1) True or False?

a) False

An agent does not need complete information about the environment in order to behave rationally. Rationality is defined relative to the information available to the agent. Even with partial perception, an agent can still choose the action that maximizes expected performance based on what it has observed and what it knows. Example: A vacuum cleaner robot may not know the exact layout of a house but can still act rationally by selecting actions that best achieve its cleaning goal given its limited sensors.

b) True

A simple reflex agent bases its decisions only on the current percept and ignores history. However, some environments require memory or state inference for rational behavior. Example: Consider a maze where two distinct locations produce identical sensor readings. The correct action depends on the previous path taken. Since a reflex agent cannot distinguish between these states based solely on the current percept, it cannot act rationally in such an environment.

c) True

Such environments can be constructed trivially. Example: Imagine an environment with only one state and a performance measure that assigns the same score to every action. Since no action can be better or worse than another, any possible agent behaves rationally in this setting.

d) False

Agent functions map percept sequences to actions. The set of possible agent functions is uncountably infinite, while the set of computer programs is only countably infinite. Therefore, there are more agent functions than programs, which means some agent functions cannot be implemented by any program. Additionally, some functions are not computable at all because they would require solving undecidable problems (e.g., variants of the Halting Problem).

e) False

Rationality does not imply guaranteed success in every single round, especially in games involving probability and hidden information. A rational agent selects actions that maximize expected utility, but outcomes can still be unfavorable due to chance. Example: A rational poker agent may go all-in with a statistically strong hand and still lose due to random card draws. Rational play maximizes long-term winnings, but it does not eliminate the possibility of short term losses.

2) Describing Environment Properties of Agents

a) Task Environment Selection

I have chosen autonomous underwater exploration as my task environment. This is done via autonomous underwater vehicles (AUVs), which are unmanned, untethered vehicles used to conduct underwater research. They are equiped with several different sensors and can be programmed to fulfill various tasks.

b) Observable

AUVs are mainly used to explore the sea floor or specific sites and has to rely on its sensors, like cameras and sonar. Because only the environment directly observed by the AUV is known, it is only partially observable.

c) Agents

In a mission is usually only a single AUV involved, at least in the immediate vicinity. It is therefore a single agent setting.

d) Deterministic Nature

The environment is stochastic as every action taken by the agent is subject to random chance. As an example could a changing current change the position of the AUV or a fish swimming past it be picked up by its sensors.

e) Episodic vs. Sequential

It is a sequential environment because every action taken by the AUV has an impact on its future actions, for example has every change in position, rotation or velocity an impact on its further navigation.

f) Dynamic Characteristic

The environment is dynamic, as the AUV has to navigate the sea, which changes regardless of the action or inaction of the vehicle.

g) Discreteness

As the environment of underwater exploration is the continuously changing sea, it is a continuous environment.

3) Agent Functions vs Agent Programs

a)

Yes, there can be more than one agent program that implements a given agent function. This is possible because we can use different data structures or algorithms to implement the same agent function. In the example of the vacuum cleaner where it cleans if there is dirt on the sqare, else moves to the next square, there are numerous ways to implement this solution. One method can be by using an if-else statement program - if the square is dirty clean, else move to the next square. And another method can be by using a switch case or a lookup table program - read the state of a square and then perfrom the action related to the state in the table.

b)

Yes, there are agent functions that cannot implemented by any agent programs. Some of them are agent functions where the decision is made based on an infinite percept history or with infinite memory consumption. Another case is when the agent function is computationally expensive that exceeds the limit of the machines. And another case is when the solution for a problem cannot be executed mathematically.

c)

The architecture stores the agent programs in the form of 0's and 1's. Therefore, the number of distinct agent programs that an n bit storage architecture can have is atmost 2^n. The reason for using atmost is that the n bit storage can also store the percept history which maybe required for some of the agent programs. Still, the number of configurations of percept and program remains the same, i.e. 2^n.

4) Two Friends

a)Detailed Formulation

i) State Space

The state space should show the current positions of the two friends.

$$S = \{(a, b) \mid a, b \in \text{Cities}\}$$

ii) Successor Function

The successor function should return all the possible states that can be reached from the current state.

$$\text{Successor}((a, b)) = \{(a', b') \mid a' \in \text{Neighbors}(a),\ b' \in \text{Neighbors}(b)\}$$

iii) Goal

The goal is that both the friends shoud be in the same city, i.e. a = b.

$$\text{Goal}((a, b)) \colon a = b$$

iv) Step-cost Function

The step cost function will be the maximum of the distance that each friend should travel in a single iteration.

$$\text{Cost}((a,b),(a',b')) = \max(d(a,a'),d(b,b'))$$

b) Heuristic Functions

A heuristic function h(f) is admissable iff,

$$h(f) \leq h'(f)$$

where h'(f) is the true optimal distance.

i) D(i, j)

In this case D(i, j) is admissable as the displacement is always less than or eqaul to the optimal distance between two cities.

$$D(i,j) \leq h'(f)$$

ii) 2.D(i, j)

In this case the heuristic function is not admissable as it can be greater than the optimal distance.

$$2.D(i,j) > h'(f)$$

iii) D(i, j)/2

In this case the heuristic function is admissable as dividing the lower value will always be lesser than the optimal value.

$$D(i,j)/2 \leq h'(f)$$

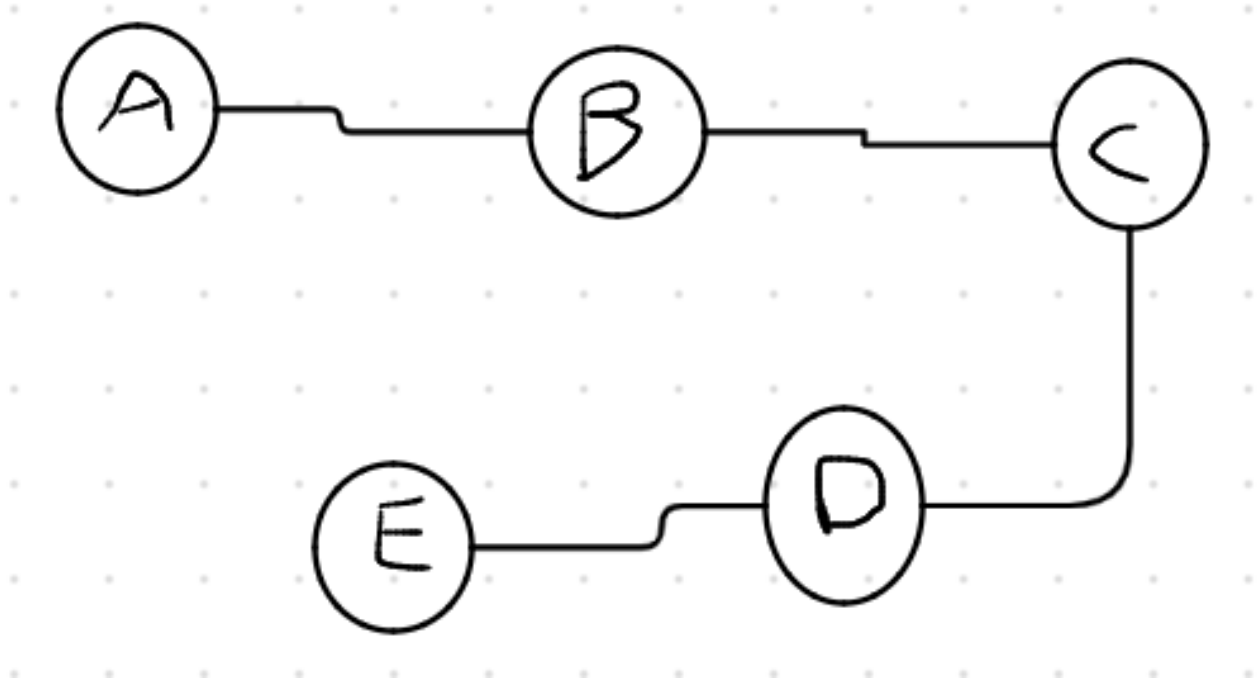
c) An example where no solution exists

An example for a completely connected map where no solutions exists is the case where there are only 2 states (say A and B). Since they start in adjacent cities and there is only one other state city that they can go to, they will never meet. Therefore, no solution exists for this problem.

d) Modification of the map for the previous promblem

The only way for the above map to have a solution, while being completely connected is by adding one more city to the map. In this way, since they always start on adjacent cities, they can always meet at the third city.

e) Map in which all solutions requireing one friend to visit the same city twice



In the above map if the friends were to start from adjacent cities (say B and C), both of them would have to visit one city twice in all solutions. But there are no maps where only one friend visits the same city twice in all solutions.

5) Binary tree traversals

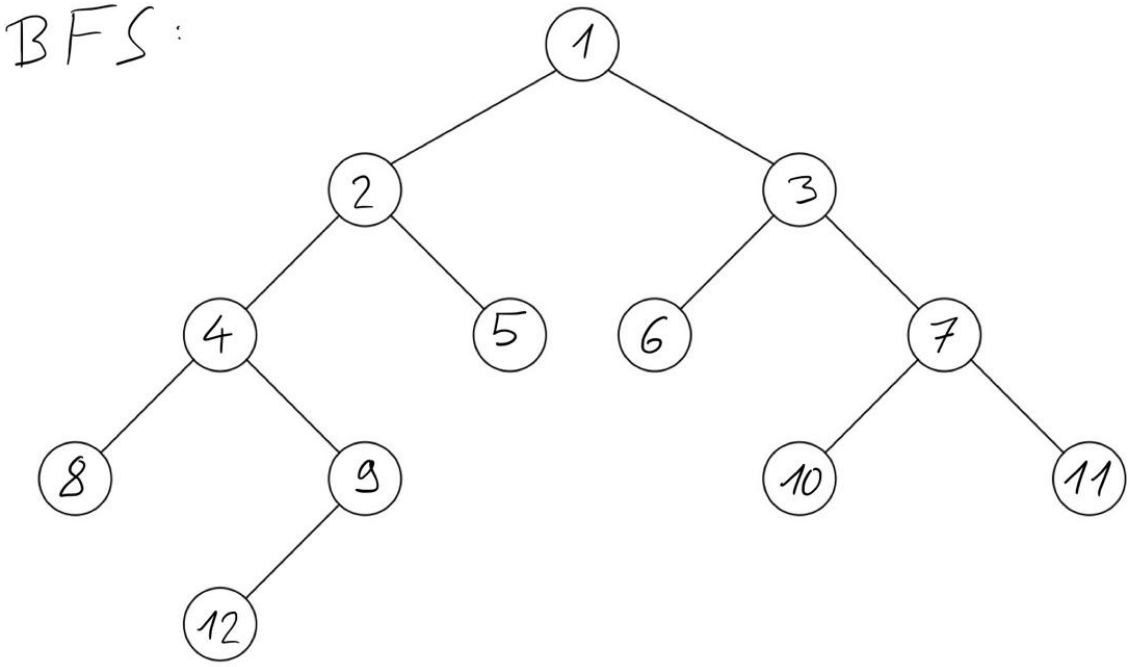


Figure 1: A binary tree.

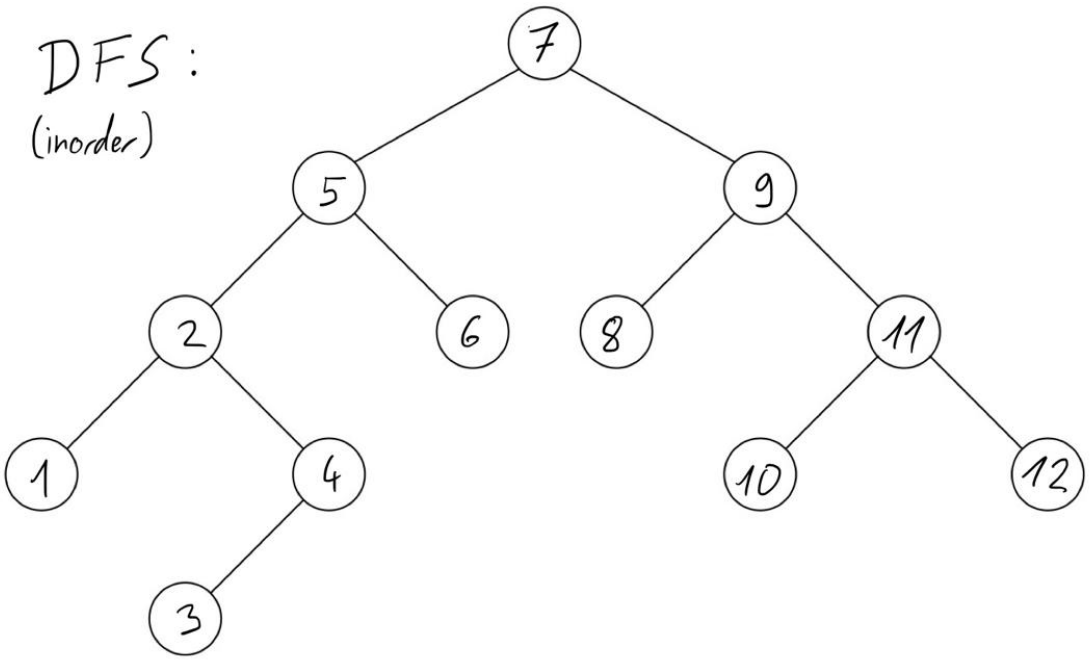


Figure 1: A binary tree.

Oliver Löhr (3311903) Bachelor Informatik
Lara Aziz (3720604) Bachelor Informatik
Joel Thomas (3814387) Master Computer Science

6) Graph traversals

a) BFS

BFS : 1. Schritt Q = 0

Queue Q 3,5,7

FIFO 5,7,2

7,2,6

2,6

6,1,4

1,4

4

–

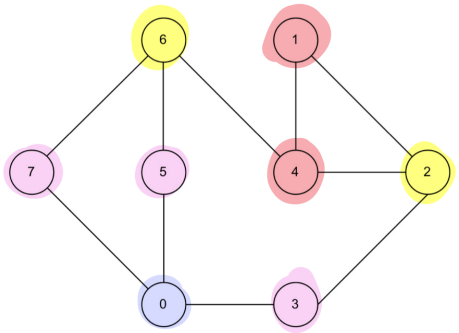


Figure 2: A 2-dimensional graph. Apply BFS and DFS starting at node 0.

b) DFS - lowest value first

DFS S = 0 3 2 1 4 6 5 7

Stack S

LIFO

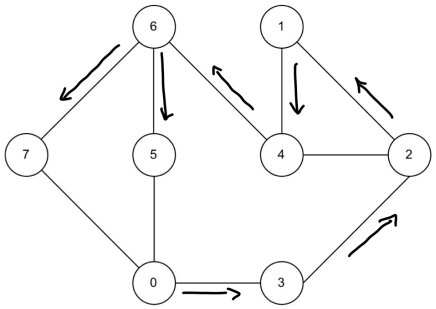


Figure 2: A 2-dimensional graph. Apply BFS and DFS starting at node 0.

b) DFS - highest value first

DFS S = 0 7 6 5 4 2 3 1

Stack S

LIFO

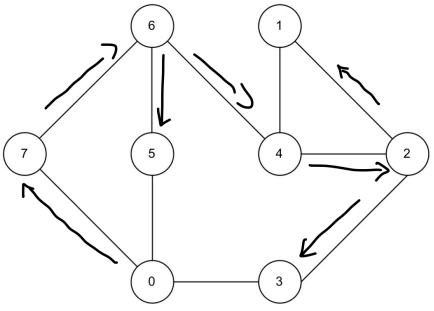


Figure 2: A 2-dimensional graph. Apply BFS and DFS starting at node 0.