Graphs and Algorithms

Exercise 1

(a) \(\theta\alpha\) is bijective since it is the composition of two bijective maps. We have

\[
\{u, v\} \in E(G) \iff \{\alpha(u), \alpha(v)\} \in E(G) \iff \{\theta(\alpha(u)), \theta(\alpha(v))\} \in E(H)
\]

because (i) \(\alpha\) is an automorphism, and (ii) \(\theta\) is an isomorphism. Taking both equivalences together, \(\theta\alpha\) is an isomorphism.

(b) Let \(I\) be the set of all isomorphisms from \(G\) to another graph. (a) proves that \(\theta \text{Aut}(G) \subseteq I\).

To show \(I \subseteq \theta \text{Aut}(G)\), take any \(\phi \in I\). Then \(\theta^{-1}\phi\) is an automorphism on \(G\): clearly it exists and is bijective. Similar to (a) we have

\[
\{u, v\} \in E(G) \iff \{\phi(u), \phi(v)\} \in E(H) \iff \{\theta^{-1}\phi(u), \theta^{-1}\phi(v)\} \in E(G).
\]

But this means \(\theta^{-1}\phi \in \text{Aut}(G)\), hence \(\phi \in \theta \text{Aut}(G)\).

(c) Let \(P\) be the set of permutations of \(V(G)\). Thus \(|P| = n!\). Let \(\sim\) be the equivalence relation given by \(\pi \sim \tau\) iff \(\pi^{-1}\tau\) is an automorphism on \(G\). Then the equivalence class \([\text{Id}]\) in \(P/\sim\) of the identity permutation \(\text{Id}\) is exactly \(\text{Aut}(G)\), and because of (b) each equivalence class is the set of isomorphisms from \(G\) to some \(H\), and in particular all classes are of order \(|\text{Aut}(G)|\). Thus we have

\[
n! = |\text{Aut}(G)| \cdot |G/\sim| = |\text{Aut}(G)| \cdot \{H \text{ isomorphic to } G\}
\]

as desired.

Exercise 2

(a) The number of labelled trees with a prescribed degree sequence \(d_1, \ldots, d_n\) is

\[
\frac{(n - 2)!}{\prod_{i=1}^{n}(d_i - 1)!}
\]

This can be seen as follows. First observe that each vertex \(i\) appears \(d_T(i) - 1\) times in the Prüfer code of a tree \(T\): A non-leaf vertex \(v\) appears one time for each deleted neighbor. When \(v\) has only one remaining
neighbor, it is either deleted or remains with the last edge. Leaves are not recorded at all.

To count trees with prescribed degree sequence \(d_1, \ldots, d_n\) we can therefore count lists of length \(n - 2\) that contain \(d_i - 1\) entries of vertex \(i\) for each \(i = 1, \ldots, n\). For every vertex \(i\) we make its occurrences in the list distinguishable by introducing subscripts to each occurrence. Then we are permuting \(n - 2\) distinct objects, namely \(\{1, \ldots, d_i - 1\}, \ldots, 1, \ldots, n, \ldots, n_{d_n - 1}\}, \) and there are \((n - 2)!\) lists with these objects. Since every entry \(i_j\) in the list refers to the same vertex \(i\), we counted each desired arrangement \(\prod_{i=1}^{n}(d_i - 1)!\) times, once for each permutation of the subscripts on each vertex.

(b) There are

\[
\binom{n}{3} \cdot (n - 3) \cdot \frac{(n - 2)!}{2}
\]

labelled trees with three leaves. This can be seen as follows. Each such tree corresponds to a Prüfer code in which exactly three vertices do not appear, and hence exactly one vertex appears twice. This shows that in each such tree there are three vertices of degree 1, one vertex of degree 3, and all remaining vertices have degree 2. There are \(\binom{n}{3} \cdot \binom{n-3}{1}\) degree sequences with this property. And by (a) each such sequence gives us \(\frac{(n-2)!}{2}\) trees.

(c) Obviously, the number of trees on \(n\) vertices that contain the edge \(e\) is the same for every fixed edge \(e \in \binom{[n]}{2}\). (The choice of the edge \(\{3, 4\}\) has no influence on the result. Asking for the number of trees containing the edge \(\{1, 4\}\) yields the same number.) Combining that

- there are in total \(n^{n-2}\) trees on \(n\) vertices by Cayley’s formula,
- there are in total \(\binom{n}{2}\) edges on \(n\) vertices, and
- every tree has \(n - 1\) edges

we obtain that the number of trees containing any fixed edge \(e\) is

\[
\frac{n-1}{\binom{n}{2}} \cdot n^{n-2} = \frac{2}{n} \cdot n^{n-2} = 2n^{n-3}.
\]

(We tacitly imply that \(n \geq 4\). Otherwise there are clearly no trees containing the edge \(\{3, 4\}\).)

Exercise 3

We can adapt the DFS algorithm of the lecture to compute the biconnected components (blocks) of a graph as demonstrated below. By computing the blocks we mean enumerating the vertices of each block (other interpretations are possible). Note that it is not sufficient to compute the articulation nodes of the graph only, because even if we know that a vertex \(v\) is an articulation node, from this information alone it is not clear which neighbors of \(v\) belong to the same block.
The main idea of the extended algorithm is the following statement:

**Lemma.** Let \( u \) be an articulation node and \( v \) a child of \( u \) satisfying \( d[u] \leq \text{low}[v] \). Then the block of \( G \) containing \( v \) consists of \( u \) and all descendants of \( v \) that are not descendants of some articulation point \( x \) with \( d[x] > d[u] \).

We use a stack \( S \) on which we store all vertices in the order of encounter. The variable \( bc \) (block counter) counts the number of blocks. We detect an articulation node \( u \) by checking the condition \( d[u] \leq \text{low}[v] \) when backtracking from a child \( v \) of \( u \) in the DFS-tree. In this case we empty the stack up to \( v \), and store all those vertices together with \( u \) as a list \( \text{blocks}[bc] \). The lemma above shows that at this moment the stack contains all vertices belonging to the block containing \( v \). Note that \( u \) may appear as an articulation node several times for different children.

**Find-Blocks** \((G, s)\)

1. \( \forall v \in V(G) : d[v] \leftarrow 0, \text{pred}[v] \leftarrow \text{NIL} \)
2. \( \text{time} \leftarrow 0 \)
3. \( bc \leftarrow 1 \)
4. \( S \leftarrow \text{new} \text{Stack} \)
5. \( \text{DFS-Visit}(s) \)

**DFS-Visit** \((u)\)

1. \( \text{time} \leftarrow \text{time} + 1 \)
2. \( d[u] \leftarrow \text{time} \)
3. \( \text{low}[u] \leftarrow d[u] \)
4. \( \text{for each } v \in \Gamma(u) \setminus \{\text{pred}[u]\} \)
5. \( \text{do if } d[v] = 0 \quad // \ v \text{ has never been visited before} \)
6. \( \quad \text{then } \text{pred}[v] \leftarrow u \)
7. \( \quad \text{put } v \text{ on stack } S \)
8. \( \quad \text{DFS-Visit}(v) \)
9. \( \quad \text{if } d[u] \leq \text{low}[v] \quad // \ u \text{ is an articulation node or the root} \)
10. \( \quad \text{then } \text{blocks}[bc] \leftarrow \emptyset \)
11. \( \quad \text{repeat} \)
12. \( \quad \text{take topmost vertex } w \text{ off stack } S \)
13. \( \quad \text{blocks}[bc] \leftarrow \text{blocks}[bc] \cup \{w\} \)
14. \( \quad bc \leftarrow bc + 1 \)
15. \( \quad \text{until } w = v \)
16. \( \quad \text{blocks}[bc] \leftarrow \text{blocks}[bc] \cup \{u\} \)
17. \( \quad \text{if } \text{low}[u] > \text{low}[v] \)
18. \( \quad \text{then } \text{low}[u] \leftarrow \text{low}[v] \)
19. \( \quad \text{else } // \ v \text{ has been visited before} \)
20. \( \quad \text{if } \text{low}[u] > d[v] \)
21. \( \quad \text{then } \text{low}[u] \leftarrow d[v] \)

**Proof of the Lemma.** Let \( u \) be an articulation node and \( v \) a child of \( u \) satisfying \( d[u] \leq \text{low}[v] \). First, let \( w \) be a descendant of \( v \) that is not a descendant of some articulation point \( x \) with \( d[x] > d[u] \). Then we know that the vertices on the
path from $v$ to $w$ in the DFS-tree, except for $w$, are not articulation nodes. This shows that $w$ and $v$ are in the same block of $G$.

Now, let $w$ be a vertex in the block containing $v$. Because $d[u] \leq \text{low}[v]$, the block containing $u$ and $v$ is not just the edge $\{u, v\}$. Hence, $w$ and $v$ lie on a common cycle in $G$. Because of this, $w$ cannot lie in a different branch of the DFS-tree than $v$ and $u$ (There are no cross-edges!). Also, $w$ cannot be a predecessor of $u$ as otherwise there would be a back edge from a descendant of $v$ to a predecessor of $u$, contradicting $d[u] \leq \text{low}[v]$. This shows that $w$ has to be a descendant of $u$.

Even more, $w$ has to be a descendant of $v$ because there are no cross edges between different branches of the subtree rooted at $u$. If any vertex on the path connecting $v$ and $w$ was an articulation point, $v$ and $w$ would not be in the same block.

\qed