Solutions to exercises of lecture 7

Exercise 7.1. Show how to build a tree decomposition of any tree of width 1. How would you win the cops and robber game if you control the two cops?

Solution: Call the given tree S, and root the tree at an arbitrarily picked vertex r of degree 1. Let the edges of the tree be $e_1, \ldots, e_m$. Create a tree decomposition (X, T) where $X = \{e_1, \ldots, e_m\}$ (note an edge is to be interpreted as the set of the 2 incident vertices here), and T is a tree on vertex set $e_1, \ldots, e_m$ where e_i and e_j are adjacent if $e_i \cap e_j = \{v\}$ and e_i is the edge between v and the parent of v. Note that T is connected since all vertices e_i of T are connected to the edge incident to the root of S. Also, T has $m = n - 1$ vertices since it has a vertex for every edge of S and $n - 2$ edges since it has an edge for every child-parent pair of T not including the root, so it must be a tree. For the first and second property in the definition of tree decomposition, this trivially covers all vertices and edges. See also the example at slide 5 of L7.

To win the cops and robber game, pick a root r of the given tree S arbitrarily. Observe for which child c of r the robber is in the tree $T[c]$ rooted at c. Place the second cop on this vertex c and iterate. After one iteration, we know for sure the robber is in $T[c]$. After some number of iterations, $T[c]$ has height 1 and placing the cop will lead to capture.

Exercise 7.2. Explain why every n-vertex graph has treewidth at most $n - 1$, can you find an n-vertex graph with treewidth $n - 1$?

Solution: Use one bag V. The unique graph on one vertex is a tree and all vertices are in V and edges are in $V \times V$.

Exercise 7.3. Show that removing a vertex does not increase the treewidth.

Solution: Let $X = \{X_1, \ldots, X_\ell\}$ and (X, T) be a tree decomposition. Let $X' = \{X'_1, \ldots, X'_m\}$ where $X'_i = X_i \setminus v$ then (X', T') (where T' is the same as T except that the vertex set is renamed) also a tree decomposition since the properties only became less strict.

Exercise 7.4. Give an algorithm that given a graph G and tree decomposition of G of width at most w, determines whether G is 3-colorable in $O^*(3^w)$ time.

Solution: First transform the given tree decomposition into a nice tree decomposition (X, T) where $X = \{X_1, \dots, X_l\}$ of same width in polynomial time as discussed in the lecture. Then, for a bag $i = 1, \dots, \ell$, and $c : X_i \to \{1, 2, 3\}$ define $A[i, c]$ to be true if there exists a 3-coloring c' of G_i such that $c(v) = c'(v)$ for every $v \in X_i$.

- **Leaf bag**:
$$A[i, c] = \textbf{true}, \text{ for every } c : \{v\} \to \{1, 2, 3\}$$

- **Introduce vertex bag**:
$$\begin{aligned} A[i, c] &= A[j, c'], && \text{if } \forall w \in N(v) \cap X_i : c(v) \neq c(w), \\ A[i, c] &= \textbf{false}, && \text{otherwise,} \end{aligned}$$
here c' denotes c restricted to X_j, i.e. the function $c' : X_j \to \{1, 2, 3\}$ such that $c'(u) = c(u)$ for every $u \in X_j$. We need to check whether the vertex introduced does get a valid color at this point.

- **Forget bag**:
$$A[i, c] = A[i, c_1] \vee A[i, c_2] \vee A[i, c_3]$$
here c_z denotes c extended to X_i, i.e. the function $c' : X_j \to \{1, 2, 3\}$ such that $c_z(u) = c(u)$ for every $u \in X_i$ and $c(v) = z$. We make a decision of the color the forgotten vertex v receives here.

- **Join bag**:
$$A[i, c] = A[j, c] + A[j', c]$$
The coloring needs to be consistent in both independent subproblems.

Exercise 7.5. Show that any complete bipartite graph where both parts have n vertices has treewidth at most n.

Solution: We can use a strategy to win the cops and robber game with $n+1$ robbers. To do this, simply put n robbers on one side of the graph, and place the last robber on the vertex on which the robber resides (which cannot move since all neighbors of his vertex are occupied by cops).

Exercise 7.6. Show that the $l \times l$-grid has treewidth at most l.

Solution: We can use a strategy to win the cops and robber game with $l + 1$ robbers. Let the vertices of the grid by denoted by v_{ij} in the natural way. First place robbers on v_{i1} for every j (i.e. the 'first column'). Then for $j = 1, \dots, l$ add a cop to $v_{j,2}$ and remove the cop from $v_{j,1}$ until all vertices $v_{i,2}$ are occupied by cops. Continue this process until all vertices v_{il} are occupied by cops. We see that at any iteration the cops separate the later columns from the earlier columns so the robber cannot reach the earlier columns and will be captured eventually.

Exercise 7.7. A graph is called outerplanar if it can be drawn in the plane such that all vertices are on the outerface. Show there is a constant c such that that every outerplanar graph has treewidth at most c.

Solution: One strategy is to use Theorem 7.3: add a vertex v that is adjacent to all vertices. This graph is still planar since if v is placed on the outerface all its incident edges can be drawn without crossings. A spanning tree from v of height 1 exists and we obtain that the treewidth is at most 2.

Exercise 7.8. Show that every graph has treewidth at most $O(\sqrt{n})$, conclude that 3-coloring, dominating set and independent set can be solved in time $O^*(2^{O(\sqrt{n})})$ on planar graphs.

Solution: This follows directly by the grid minor theorem since an n vertex graph cannot have a $((n+1) \times (n+1))$-grid as a minor, so it must have treewidth $O(\sqrt{n})$. The theorem then gives us a tree decomposition of width $O(\sqrt{n})$ and by applying the $O^*(2^{O(w)})$-time algorithms for the mentioned problems we obtain $O^*(2^{O(\sqrt{n})})$ time algorithms.

Exercise 7.9. Show that every graph has treewidth at most $O(\sqrt{n})$ using Baker's layering approach combined with Theorem 7.3.

Solution: Pick a vertex s arbitrarily and perform a BFS from s. Partition the graph into layers $L_1, \dots, L_h$. If $h \leq \sqrt{n}$ we are done by Theorem 7.3 since the BFS tree has height at most h. Otherwise, since $L_1, \dots, L_h$ partition V, there must exists some $l \leq \sqrt{n}$ such that $|V_l| \leq \sqrt{n}$, where $V_l = \bigcup_{j \equiv l (\mod \sqrt{n})} L_j$. Also, as in the lecture, we have that the treewidth of $G[V \setminus V_l]$ is at most $O(\sqrt{n})$ by Theorem 7.3. Let (X, T) be a tree decomposition of $G[V \setminus V_l]$ of width $O(\sqrt{n})$. Then (X', T) where $X' = \{X'_1, \dots, X'_\ell\}$ and $X'_i = X_i \cup V_i$ also is a tree decomposition, and it has width at most $O(\sqrt{n}) + |\sqrt{n}| = O(\sqrt{n})$.

Exercise 7.10. In the minor checking problem we are given graphs G and H and need to determine whether H is a minor of G. Why is this an NP-hard problem (hint: show it reduces to a famous NP-hard problem by fixing H to be a specific n-vertex graph)?

Solution: If H is a cycle on n vertices this reduces to the Hamiltonian cycle problem.

Exercise 7.11. In this exercise we are going to find an $O^*(2^{O(\sqrt{k})})$-time algorithm that given a planar graph G and integer k, determines whether G has a vertex cover of size k.

1. Show that in the $(l \times l)$-grid, the minimum vertex cover is of size $\Omega(l^2)$.
2. Show that if H is a minor of G and G has a vertex cover of size k, then so does H.
3. Show that given a graph G and tree decomposition of G of width w, we can find the minimum vertex cover in $O^*(2^w)$ time.
4. Use the grid minor theorem to give an $O^*(2^{O(\sqrt{k})})$-time algorithm for the mentioned problem.

Solution:

1. There are at least $l(l-1)$ edges and any vertex can cover at most 4, so a vertex cover has size at least $l(l-1)/4 \geq l^2/8$.

2. If vertices or edges are removed, there are only fewer edges to cover. If an edge is contracted, keep the new vertex in the vertex cover if one of the original vertices were. This does not increase the number of vertices in the vertex cover and still covers all edges.

3. We compute a nice tree decomposition and use dynamic programming to compute $A[i, Y]$ for all $Y \subseteq X_i$ which we define as the minimum size vertex cover W of G_i such that $W \cap X_i = Y$, which might be intstructive to try yourself. However, can also simply use the algorithm of the lecture to compute the maximum independent set as the minimum size of a vertex cover equals the number of vertices minus the maximum size of an independent set.

4. Apply the grid minor theorem with $l = 3\sqrt{k}$. If we find a $(l \times l)$-grid as a minor, we may safely return **false** since by the second point the graph has no vertex cover of size at most $9k/8$. Otherwise, we find a tree decomposition of width $27l$ and can use dynamic programming as observed in point three to solve the instance in $O^*(2^{27\sqrt{k}})$ time.

Exercise 7.12. Use the same approach as in the previous exercise to solve k-path in planar graphs in time $O^*(\sqrt{k}^{O(\sqrt{k})})$. You may use as a blackbox an algorithm that given a graph G, integer k and tree decomposition of G of width w, determines whether G has a k-path in $O^*(w^w)$ time.

Solution: It is easy to see that in an $(l \times l)$-grid there is a simple path on l^2 vertices by visiting all columns consecutively. Moreover, if H is a minor of G and H contains a k-path than G also contains a k-path: adding vertices or edges does not disturb the k-path and if H is obtained from G by contracting (u, w), the path is still in G if both edges in the k-path incident to the vertex (uw) contain a neighbor of u or of v, and otherwise, we can obtain a k-path in G from the k-path in H by adding the edge u, w. Now we may use the grid minor theorem again, with $l = \sqrt{k}$: if G contains a $(\sqrt{k} \times \sqrt{k})$-grid we may safely return **yes** and otherwise the grid minor theorem gives a tree decomposition of width at most $9\sqrt{k}$ at we may use the $O^*(w^w)$ time algorithm to solve the instance in $O((9\sqrt{k})^{9\sqrt{k}})$ time.

Exercise 7.13. Give an $O^*(c^k)$-time algorithm that takes as input a planar graph G and integer k and determines whether G has a dominating set of size at most k, for some c. Hint: perform a BFS from an arbitrary node, conclude that G has no dominating set if the BFS tree is too high.

Solution: If the BFS tree is of height at least $3k+1$, we see that we need at least one vertex for every 3 levels so we may safely return **false**. Otherwise, Theorem 7.3 gives a tree decomposition of width at most $O(k)$ and the $O^*(9^w)$ algorithm as mentioned in the lecture runs in time $O^*(2^{O(k)})$ which is $O^*(c^k)$ for some c.

Exercise 7.14. In the following game Alice and Bob need to find out whether an undirected graph G has an Hamiltonian cycle. Unfortunately, they both have different parts from the graph and communication between the two is costly.

Suppose $G = (V, E)$, $S \subseteq V$ and $G[S]$ two connected components $A, B \subseteq V$. Suppose Alice knows only $G[A \cup S]$ and Bob knows only $G[B \cup S]$, how much bits of information does Alice need to send to Bob so Bob can figure out whether G has a Hamiltonian cycle (given an infinite amount of time) if $|S| = 4, 5$? Can you find a sufficient number of bits for any $|S|$?

For $|S| = 0, 1$ the answer is always no since the graph is disconnected or 1-connected. For $|S| = 2$ Alice only needs to send one bit indicating whether there is an Hamiltonian path of $G[A \cup S]$ with endpoints in S.

Solution: Alice would like to send which kind of partial solutions here subgraph supports, the important characteristics of a partial solution (which is a set of paths with endpoint in S here) are the degrees of each vertex in S, and for the vertices of degree 1 a matching indicating which vertex is connected to which one (which is important as illustrated in the slides).

For $|S| = 4$ there could be

1. 4 vertices of degree 1, and there are 3 perfect matchings on 4 vertices: 3 types,
2. 2 vertices of degree 1, and both other vertices can have either degree 0 or 2: $\binom{4}{2}2^2 = 24$ types,

and Alice can send for every of these 27 possibilities whether in her graph there is a partial solution of that type.

For $|S| = 5$ there could be

1. 4 vertices of degree 1, and there are 3 perfect matchings on 4 vertices and the remaining vertex could have degree 1 or 2: $5 \cdot 3 \cdot 2$ types,
2. 2 vertices of degree 1, and all other vertices can have either degree 0 or 2: $\binom{5}{2}2^3 = 80$ types,

and Alice can send for every of these 110 possibilities whether in her graph there is a partial solution of that type.

In general Alice can do with at most $\sum_{i=2}^{|S|} \binom{|S|}{i} i^i$ bits (where i^i is a rough upper bound on the number of perfect matchings on i vertices).

Exercise 7.15. Give an algorithm that given a graph G and tree decomposition of G of width at most w, finds the minimum weight dominating set in $O^*(9^w)$ time.

Solution: Assume like usual there is a weight function $\omega : V \to \mathbb{N}$ and denote $\omega(X) = \sum_{v \in X} \omega(v)$. First transform the given tree decomposition into a nice tree decomposition (X, T) where $X = \{X_1, \ldots, X_l\}$ of same width in polynomial time as discussed in the lecture. Then, for a bag $i = 1, \ldots, \ell$, and a partition I, E, D of X_i define $A[i, I, E, D]$ to be the $\omega(W)$ over all $W \subseteq V_i$ such that $X \cap W = I$ and for every $u \in V_i \setminus D$, $N(u) \cap W \neq \emptyset$.

- **Leaf bag**:

$$A[i, \{v\}, \emptyset, \emptyset] = \omega(v) \qquad A[i, \emptyset, \{v\}, \emptyset] = \infty \qquad A[i, \emptyset, \emptyset, \{v\}] = 0$$

  If v is included we account for its weight, if it is not but needs to be dominated we minimize over an empty set and get ∞. Otherwise we include nothing and get weight 0.

- **Introduce vertex bag**:

$$\begin{aligned}
&A[i, I \cup \{v\}, E, D] = \omega(v) + A[j, I, E \setminus N(v), D \cup N(v)] && \\
&A[i, I, E \cup \{v\}, D] = \infty, && \text{if } N(v) \cap I = \emptyset, \\
&A[i, I, E \cup \{v\}, D] = A[j, I, E, D], && \text{if } N(v) \cap I \neq \emptyset, \\
&A[i, I, E, D \cup \{v\}] = A[j, I, E, D], &&
\end{aligned}$$

  If v is included, we account for its weight and all its neighbors do not need to be dominated. If v is excluded we check whether its dominated and remove it if so (and if not, return ∞ since we minimize over an empty set). If v is excluded and dominated we simply remove it.

- **Forget bag**:

$$A[i, I, E, D] = \min\{A[j, I, E \cup \{v\}, D], A[j, I, E, D \cup \{v\}]\}$$

  Here we simply decide whether the vertex is included or excluded.

- **Join bag**:

$$A[i, I, E, D] = \min_{D' \subseteq D} \{A[j, I, E \cup D', (D' \setminus D')] + A[j', I, E \cup (D \setminus D'), D'] - \omega(I)\}$$

  Here we iterate through all possibilities over in which of the two subproblems the elements of D are dominated. Here $D' \subseteq D$ represents the subset of all vertices dominated by $X \cap V_j$. We ensure that D' is dominated by $X \cap V_j$ adding D' to the set E. For the other problem, then $D \setminus D'$ needs to be dominated by $X \cap V_{j'}$ so we add $D \setminus D'$ to E here. As for weighted independent set we subtract $\omega(I)$ since its accounted for in both subproblems.