Chapter 2 : Time complexity : Solutions of the exercises

Section 2.1

1. Whether a graph G is connected can be determined (in poly-time) by growing a BFS tree rooted at any vertex of the graph. G is connected, if and only if this tree spans all the vertices of G.

A graph G is bipartite, if and only if every component of G is bipartite. For proving if a connected graph G is bipartite, we grow a BFS tree rooted at any node of G. It's a property of the BFS algorithm that if there is an edge in G between a node at distance i and a node at distance j from the root of the spanning tree, then we must have $i = j$ or $|i - j| = 1$. It's well-known that G is bipartite, if and only if G does not contain an odd cycle. It follows that G is bipartite, if and only if it does not contain an edge between two nodes at the same distance from the root in the spanning tree.

In order to check if a graph G with m vertices is triangle-free, it is sufficient to consider all triples (u, v, w) of pairwise distinct vertices of G. There are only $\binom{m}{3} = \mathrm{O}(m^3)$ of such triples.

2. If $G_1 \cong G_2$, a bijective map $G_1 \to G_2$ that respects adjacency in the two graphs constitutes a succinct certificate for the isomorphism.

3. We know that the multiplicative group $\mathbb{Z}_p^*$ is cyclic and of order $p-1$, if and only if p is a prime. Let $p-1 = p_1^{e_1} \cdots p_r^{e_r}$ be the prime decomposition of $p - 1$. If p is prime, then $g \in \mathbb{Z}_p$ generates $\mathbb{Z}_p^*$ if and only if $g^{(p-1)/p_i} \not\equiv 1 \pmod{p}$ for all $i = 1, \ldots, r$. Each such modular exponentiation can be carried out in poly-time by the repeated square-and-multiply algorithm. So g along with the prime divisors of $p - 1$ produces a certificate for the primality of p. One may additionally supply the exponents $e_1, \ldots, e_r$, but those can be computed in deterministic polynomial time (for the check that $p - 1 = p_1^{e_1} \cdots p_r^{e_r}$).

This certificate is incomplete, because it does not include a certificate that each p_i is prime. But each such certificate can be generated inductively. In other words, a certificate $C(p)$ for p looks like:

$$C(p) = (g, (p_1, e_1, C(p_1)), \ldots, (p_r, e_r, C(p_r))).$$

The induction stops for the prime 2 for which $2 - 1$ has a trivial factorization. As an example, consider $p = 67$. 2 is a generator of $\mathbb{Z}_{67}^*$. Also $67 - 1 = 66 = 2 \times 3 \times 11$. In this decomposition 2 is surely prime. 3 is a prime, because $3 - 1 = 2$ has a certifiable prime decomposition and 2 is a generator for $\mathbb{Z}_3^*$. With 11 we have to do more. 8 is a generator of $\mathbb{Z}_{11}^*$, $11 - 1 = 10 = 2 \times 5$, 2 is prime, and finally 5 is prime, since 3 generates $\mathbb{Z}_5^*$ and $5 - 1 = 4 = 2^2$. To sum up we have:

$$\begin{aligned}
C(67) &= (2, (2, 1, C(2)), (3, 1, C(3)), (11, 1, C(11))) \\
&= (2, (2, 1, (1)), (3, 1, (2, (2, 1, C(2)))), (11, 1, (8, (2, 1, C(2)), (5, 1, C(5))))) \\
&= (2, (2, 1, (1)), (3, 1, (2, (2, 1, (1)))), (11, 1, (8, (2, 1, (1)), (5, 1, (3, (2, 2, C(2))))))) \\
&= (2, (2, 1, (1)), (3, 1, (2, (2, 1, (1)))), (11, 1, (8, (2, 1, (1)), (5, 1, (3, (2, 2, (1)))))))
\end{aligned}$$

But how big is this certificate $C(p)$. One can inductively prove that $|C(p)| = \mathrm{O}(\lg^2 p)$, since $\prod_{i=1}^{r} p_i \leqslant p - 1$, each $e_i \leqslant \lg p$ and $r \leqslant \lg p$. It is also easy to check that this certificate can be verified in $\mathrm{O}(\lg^4 p)$ time. For details look at Papadimitriou, Section 10.2.

4. **(a)** Let $L_1, L_2 \in \mathrm{P}$ with respective poly-time DTMs M_1 and M_2. We will construct poly-time DTMs for the following languages. We assume that α is the input.

$[L_1 \cup L_2]$
Simulate M_1 on α. If M_1 accepts, accept and halt. Otherwise, simulate M_2 on α. If M_2 accepts, accept and halt, otherwise reject and halt.

$[\overline{L_1}]$
Simulate M_1 on α and flip its decision.

$[L_1 \cap L_2]$
Simulate M_1 and M_2 sequentially on α. Accept, if both M_1 and M_2 accept; else reject. Alternatively, use the fact that $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

$[L_1 L_2]$
Let $\alpha = a_1 a_2 \ldots a_n$. For each $i \in \{0, 1, \ldots, n\}$ check if $\beta_i := a_1 \ldots a_i \in L_1$ and $\gamma_i := a_{i+1} \ldots a_n \in L_2$ by simulating M_1 and M_2 on β_i and γ_i respectively. If there is an i for which $\beta_i \in L_1$ and $\gamma_i \in L_2$, accept; else reject.

$[L_1^*]$
Again let $\alpha = a_1 a_2 \ldots a_n$. For each i, j with $1 \leqslant i \leqslant j \leqslant n$ check if $\beta_{i,j} := a_i \ldots a_j \in L_1$ by simulating M_1 on $\beta_{i,j}$. Now generate an $n \times n$ matrix $T := (t_{i,j})$. Only the entries $t_{i,j}$ for $i \leqslant j$ (i.e., the entries on and above the main diagonal) will be used. Generate T in the diagonal-major order. First for $i = 1, \ldots, n$ set $t_{i,i} = 1$ if $\beta_{i,i} \in L_1$; else set $\beta_{i,i} = 0$. Then for $i = 1, \ldots, n-1$ set $t_{i,i+1} = 1$ if $\beta_{i,i+1} \in L_1$, or if $t_{i,i} = t_{i+1,i+1} = 1$; else set $t_{i,i+1} = 0$. Now inductively assume that $t_{i,j}$ are generated for all i, j with $0 \leqslant j - i < l$. Then for $i = 1, \ldots, n - l$ set $t_{i,i+l} = 1$ if $\beta_{i,i+l} \in L_1$ or if $t_{i,k} = t_{k+1,i+l} = 1$ for some $k \in \{i, i+1, \ldots, i+l-1\}$.

(b) The constructions of Part (a) apply *mutatis mutandis* for NTMs. The only exception is with $\overline{L_1}$. Convince yourself why the construction for DTMs does not work with NTMs in this case.

Section 2.2

1. Clearly, $L \in \mathrm{NP}$. Take any $A \in \mathrm{NP}$. We have to show that $A \leqslant_P L$. Since L is neither empty nor Σ^*, we can find $\beta \in L$ and $\gamma \in \overline{L}$. Now we construct a poly-time computable function $f : \Sigma^* \to \Sigma^*$ such that $\alpha \in A$ if and only if $f(\alpha) \in L$. Since $\mathrm{P} = \mathrm{NP}$, first decide in deterministic poly-time whether $\alpha \in A$. Then set $f(\alpha) = \beta$ if $\alpha \in A$; else set $f(\alpha) = \gamma$. Since β and γ have constant lengths (with respect to $|\alpha|$), this last step can be done in deterministic poly-time.

2. Assume that $\delta(q_1, a) = \{(q_2, b, L), (q_3, c, R)\}$. Then both $\begin{array}{|c|c|}\hline c & q_1 \\ \hline q_2 & c \\ \hline\end{array}$ and $\begin{array}{|c|c|}\hline q_1 & a \\ \hline c & q_3 \\ \hline\end{array}$ are legal 2×2 windows. If we conjoin these windows at the facing columns, we get the 2×3 window $\begin{array}{|c|c|c|}\hline c & q_1 & a \\ \hline q_2 & c & q_3 \\ \hline\end{array}$ which is clearly illegal.

3. We reduce 2SAT to $\overline{\mathrm{PATH}}$ in poly-time. Since $\mathrm{PATH} \in \mathrm{P}$ (and P is closed under complement), this reduction will show that $2\mathrm{SAT} \in \mathrm{P}$ too. Carefully look at the direction of the reduction. When we want to prove a problem is easy, we reduce this problem to a problem that is already known to be easy. On the other hand, when we want to prove a problem is difficult, we reduce a known difficult problem to this new problem.

Back to the business! Given a formula ϕ in 2-cnf, we construct a graph G such that ϕ is not satisfiable if and only if there are u, v and v, u paths in G for some $u, v \in V(G)$. The construction first: Let $\phi = (a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge \cdots \wedge (a_l \vee b_l)$. For each variable x create a node labeled x and another labeled $\bar{x}$. For each clause $(a \vee b)$ add an edge from node $\bar{a}$ to node b and also an edge from $\bar{b}$ to node a. That's all!

Claim ϕ is not satisfiable, if and only if for some variable x the graph G contains both $x, \bar{x}$ and $\bar{x}, x$ paths.

Proof [if] Suppose that there is such a variable x. Consider an arbitrary assignment of the variables. Assume that $x = 0$ in this assignment. (The other case can be handled symmetrically.) G contains an $\bar{x}, x$-path. But $\bar{x} = 1$ and $x = 0$; so there exists an edge (u, v) on this path such that $u = 1$ and $v = 0$. By construction $(\bar{u} \vee v)$ is a clause in ϕ and this clause (and hence ϕ) evaluate to 0 under this truth assignment.

[only if] Suppose that there is no such variable x. We have to obtain a satisfying assignment for ϕ. Repeat the following steps, as long as there is a node in G that has not been assigned a truth value. "Choose such a node a for which G does not contain an $a, \bar{a}$ path. Assign all nodes reachable from a (including a itself) the value 1 and all complements of these nodes the value 0."

First note that such a choice for a is always possible, since G does not contain both $a, \bar{a}$ and $\bar{a}, a$ paths for any a and we assign truth values to literals b and $\bar{b}$ simultaneously (and consistently). Next we show that the truth assignment in the loop is well-defined, i.e., we do not attempt to make a node b simultaneously true and false. Assume that there is such a b. First consider that b (and $\bar{b}$) are not assigned truth values earlier. But then there are a, b and $a, \bar{b}$ paths in G.

By symmetry in the construction there is also a $b, \bar{a}$ path in G and hence an $a, \bar{a}$ path in G, contrary to the choice of a. Next consider that there is an a, b path with b assigned false earlier. But then there is a $\bar{b}, \bar{a}$ path and $\bar{b} = 1$. Thus $\bar{a}$ must have been considered earlier, again a contradiction to the choice of a.

Thus the procedure of truth assignment terminates with a logically consistent assignment of truth values to the nodes in G. In this assignment there is no edge from a true literal to a false literal. If $(a \vee b)$ is a clause in ϕ, then both $(\bar{a}, b)$ and $(\bar{b}, a)$ are edges in G. If $a = 0$ in the assignment, then $\bar{a} = 1$ and so $b = 1$. Similarly, if $b = 0$, we have $a = 1$. Thus we always have $a \vee b = 1$ in this assignment. ◀

4. We solve this exercise only for directed graphs, the construction for undirected graphs being similar.

Clearly, HAMCYCLE $\in$ NP. In order to prove HAMCYCLE is NP-hard, we reduce HAMPATH to HAMCYCLE. Let $G = (V, E)$ be a directed graph with two vertices s and t. We plan to convert $\langle G, s, t \rangle$ to a directed graph $G' = (V', E')$ such that G has an s, t Hamiltonian path if and only if G' has a Hamiltonian cycle. Take

$$\begin{aligned} V' &:= V \uplus \{u\}, \text{ and} \\ E' &:= E \cup \{(u, s), (t, u)\}. \end{aligned}$$

Suppose that G has an s, t Hamiltonian path $s, v_1, v_2, \ldots, v_m, t$. Then $u, s, v_1, v_2, \ldots, v_m, t, u$ is a Hamiltonian cycle in G'. Conversely, let G' have a Hamiltonian cycle. If we traverse around the cycle starting from u, we must first reach s after leaving u. In order to complete the cycle we must take the edge (t, u). Between s and t the cycle visits every other node of G exactly once, i.e., this cycle must be of the form $u, s, v_1, v_2, \ldots, v_m, t, u$. But then $s, v_1, v_2, \ldots, v_m, t$ is an s, t Hamiltonian path in G.

5. Clearly, TSP $\in$ NP. To prove TSP is NP-hard we reduce UHAMCYCLE to TSP. Let G be an instance for UHAMCYCLE. Let m be the number of vertices in G. Consider the complete graph G' with $V(G') = V(G)$ and with the cost of edge (u, v) equal to $\begin{cases} 1 & \text{if } (u,v) \in E(G), \\ m+1 & \text{if } (u,v) \notin E(G). \end{cases}$ Then the converted instance for TSP will be $\langle G', m \rangle$. Clearly, a Hamiltonian cycle in G translates to a Hamiltonian cycle in G' with each edge cost equal to 1. Conversely, if G' has a Hamiltonian cycle of cost $\leqslant m$, that cycle cannot use an edge of cost $m + 1$, i.e., an edge not in $E(G)$. Thus this cycle resides in G as well.

6. Clearly, DOUBLE-SAT $\in$ NP. We then show SAT $\leqslant_P$ DOUBLE-SAT. Let $\phi = \phi(x_1, \ldots, x_m)$ be an instance for SAT. We convert ϕ to an instance ϕ' for DOUBLE-SAT. If $m = 0$, take $\phi' := \phi \vee y$ for a new variable y. In this case, if $\phi = 0$, then ϕ' has only one satisfying assignment, namely $y = 1$, whereas if $\phi = 1$, then both the choices for y let ϕ' evaluate to 1. So assume that $m \geqslant 1$. Introduce new variables $y_1, \ldots, y_m$ and take $\phi' := \phi(x_1, \ldots, x_m) \vee \phi(y_1, \ldots, y_m)$. If ϕ is unsatisfiable, so is ϕ' too. If ϕ is satisfiable, take a satisfying assignment of $x_1, \ldots, x_m$ for ϕ and let $y_1, \ldots, y_m$ assume any of the $2^m \geqslant 2$ possible values.

7. **(a)** A graph is 2-colorable if and only if it is bipartite. We have proved BIPARTITE $\in$ P.

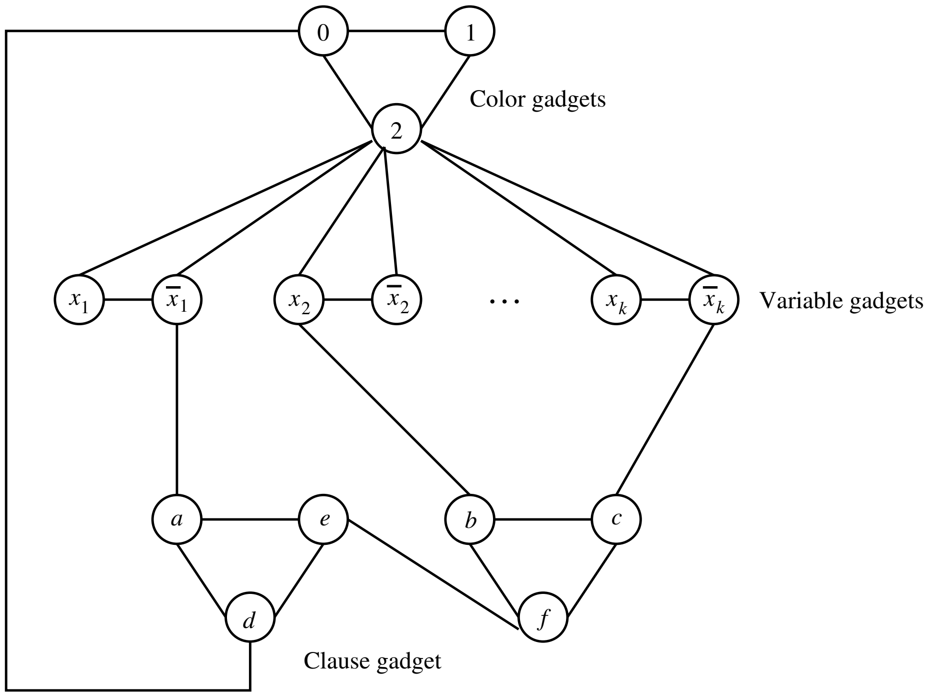
(b) We clearly have 3COLOR $\in$ NP. We show that 3SAT $\leqslant_P$ 3COLOR. The reduction procedure is a bit complicated. Let $\phi = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_m \vee b_m \vee c_m)$ be a 3-cnf formula in k variables $x_1, \ldots, x_k$. Our aim is to construct an undirected graph G with the property that G is 3-colorable if and only if ϕ is satisfiable.

Let us name the colors as $0, 1, 2$ with the logical implication that 0 means 'false', 1 means 'true' and 2 means neither! We first create a triangle with nodes representing the colors $0, 1, 2$. Then for each variable x_i create two nodes labeled x_i and $\overline{x}_i$ and complete the triangle on these two nodes and the color node 2. Finally, consider a clause $(a \vee b \vee c)$ in ϕ. Create six nodes a, b, c, d, e, f. Add edges to make a triangle on a, d, e and another on b, c, f. Join d to color node 0, e to f and each of a, b, c to the same literal in the variable gadgets. This construction is described in Figure 2.1. Only one clause $(\overline{x}_1 \vee x_2 \vee \overline{x}_k)$ is shown in this figure.

For the proof of the correctness of this construction first note that every triangle is 3-colorable, but not 2-colorable. That is, the three nodes in any triangle must receive three different colors. We now claim that G is 3-colorable if and only if each clause gadget has a true literal.

First consider a (valid) 3-coloring of G. Permute the colors, if necessary, so that the color node i receives color i for each $i = 0, 1, 2$. Since each variable gadget x_i forms a triangle with the color node 2, we must have the colors of x_i

Figure 2.1: Reduction of 3SAT to 3COLOR



and $\overline{x}_i$ either $0, 1$ or $1, 0$. In the former case assign x_i the truth value 0 (false); in the other case the truth value 1 (true). I have to show that this truth assignment makes at least one literal true in each clause gadget. Suppose, on the contrary, that there is a clause $(a \vee b \vee c)$ for which the above truth assignment gives $a = b = c = 0$. Consider the gadget (on nodes a, b, c, d, e, f) for this clause. The nodes b and c in this clause gadget cannot receive the color 0 (since they are connected to nodes with color 0 in variable gadgets). Since b, c, f from a triangle, f must then receive the color 0. But then e cannot receive color 0. Also since the literal a is false, the node a in the clause gadget cannot also receive the color 0. Thus node d must receive color 0. But d is adjacent to the color node 0, a contradiction.

For the converse, consider a satisfying assignment for ϕ. If $x_i = 0$, color x_i by 0 and $\overline{x}_i$ by 1 in the variable gadget for x_i. If $x_i = 1$, do the reverse coloring. Also give the color 0, 1 and 2 to the color nodes 0, 1 and 2 respectively. What remains is to color vertices in each clause gadget in a valid manner. Since there is no edge between two different clause gadgets, we can color each such gadget independently. Consider the clause gadget with nodes a, b, c, d, e, f for the clause $(a \vee b \vee c)$. The following table summarizes valid colorings for all truth values of the literals a, b, c with at least one true.

Truth			Coloring					
a	b	c	a	d	e	b	c	f
0	0	1	2	1	0	1	0	2
0	1	0	2	1	0	0	1	2
0	1	1	2	1	0	0	2	1
1	0	0	0	1	2	1	2	0
1	0	1	2	1	0	1	0	2
1	1	0	2	1	0	0	1	2
1	1	1	2	1	0	0	2	1

Section 2.3

1. **(a)** It is easy to see that the constructions of Exercise 2.1.4 work for the class EXP as well.

(b) Let $A \in \text{coNP}$. Then $\overline{A} \in \text{NP}$ and so $\overline{A} \in \text{EXP}$. But EXP is closed under complement, so $A \in \text{EXP}$.

2. The idea behind this problem is that when an NTM says 'accept', the input gets accepted. However, a branch of computation ending in a 'reject' verdict does not necessarily imply rejection of the input. That leaves us in the 'notsure' case.

[if] Let $L \in \mathrm{NP} \cap \mathrm{coNP}$. Let N and $\bar{N}$ be two poly-time NTMs accepting L and $\bar{L}$ respectively. Create a strong NTM S to accept L as follows. S runs the two machines N and $\bar{N}$ sequentially on the (same) input α. First, if the simulated N accepts α, S also accepts and halts. If N (i.e., a branch of N) rejects, S simulates $\bar{N}$ on α. If $\bar{N}$ accepts, S rejects α and halts. If $\bar{N}$ also rejects (like N), S halts after outputting 'notsure'. If $\alpha \in L$, at least one branch of computation of N and hence of S lead to its acceptance. On the other hand, if $\alpha \notin L$, no branch of N accepts it, but at least one branch of $\bar{N}$ accepts it. Thus a rejecting branch of computation by S is available from an accepting branch of $\bar{N}$.

[only if] Let S be a poly-time strong NTM that decides L. First design a poly-time NTM N to accept L: simulate S with the 'notsure' outcomes of S changed to 'reject'. Conversely, an NTM $\bar{N}$ for $\bar{L}$ can be designed by simulating S with the 'reject' outcomes of S interpreted as 'accept' for $\bar{N}$, and with 'accept' and 'notsure' outcomes of S interpreted as 'reject'.

3. Observe that n has a divisor d with $1 < d < k$ if and only if n has a *prime* divisor p with $1 < p < k$. So a complete factorization of n is a succinct certificate for both FACTORING and $\overline{\text{FACTORING}}$. Since PRIME $\in$ NP, this certificate must be furnished with certificates of primality for each of its prime divisors. Now we know that PRIME $\in$ P, so these primality certificates are actually not necessary.

(Note that a divisor d of n with $1 < d < k$ is also a succinct certificate for FACTORING. For $\overline{\text{FACTORING}}$, however, existence of a similar certificate is not obvious.)

4. Let $L \in \mathrm{NEXP}$ and let N be an NTM that decides L in time 2^{n^k} for some $k \in \mathbb{N}$. Define the language $L' := \{\alpha \underline{\sqcup}^{2^{|\alpha|^k} - |\alpha|} \mid \alpha \in L\}$, where $\underline{\sqcup}$ is a 'quasiblank' symbol. We design a poly-time NTM N' for L'. Upon input α' the machine N' first checks if α' is of the form $\alpha \underline{\sqcup}^{2^{|\alpha|^k} - |\alpha|}$ for some $\alpha \in \Sigma^*$. If not, N' rejects α' and halts. Otherwise, N' simulates N on α, this time treating quasiblanks as blanks. N' runs in exp-time in $|\alpha|$ and hence in poly-time in $|\alpha'|$, i.e., N' is a poly-time NTM for L'.

By hypothesis, P $=$ NP and so there is a poly-time DTM M' to accept L'. What remains is to convert M' to an exp-time DTM M to accept L. M on input α appends the requisite number of quasiblanks to obtain $\alpha' := \alpha \underline{\sqcup}^{2^{|\alpha|^k} - |\alpha|}$, and simulates M' on α'. M accepts α if and only if M' accepts α'.

Section 2.4

1. Assume that TSP $\in$ P. Let M be a DTM that decides TSP in poly-time. We may assume without loss of generality that M has the binary input alphabet. We construct a DTM M' that solves TTSP using polynomially many invocations of the machine M. Let $\alpha = \langle G \rangle$ be an input for TTSP, where G is a complete graph with labeled edges. Suppose $n := |\alpha|$. Since each edge weight is encoded in α, G cannot have a Hamiltonian cycle of weight $> 2^n$. M' simulates TSP on $\langle G, 2^{n-1} \rangle$. If M returns 'yes', i.e., if G contains a Hamiltonian tour of total cost $\leqslant 2^{n-1}$, M' invokes M with input $\langle G, 2^{n-2} \rangle$, else M' invokes M on input $\langle G, (2^n + 2^{n-1})/2 \rangle$. In general, if M' has computed that a cost of c is attainable, whereas a cost c' is not, for some c, c' with $c' < c$ (initially $c = 2^n$ and $c' = 0$), M' simulates M on $\langle G, (c + c')/2 \rangle$ in order to reduce the search space from $[c', c)$ to $[c', (c + c')/2)$ or $[(c + c')/2, c)$ depending on the outcome of this simulation. It is clear that $\mathrm{O}(n)$ invocations of M suffice to determine the exact cost, call it C, of an optimal Hamiltonian tour in G.

Now M' repeats the following steps for each edge e of G (chosen in some arbitrary order): M' changes the label of e to $C + 1$ and simulates M on this new graph and on the cost C. If M returns 'yes', there is an optimal Hamiltonian tour that does not use e, and the cost of e is maintained at $C + 1$. If M returns 'no', every optimal Hamiltonian tour uses e, so its cost is changed back to the old value and another edge of G is considered.

When all edges of G are considered, G has precisely n edges whose costs are not equal to $C + 1$. These edges form an optimal Hamiltonian tour in G. Notice that G may have many such tours. The tour computed as above depends on the order in which the edges of G are processed. Since G has $\binom{n}{2} = \mathrm{O}(n^2)$ edges, M' makes $\mathrm{O}(n^2)$ simulations of M, each on an input of size polynomial in n. Thus M' runs in poly-time in n.

2. This exercise can be solved using a binary search algorithm similar to the one described in Exercise 2.4.1. Given a poly-time DTM M for FACTORING, we construct a poly-time DTM M' for TFACTORING as follows: Let

$\alpha := \langle m \rangle$ be an input for M'. M' first calls M on input $\langle m, m \rangle$. If the outcome is 'no', m is prime and the factorization of m is available. Otherwise, M' calls M on $\langle m, \lfloor m/2 \rfloor \rangle$. If the outcome is 'yes', M' calls M on $\langle m, \lfloor m/4 \rfloor \rangle$, else on $\langle m, \lfloor 3m/4 \rfloor \rangle$. Proceeding in this way M' determines the smallest (non-trivial) divisor p_1 of m. Clearly, p_1 has to be prime. M' determines the multiplicity e_1 of p_1 in m. If $m = p_1^{e_1}$, M' is done, else it repeats the above steps on $m/p_1^{e_1}$.

(Notice that m is composite, if and only if it has a divisor $\leqslant \lfloor \sqrt{m} \rfloor$. This observation allows us to make the above search somewhat more efficient. For the time being, we are not bothered by efficiency issues anyway.)

3. Since the total number of non-empty subsets of $\{a_1, \ldots, a_n\}$ is $2^n - 1$, by the pigeon-hole principle there exist at least two (distinct) subsets S' and T' having the same sum. Let $U := S' \cap T'$. Take $S := S' \setminus U$ and $T := T' \setminus U$. Clearly, S and T are disjoint, non-empty and continue to have the same sum.