Exercise 1.

a) **1i** Initialize the labeling l, the equality graph G_l and the matching M.

Vertex	u_1	u_2	u_3	u_4	u_5	v_1	v_2	v_3	v_4	v_5
Labeling l	9	7	7	9	5	0	0	0	0	0

$$G_l = (U \cup V, \{(u_1, v_5), (u_2, v_1), (u_3, v_2), (u_4, v_5), (u_5, v_3)\})$$

$$M = \emptyset$$

2i M is not perfect, therefore set $S = \{u_1\}$ and $T = \emptyset$.

3i $N_l(S) \neq T$.

4i $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_5$. b is free, we therefore augment M and get $M = \{(u_1, v_5)\}$.

2ii M is not perfect, therefore set $S = \{u_2\}$ and $T = \emptyset$.

3ii $N_l(S) \neq T$.

4ii $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_1$. b is free, we therefore augment M and get $M = \{(u_1, v_5), (u_2, v_1)\}$.

2iii M is not perfect, therefore set $S = \{u_3\}$ and $T = \emptyset$.

3iii $N_l(S) \neq T$.

4iii $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_2$. b is free, we therefore augment M and get $M = \{(u_1, v_5), (u_2, v_1), (u_3, v_2)\}$.

2iv M is not perfect, therefore set $S = \{u_4\}$ and $T = \emptyset$.

3iv $N_l(S) \neq T$.

4iv $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_5$. b is not free, we therefore set $S = \{u_4, u_1\}$, $T = \{v_5\}$.

3v $N_l(S) = T$, therefore we improve l.

$$\lambda = \min\{l(a) + l(b) - w(a, b) \mid a \in S, b \in V - T\} = \min\{3, 6, 4, 8, 1, 2, 5, 3\} = 1$$

Update l

Vertex	u_1	u_2	u_3	u_4	u_5	v_1	v_2	v_3	v_4	v_5
Labeling l	8	7	7	8	5	0	0	0	0	1

Update G_l

$$G_l = (U \cup V, \{(u_1, v_5), (u_2, v_1), (u_3, v_2), (u_4, v_1), (u_4, v_5), (u_5, v_3)\})$$

4v $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_1$. b is not free, we therefore set $S = \{u_4, u_1, u_2\}$ and $T = \{v_5, v_1\}$.

3vi $N_l(S) = T$, therefore we improve l.

$$\lambda = \min\{l(a)+l(b)-w(a, b) \mid a \in S, b \in V-T\} = \min\{5, 3, 7, 3, 4, 2, 1, 4, 2\} = 1$$

Update l

Vertex	u_1	u_2	u_3	u_4	u_5	v_1	v_2	v_3	v_4	v_5
Labeling l	7	6	7	7	5	1	0	0	0	2

Update G_l

$$G_l = (U \cup V, \{(u_1, v_5), (u_2, v_1), (u_3, v_2), (u_4, v_1), (u_4, v_2), (u_4, v_5), (u_5, v_3)\})$$

4v $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_2$. b is not free, we therefore set $S = \{u_4, u_1, u_2, u_3\}$ and $T = \{v_5, v_1, v_2\}$.

3vii $N_l(S) = T$, therefore we improve l.

$$\lambda = \min\{l(a) + l(b) - w(a, b) \mid a \in S, b \in V - T\} = \min\{2, 6, 3, 1, 4, 5, 3, 1\} = 1$$

Update l

Vertex	u_1	u_2	u_3	u_4	u_5	v_1	v_2	v_3	v_4	v_5
Labeling l	6	5	6	6	5	2	1	0	0	3

Update G_l

$$G_l = (U \cup V, \{(u_1, v_5), (u_2, v_1), (u_2, v_4), (u_3, v_2), (u_4, v_1), (u_4, v_2), (u_4, v_4), (u_4, v_5), (u_5, v_3)\})$$

4vii $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_4$. b is free, we therefore augment M and get $M = \{(u_1, v_5), (u_2, v_1), (u_3, v_2), (u_4, v_4)\}$.

2viii M is not perfect, therefore set $S = \{u_4\}$ and $T = \emptyset$.

3viii $N_l(S) \neq T$.

4viii $N_l(S) \neq T$, therefore we choose $b \in N_l(S) - T = v_3$. b is free, we therefore augment M and get $M = \{(u_1, v_5), (u_2, v_1), (u_3, v_2), (u_4, v_4), (u_5, v_3)\}$.

2ix M is perfect. Terminate.

b) Consider the following weighted, (non-complete) bipartite graph.

$$G = (\{u_1, u_2, v_1, v_2\}, \{(u_1, v_1), (u_2, v_1), (u_2, v_2)\})$$

with the weights given by the following table

	u_1	u_2
v_1	1	3
v_2	-	1

In this graph, the maximum weight maximum matching is $M = \{(u_1, v_1), (u_2, v_2)\}$, but the maximum weighted matching is $M = \{(u_2, v_1)\}$.

Exercise 2.

a) *Proof.* Since each vertex in S needs to be incident to an odd number of edges, it needs to be incident to at least one edge in M. To find the optimal S-join, having more than one incident edge on a vertex of S is therefore not optimal, since this adds additional weights. From this we can conclude that we can construct an optimal S-join from $|S|/2$ (note that $|S|$ is always even) paths that each start an end in a vertex of S.

To show why these paths need to be edge-disjoint, we consider an example where the paths are *not* edge-disjoint. Consider an odd number k paths starting and ending in a vertex of S that all include an edge $(t_1, t_2), t_1, t_2 \in V \setminus S$. In all other edges, they are disjoint. (Note that is must be an odd number of paths, since otherwise there would be an odd number of incident edges in M for t_1 and t_2.) We now consider the cost over these k paths. For each of the paths, we can divide the cost into the following three parts:

$$c_i = c_i^{(l)} + c((t_1, t_2)) + c_i^{(r)}, \quad \forall i \in \{1, \dots, k\}$$

Where $c_i^{(l)}$ denotes the cost from the end of the path i which is closer to t_1 to t_1 and $c_i^{(r)}$ the cost from the other end to t_2. Here we can see that each path does include the cost of the edge (t_1, t_2). However, we can reduce the total cost by taken paths 1 to $k-1$ and combining pairs of two path segments that are either left of the shared edge or right of it. By modifying the paths in that way, we do not change the number of paths, but only one of the paths (the kth) does still include the edge (t_1, t_2). The total cost still include the cost for all the left and right segments, but the cost for the shared edge is only included once and not k times. Since we assumed that (t_1, t_2) was the only shared edge, the paths are now edge-disjoint.

If the paths share more than one edge, the same reasoning applies. The paths can be split and rejoined multiple times until they are fully edge-disjoint. □

b) To find an optimal S-join on G, we can use the following algorithm:

Algorithm 1: Find an optimal S-join on G with the help of a matching algorithm.

```
Result: M is an optimal S-join on G.
M = ∅;
G' = (S, E' = ∅);
c' : E' → R⁺;
foreach s ∈ S do
    d[s], π[s] = shortestPath(G, c, s);
    foreach r ∈ S \ {s} do
        E' = E' ∪ {(s, r)};
        c'((s, r)) = d[s][r];
    end
end
M' = minWeightMaxMatch(G', c');
foreach (s, r) ∈ M' do
    M = M ∪ π[s][r];
end
```

The idea behind this algorithm is to find and optimal matching over the shortest paths between the vertices in S. Thereby SHORTESTPATH computes the shortest paths to all vertices, starting from s and returns the length of these paths, as well as the path itself. MINWEIGHTMAXMATCHING computes the minimum weight maximum matching. Asuming both of these subroutines run in polynomial time, the entire algorithm runs in polynomial time.

c) To solve the garden walk problem, we can use the following algorithm:

Algorithm 2: An algorithm to solve the garden walk problem

Result: Find a path W that solves the garden walk problem

1 $S = \{x \in J \mid \text{degree of } J \text{ is odd}\}$;
2 $M = \text{OPTIMALSJOIN}(G, S, l)$;
3 $G' = (J, P \cup M)$;
4 $W = \text{FINDEUCLIDIANPATH}(G', e)$;

First we double all the paths that form a optimal S-join over G with respect to all vertices with uneven degree. By that we ensure that all vertices have even degree with adding the minimum amount of paths (regarding their length) to the graph. On this modified graph we can now search for an euclidian path starting at e.

For finding the optimal S-join we can use the algorithm from b), which we showed to run in polynomial time. To find the euclidian path, we can use Hierholzer's algorithm[1] which runs in linear time. Therefore the entire algorithm runs in polynomial time.

[1] https://en.wikipedia.org/wiki/Eulerian_path#Hierholzer's_algorithm

Exercise 3.

As a first step in our algorithm to find a maximum product maximum matching, we construct a modified graph G' with the same vertices and edges, but all the weights are the logarithm of the original weights.

$$G' = (V \cup W, E = W \times W)$$

$$c'(e) = \log(c(e)),\ \forall e \in E$$

Now we call the hungarian algorithm to find a maximum weight maximum matching on G'. This maximum weight maximum matching is now also a maximum product maximum matching on G.

Proof. M is a maximum weight maximum matching on G', therefore the following is maximum:

$$\sum_{e \in M} c'(e) = \sum_{e \in M} \log(c(e))$$

Since $\log(a) + \log(b) = \log(ab)$, we can rewrite this to the following:

$$\sum_{e \in M} \log(c(e)) = \log\Big(\prod_{e \in M} c(e)\Big)$$

Since the logarithm grows monotonically with it's argument, we know that if $log(a)$ is maximum, a is also maximum. Therefore $\prod_{e \in M} c(e)$ is maximized and M is a maximum product maximum matching. □

Exercise 4.

As a first step, we rewrite the objective function as a maximization and turn all the constraints into upper bounds:

$$\begin{array}{rrrrrrrrr}
\text{maximize} & -3 & x_1 & +4 & x_2 & -9 & x_3 & & \\
\text{subject to} & -3 & x_1 & + & x_2 & - & x_3 & \leq & 7 \\
& -6 & x_1 & +5 & x_2 & -5 & x_3 & \leq & 13 \\
& 6 & x_1 & -5 & x_2 & +5 & x_3 & \leq & -13 \\
& & & x_1 \leq & 0, & 2 \leq & x_2 & \leq & 8
\end{array}$$

As a second step we set $x_3 = x_3' - x_3''$ and we rewrite the bounding of x_2 as regular constraints:

$$\begin{array}{rrrrrrrrrrr}
\text{maximize} & -3 & x_1 & +4 & x_2 & -9 & x_3' & +9 & x_3'' & & \\
\text{subject to} & -3 & x_1 & + & x_2 & - & x_3' & + & x_3'' & \leq & 7 \\
& -6 & x_1 & +5 & x_2 & -5 & x_3' & +5 & x_3'' & \leq & 13 \\
& 6 & x_1 & -5 & x_2 & +5 & x_3' & -5 & x_3'' & \leq & -13 \\
& & & & x_2 & & & & & \leq & 8 \\
& & & - & x_2 & & & & & \leq & -2 \\
& & x_1, & & x_2, & & x_3', & & x_3'' & \geq & 0
\end{array}$$

This is the final, equivalent linear program in standard form.