1. Kruskal’s (as well as Prim’s) algorithm is a deterministic algorithm which computes a minimum spanning tree in polynomial time (given a graph with \( n \) nodes and \( m \) edges, it takes at most \( O(m \cdot \log n) \) time).

**Remark:** If different edges have the same weight, then there may be multiple minimum spanning trees. However, Kruskal’s algorithm can still be formulated in a deterministic way. For example, two edges with the same weight can (at the start) be ordered based on the names of the nodes they connect.

2. Suppose there are \( k \) tile types. Given is a sequence of \( n \) (matching) tiles that form the first row.

Build a nondeterministic TM which, on the given input, has \( k^{n(n-1)} \) different (nondeterministic) executions. Each execution puts tiles at the \( n(n-1) \) positions above the input string in the \( n \times n \) surface; there are \( k^{n(n-1)} \) different ways of doing this. Next it is checked whether the tiling chosen by the execution meets the requirements, meaning that both horizontally and vertically all touching sides of tiles have the same color. If this is the case, the TM goes to the final state (end else it halts in a non-final state).

Clearly, the input row of \( n \) tiles is accepted by this TM if and only if the bounded tiling problem has a solution with regard to this input. And each execution takes at most \( O(n^2) \) time, because a quadratic number of touching sides need to be checked.

3. (a) To build the first row:

\[
\begin{array}{cccccc}
q_0, a & q_0, b & a & b & \\
\end{array}
\]

Instructions of \( M \):
To leave the tape unchanged:

On each input string of size $k$, $M$ takes no more than $k$ steps. Hence we can take $p(k) = k$, and so $n = 2k + 1$. Since the input string $abb$ has size 3, we take $n = 7$.

The first row consists of:

(b) $M$ starts on the first symbol $a$ of the input string, in the start state $q_0$. Since $\delta(q_0, a) = (q_0, a, R)$, it leaves the $a$ unchanged, moves one step to the right, and stays in state $q_0$.

Now $M$ reads a $b$. Since $\delta(q_0, b) = (q_1, b, R)$, it leaves the $b$ unchanged, moves one step to the right, and goes to state $q_1$.

Now $M$ again reads a $b$. Since $\delta(q_1, b) = (q_f, b, L)$, it leaves the $b$ unchanged, moves one step to the left, and goes to the final state $q_f$.

(c)
4. (a) Suppose there are two instructions \((r_1, b_1, R)\) and \((r_2, b_2, R)\) in \(\delta(q, a)\). Then the tile with \(q, a\) as bottom, \(b_1\) as top and \(R\) as right color (which belongs to the first instruction) could be placed to the left of the tile with \(c\) as bottom, \(r_2, c\) as top and \(R\) as left color, for any \(c\) in the tape alphabet \(\Gamma\) (which belongs to the second instruction).

This pair corresponds to the (non-existent) instruction \((r_2, b_1, R)\) in \(\delta(q, a)\).

(b) Suppose there are two instructions \((r, b_1, R)\) and \((r, b_2, L)\) in \(\delta(q, a)\). Then the tile with \(q, a\) as bottom, \(b_1\) as top and \(r\) as right color (which belongs to the first instruction) could be placed to the left of the tile with \(q, a\) as bottom, \(b_2\) as top and \(r\) as left color (which belongs to the second instruction).

This pair does not correspond to any TM instruction.

5. Suppose that languages \(L_1\) and \(L_2\) are in \(P\), so they are accepted by polynomial-time bounded deterministic TMs \(M_1\) and \(M_2\), respectively.

Let the TM \(N_1\) be the same as \(M_1\), except that the final states in \(M_1\) are made non-final in \(N_1\). Furthermore, for each non-accepting halt state in \(M_1\), add a transition in \(N_1\) (for any input symbol) to a new final state.

Since \(M_1\) is deterministic, clearly \(N_1\) accepts \(\overline{L_1}\). And since \(M_1\) is a polynomial-time bounded deterministic TM, the same holds for \(N_1\).

Let the TM \(N_2\) first simulate \(M_1\) on the input string. If that leads to a final state, then \(N_2\) also reaches a final state. If on the other hand that simulation reaches a non-accepting halt state, then \(N_2\) simulates \(M_2\) on the input string. If that leads to a final state, then again \(N_2\) reaches a final state.

Clearly, \(N_2\) accepts \(L_1 \cup L_2\). And since \(M_1\) and \(M_2\) are polynomial-time bounded deterministic TMs, the same holds for \(N_2\).

Let the TM \(N_3\) first simulate \(M_1\) on the input string. If that leads to a final state, then \(N_3\) simulates \(M_2\) on the input string. If that also leads to a final state, then \(N_3\) reaches a final state.

Clearly, \(N_3\) accepts \(L_1 \cap L_2\). And since \(M_1\) and \(M_2\) are polynomial-time bounded TMs, the same holds for \(N_3\).

6. Suppose that languages \(L_1\) and \(L_2\) are in \(NP\), so they are accepted by polynomial-time bounded (nondeterministic) TMs \(M_1\) and \(M_2\), respectively.

For both union and intersection, the argumentation from the previous exercise can be copied, but now for nondeterministic instead of deterministic TMs.

(Actually, for union there is now a simpler alternative solution: let \(N_2\) nondeterministically decide to simulate either either \(M_1\) or \(M_2\) on the input string.)

For the case of complements, the argumentation from the previous exercise can’t be copied. Namely, since \(M_1\) is nondeterministic, it can have an execution to a final state as well as an execution to a non-accepting halt state.
7. On the slides it is stated that polynomial-time reductions are compositional: the composition $g \circ f : \Sigma_1^* \to \Sigma_3^*$ of polynomial-time reductions $f : \Sigma_1^* \to \Sigma_2^*$ and $g : \Sigma_2^* \to \Sigma_3^*$ is a polynomial-time reduction.

Consider any language $L$ in NP. Since $L_1$ is NP-complete, there exists a polynomial-time reduction from $L$ to $L_1$. And by assumption there exists a polynomial-time reduction from $L_1$ to $L_2$. The composition of these two reductions produces a polynomial-time reduction from $L$ to $L_2$.

Since this holds for any language $L$ in NP, and $L_2$ is in NP, by definition, $L_2$ is NP-complete.

8. If $P = NP$, then there is a polynomial-time algorithm to solve say the bounded tiling problem (or any other NP-complete problem). By presenting this algorithm and proving that it takes polynomial time at most, it can be concluded that the bounded tiling problem is in $P$. And since the bounded tiling problem is NP-complete, it follows that $P = NP$.

On the other hand, if $P \neq NP$, then in principle it may be the case that no proof of this fact exists. Gödel’s incompleteness theorem states that not each valid mathematical theorem allows a proof. $P \neq NP$ might be an example of a valid theorem for which no proof exists.