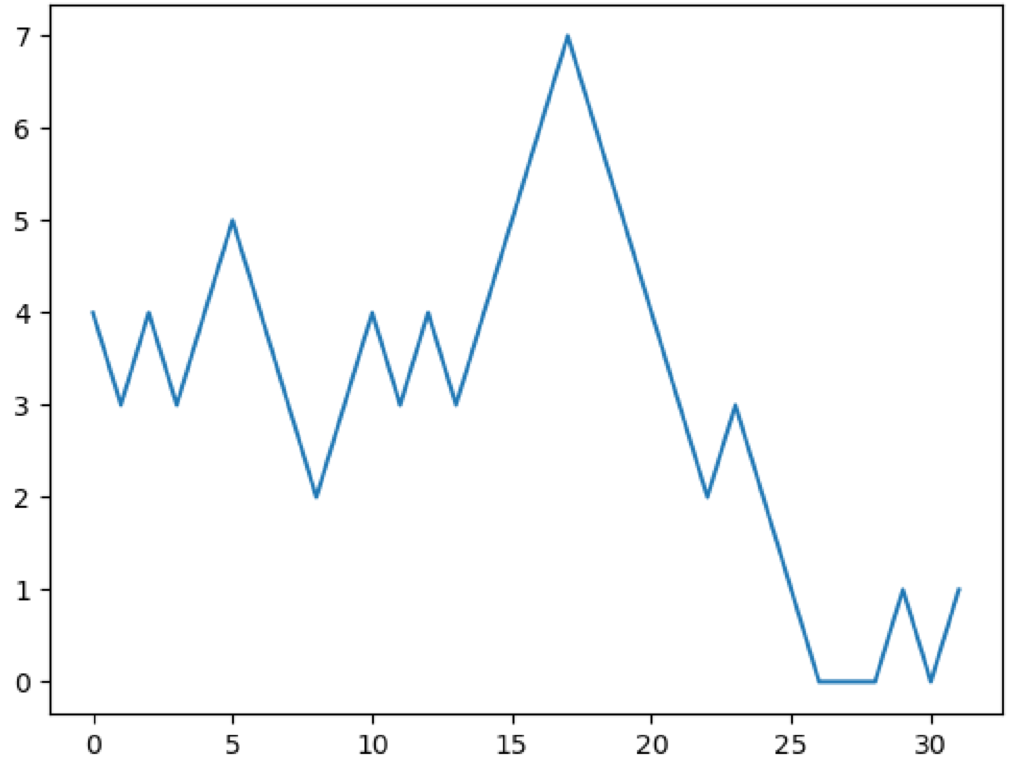
BUT: there is no winning in random solution landscapes → a random selection of $x_1, x_2, x_3, \ldots$ is on average as good as the most clever optimization algorithm





No Free Lunch Theorem

- Consequences → we must make assumptions about the solution landscape
- By far the most often used assumption: smoothness
- There is now a correlation between adjacent states
- Local Search Methods (like Tabu Search) now perform better then random search



Exercise 1

Exercise 1 (a): Important Concepts

- *Agent*: Entity that perceives and acts
- *Environment*: World in which the agent perceives and acts
- *Agent function*: (policy in RL). Function that maps perceptions to actions
- *Rationality*: Choose the action that maximizes the expected performance metric based on knowledge and the previous perception sequence
- *Autonomy*: An agent's behavior depends on its own experience and not on fixed programming

Exercise 1 (a): Important Concepts

- *Model based agent*: Agent whose actions are derived directly from an internal model of the current state of the environment
- *Goal oriented agent*: Agent that selects actions that it believes will help reach a given goal
- *Benefit oriented agent*: Agent that selects actions that it believes will maximize expected utility
- *Reflex agent*: Agent whose actions are only conditioned on the current perception

Exercise 1 (b): Important Concepts

Are reflexive actions - like pulling one's hand back from a hot stove top - rational, intelligent, or both?

- Sub-question to answer this question: what is the performance metric of the agent → Negative burning degree (maximize)

- Rational, because the action maximizes the performance metric

- Not intelligent, because the agent does not think about the action and its consequences

Exercise 1 (c): Important Concepts

Justify why it makes sense to perform the formulation of the problem <u>after</u> the formulation of the goal?

Solution:

- Goal formulation = what aspects of the world are we interested in?
- Problem formulation = how do interesting aspects need to be manipulated (we must know what the interesting aspects are)

Exercise 1 (d): Important Concepts

- *State space*: Graph where nodes represent states and edges represent actions
- *State*: Situation in which an agent finds itself
- *Search tree*: Tree of states (root = starting state)
- *Goal*: state we want to reach
- *Action*: transition between states
- *Successor function*: Given a state the successor function returns a set of state-action pairs
- *Branching factor*: maximum number of actions for a state in the search tree