Solutions to exercises of lecture 5

**Exercise 5.1.** Recall that in the Traveling Salesman problem, we are given a graph $G = (V, E)$ with an integer weight $w_e$ and the question is to find a Hamiltonian cycle $C \subseteq E$ minimizing $\sum_{e \in C} w_e$. Can you solve it in $O^*(n!)$ time?

**Solution:** Every Hamiltonian cycle is described by a permutation $v_1, \ldots, v_n$ of the vertices so we can simply iterate over all permutations $v_1, \ldots, v_n$ of $V$ and see which one minimizes $w(v_n, v_1) + \sum_{i=1}^{n-1} w(v_i, v_{i+1})$.

**Exercise 5.2.** In the $k$-coloring problem we are given a graph $G$ and integer $k$ and need to determine whether $G$ has a $k$-coloring. Do you expect this problem parameterized by $k$ to be FPT?

**Solution:** Not assuming $P \neq NP$: an $O(f(k)n^c)$ time algorithm for this, for some constant $c$, would imply an $O(n^c)$ time algorithm for 3-coloring.

**Exercise 5.3.** Find an algorithm detecting cliques of size at least $k$ in $O(n^k k^2)$ time, why is this running time not sufficient to prove the problem to be FPT?

**Solution:** We cannot write $O(n^k)$ as $f(k)\text{poly}(n)$, since the latter implies a fixed exponent of $n$ while in the first the exponent depends on $k$.

**Exercise 5.4.** Show that if $G$ has a FVS of size at most $k$, it has a $k+2$-coloring. Can you give an example of a graph with a FVS of size at most $k$ but no $k+1$ coloring?

**Solution:** use colors $1, \ldots, k$ to color all vertices in the FVS with a distinct color, use $k+1, k+2$ for a two-coloring of the forest (which is easily seen to exist by fixing one color and propagating). A complete graph on $k+2$ vertices would be such an example.

**Exercise 5.5.** Give an $O^*(2^{n/2})$ time, $O^*(2^{n/4})$ space algorithm for Subset Sum using the 4SUM algorithm.
**Solution:** Assume \( n \) is a multiple of 4, construct an integer \( a_i \) for every \( W \subseteq \{1, \ldots, n/4\} \), \( b_i \) for every \( X \subseteq \{n/4+1, \ldots, n/2\} \), \( c_i \) for every \( Y \subseteq \{n/2+1, \ldots, 3n/4\} \), \( d_i \) for every \( Z \subseteq \{3n/4+1, \ldots, n\} \), set the target of the 4SUM problem to be \( t \). This 4SUM instance has a solution if and only if the subset sum instance has one since every subset \( S \subseteq \{1, \ldots, n\} \) can be written as \( W \cup X \cup Y \cup Z \).

**Exercise 5.6.** Can you solve 4-coloring in \( O^*(2^n) \) time? What about 3-coloring in \( O^*((2-\epsilon)^n) \) time, for some \( \epsilon > 0 \) (Hint: use that \( \binom{n}{k} \leq 2^{0.92n} \) for \( k \leq n/3 \))?

**Solution:** For 4-coloring, we may iterate over all vertex sets \( X \subseteq V \) that could have the first two colors. Given such \( X \) we just need to see whether both \( G[X] \) and \( G[V \setminus X] \) are 2-colorable.

For the second question, note that one color class must be of size at most \( n/3 \) so in Algorithm 3colv2 we may iterate over all sets of size at most \( n/3 \) instead.

**Exercise 5.7.** Solve Vertex Cover in \( O^*(1.4656^k) \) time.

**Solution:** Adjust \( vc2 \) as follows: if there exists no vertex of degree at least 3, we have a set of cycles, paths and isolated vertices and an optimal solution is computed in polynomial time by a simple greedy argument. Otherwise, branch as in Line 4 of \( vc2 \). If \( T(k) \) denotes the number of leaves in the branching tree we see that \( T(0) = 1 \) and for \( k > 0 \)

\[
T(k) \leq \max_{d \geq 3} T(k-1) + T(k-d).
\]

We see that \( T(k) \) is bounded by \( 1.4656^k \) since \( 1.4656^{-1} + 1.4656^{-3} \leq 1 \).

**Exercise 5.8.** Recall the definition of NP. Why can any problem instance \( x \in \{0,1\}^n \) of a language in NP be solved in \( 2^{poly(|x|)} \) time?

**Solution:** NP: there exists a polynomial time verifier \( V \), (e.g., an algorithm that runs in time polynomial in \( x \) with the following property: there exists a certificate \( c \) such that \( V(x, c) \) returns true if and only if \( x \in L \)). Since \( V \) runs in time polynomial in the input, \( |c| \) needs to be polynomial in \( |x| \), so given an instance \( x \), we can iterate over all \( 2^{|c|} = 2^{poly(|x|)} \) possible \( c \) and see whether \( V(x, c) \) gives true somewhere.

**Exercise 5.9.** An algorithm running in time \( n^{\lg(n)^c} \) for some constant \( c \) is called quasi-polynomial. Recently, in a big breakthrough, László Babai showed that the ‘Graph Isomorphism problem’ can be solved in quasi-polynomial time. Graph Isomorphism is not known to be NP-complete. Can you explain why a quasi-polynomial time algorithm for an NP-complete problem would be a huge result (Hint: recall the definition of NP-completeness)?

**Solution:** NP-complete: \( L \) is NP-complete if for every other problem \( L' \) in NP there exists a polynomial time reduction from \( L' \) to \( L \), e.g., an algorithm \( R \) such that for every input \( x, x \in L' \) if and only if \( R(x) \in L \). Since \( R \) is polynomial time, \( |R(x)| \) is polynomial in \( |x| \) thus if \( L \) is solved in time \( |x|^{\lg(|x|)^c} \), then this gives an

\[
|R(x)|^{\lg(|R(x)|)^c} = (|x|^{c'})^{\lg(|x|)^c} = |x|^{c'\lg(|x|)^c},
\]

\(^1\) (see e.g., http://www.quantamagazine.org/20151214-graph-isomorphism-algorithm/)
time algorithm for problem \( L' \). So this would be a huge result because it implies a quasi-polynomial
time algorithm for any NP-complete problem.

**Exercise 5.10.** Show that Feedback Vertex Set is NP-hard. In particular, show that given an
instance \((G, k)\) of vertex cover, we can compute in polynomial time an equivalent instance \((G', k)\)
of feedback vertex set.

**Solution:** Given an instance \( G = (V, E) \), \( k \) of vertex cover, add a vertex \( v_e \) for every edge \((u, v)\)
with neighbors \( \{u, v\} \). There always exists an optimal FVS in which no added vertex is picked since
\( v_e \) can be replaced with either \( u \) or \( w \) if \( e = \{u, w\} \), and such a FVS is a FVS of the new graph if
and only if it is a vertex cover of the old graph since for every edge \( e = (u, w) \) it needs to hit the
triangle \( u, w, v_e \).

Alternatively, one could apply the degree 2 reduction rule to obtain a multigraph in which all
edges occur twice.

**Exercise 5.11.** The \( n \)'th Fibonacci number \( f_n \) is defined as follows: \( f_1 = 1, f_2 = 1 \) and for \( n > 2, f_n = f_{n-1} + f_{n-2} \). What is the running time of the following algorithm to compute \( f_n \)?

<table>
<thead>
<tr>
<th>Algorithm FIB1(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output:</strong> ( f_n )</td>
</tr>
<tr>
<td>1: if ( n = 1 ) or ( n = 2 ) then return 1</td>
</tr>
<tr>
<td>2: return FIB1((n-1))+FIB1((n-2)).</td>
</tr>
</tbody>
</table>

**Solution:** The running time is at most \( O(nf(n)) \). To see this, note that the number of leaves is
exactly \( f(n) \). If you insist to be more precise to get rid of the \( n \) factor, note that the branching
tree has no degree 2 vertices, and for any such tree the number of internal vertices is at most the
number of leaves.

**Exercise 5.12.** In the Set Partition problem we are given \( F_1, \ldots, F_m \subseteq U \) and need to find a
subset of the sets that partition \( U \). Can you do this in \( O^*(2^{m/2}) \) time?

**Solution:** Assume \( m \) is even by adding an empty set. Enumerate \( L = \{ \bigcup_{i \in X} F_i : X \subseteq \{1, \ldots, m/2\} \} \)
and \( R = \{ \bigcup_{i \in X} F_i : X \subseteq \{m/2 + 1, \ldots, m\} \} \). For every \( Y \in L \) check whether \( U \setminus Y \) is in \( R \), return
yes if so and no otherwise.

**Exercise 5.13.** In this exercise we’ll look at the \( d \)-Hitting Set problem: given sets \( F_1, \ldots, F_m \subseteq U \)
of size \( d \) each, where \( |U| = n \), we need to find a subset \( X \subseteq U \) with \( |X| = k \) that ‘hits’ every set in
the sense that \( F_i \cap X \neq \emptyset \) for every \( i \).

1. By which other name do you know 2-Hitting Set? Why is it equivalent?
2. Can you solve 3-Hitting Set in time \( O^*(3^k) \)?
3. Can you solve 3-Hitting Set in time \( O^*(2.4656^k) \)

    - Hint: Use iterative compression. Suppose you are also given a hitting set of size \( k + 1 \),
      can you solve the problem in time \( O^*(\sum_{i=1}^{k+1} (k+1)^i \cdot 1.4656^i) \). This equals \( O^*(2.4656^k) \) by
      the binomial theorem.
Solution: Vertex Cover. The elements are the vertices, the sets the edges and there is a direct correspondence.

Pick a set of size at most 3 and branch on one of the three elements that needs to be included. Suppose we have a 3-hitting set $H$ of size $k + 1$. Guess the subset $X \subseteq H$ that will be in the solution. Remove all sets that intersect with $X$ and remove all elements from $H$. Since $H$ was a hitting set, all sets are now of size at most 2. Pick elements in sets of size 1 so only sets of size 2 remain, and we have an instance of vertex cover. This instance of vertex cover can be solved in time $O^*(1.4656^{k-X})$ using the algorithm of Exercise 5.7. So indeed the running time of one compression step becomes $O^*(\sum_{i=1}^{k+1} \binom{k+1}{i} 1.4656^i)$.

Exercise 5.14. Give an algorithm that determines whether a given 3-CNF-Sat formula is satisfiable in time $O^*((2 - \epsilon)^n)$, for some $\epsilon > 0$.

Solution: Use the following branching algorithm: pick a clause of maximum size and branch on all assignments of its variables satisfying it. For example, if the clause is $\neg v_2 \lor v_4 \lor \neg v_6$, we recurse on the CNF-formula obtained by setting $v_2, v_4, v_6$ to all 8 assignments except 1, 0, 1. This results in 7 new recursive calls on formula’s with at most $n - 3$ variables, so we can use the following recurrence for the number of leaves of the branching tree $T(0) = 1$ and for $n > 0$

$$T(n) \leq \max\{7 \cdot T(n - 3), 3 \cdot T(n - 2), T(n - 1)\}.$$ 

Setting $T(n) = 7^{n/3} < 1.913$ works since

$$\max\{7 \cdot 7^{n-3/3}, 3 \cdot 7^{n-2/3}, 7^{n-1/3}\} \leq 7^{n/3}. $$