Chapter 3 : Space complexity : Solutions of the exercises

Section 3.1

1. **(a)** Let L be a PSPACE-hard language. For any $A \in \mathrm{NP} \subseteq \mathrm{PSPACE}$ we then have $A \leqslant_P L$.

(b) We already know $\mathrm{NP} \subseteq \mathrm{PSPACE}$. To prove the other inclusion take any $L \in \mathrm{PSPACE}$. Let L' be an NP-complete language. By hypothesis, L' is PSPACE-hard too, i.e., $L \leqslant_P L'$. But then this reduction followed by a poly-time nondeterministic algorithm for L' solves L in nondeterministic poly-time, i.e., $L \in \mathrm{NP}$.

2. Use constructions similar to those in Exercise 2.1.4(a).

3. It is easy to see that $\mathrm{DTM}_{\text{IN-PLACE}} \in \mathrm{PSPACE}$: Given $\langle M, \alpha \rangle$ one may simulate M on α. Let $n := |\alpha|$. If M ever tries to move out of the first n cells or runs too long (i.e., for more that $sng^{f(n)}$ steps, where s the number of states and g the number tape symbols of M), then reject. Otherwise output M's decision on α.

In order to prove the PSPACE-hardness of $\mathrm{DTM}_{\text{IN-PLACE}}$ take any $L \in \mathrm{PSPACE}$. Let M be a DTM for L which runs in space cn^k for some constants c and k. Convert $\alpha \in \Sigma^*$ to $\langle M, \alpha \sqcup^{c|\alpha|^k - |\alpha|} \rangle$. Clearly, $\alpha \in L$ if and only if M accepts the padded string $\alpha \sqcup^{c|\alpha|^k - |\alpha|}$ in place.

4. An LBA is *almost* the same as an in-place NTM. By an argument similar to that in the previous exercise one can show that A_{LBA} is in $\mathrm{NSPACE}(n)$ and so by Savitch's theorem in $\mathrm{SPACE}(n^2)$.

5. Let $\langle c \rangle$ be a configuration of the generalized tic-tac-toe game with the next turn by X. One may easily check the validity of c in poly-space (in fact, poly-time): there must be even number of occupied cells with equal number of the two markers. One may additionally check if the game has already ended by checking five same consecutive markers (there are $\mathrm{O}(n^2)$ positions to check). If there are five consecutive markers of both O and X or only of O, reject. If there are five consecutive markers only of X, accept. Finally assume that c is a valid unfinished configuration and we want to check if Player X has a winning strategy from c. If there is no unoccupied cell, reject. Otherwise repeat the following for each of the unoccupied cells in c's board:

Put the marker X on the unoccupied cell, giving configuration c'. If c' is a winning configuration for X, accept. If the board is full in c', reject. Otherwise generate from c' the configurations $c''_1, \ldots, c''_k$ by putting exactly one marker O in an unoccupied cell in c'. Make a recursive call with each c''_i (one by one and reusing space). If all of these recursive calls output 'accept', accept and halt. Otherwise look at another (unexplored) candidate for c' and repeat.

If all candidates for c' result in non-acceptance, reject and halt. Note that since we are talking about a *winning strategy*, a 'draw' is treated as a 'defeat' (for Player X).

Each recursion requires storing a few (three) board positions (plus some pointers and some constant number of tape cells). The depth of the recursion is $\mathrm{O}(n^2)$. Thus $\mathrm{O}(n^4)$ space is sufficient to decide GT and the input size is $\mathrm{O}(n^2)$, so $\mathrm{GT} \in \mathrm{PSPACE}$.

Section 3.2

1. For union and intersection use strategies as in Exercise 2.1.4(b).

Kleene closure is a bit tricky. Take any $L \in \mathrm{NL}$. Our task is to show that $L^* \in \mathrm{NL}$ too. Let N be a log-space NTM for L. Take $\alpha \in \Sigma^*$. If $\alpha = \epsilon$, accept and halt. Otherwise, nondeterministically select a prefix β of α with $|\beta| \geqslant 1$. Call γ the remaining part of α, i.e., $\alpha = \beta\gamma$. Simulate N on β. If N accepts, recursively check if $\gamma \in L^*$. Else this branch of computation rejects.

If $\alpha \in L^*$, then we have $\alpha = \beta_1\beta_2 \ldots \beta_k$ with each β_i non-empty and in L. Thus some nondeterministic choices will reveal $\beta_1, \beta_2, \ldots, \beta_k$ in succession. On the other hand, if $\alpha \notin L^*$, for any non-empty prefix β of $\alpha = \beta\gamma$ we have either $\beta \notin L$ or $\gamma \notin L^*$. Thus all branches of computation reject.

Let us now investigate the (nondeterministic) space complexity of our algorithm. The splitting of α in β and γ can be effected by a constant number of pointers. Checking if $\beta \in L$ can be done in nondeterministic logarithmic space. If $\beta \notin L$, the branch terminates, else computation restarts on γ. It is not necessary to store β for the computation on γ.

2. **(a)** Maintain a counter initialized to zero. Read the input sequentially from left to right. When a left parenthesis is read from the input, increment the counter, and when a right parentheses is read, decrement the counter. If the counter ever becomes negative or is not restored to zero at the end of the input, reject. Otherwise accept.

(b) Let α be an instance for this problem. If $\alpha = \epsilon$, accept and halt. Otherwise repeat a procedure as in Part (a), identifying [as (and] as). If this stage results in rejection, then reject and halt. Otherwise repeat the following for each symbol a in the input (sequentially from left to right):

If a is) or], skip it. Otherwise a is a left delimiter ((or [). Using a counter (as in Part (a)) find out the matching right delimiter b. Here also we identify (with [and) with]. If b is not found, or if a and b are of different types (paren and bracket, or bracket and paren), reject.

If not rejected so far, accept.

3. I show that $\overline{\text{BIPARTITE}} \in \text{NL}$. Since $\text{NL} = \text{coNL}$, the desired result follows. An undirected graph G is *not* bipartite, if and only if G contains an odd cycle. Given G, we then search for odd cycles nondeterministically using log-space.

First nondeterministically choose a vertex $u \in V(G)$ and remember u till the end of all branches of computation originating from the selection of u. Also maintain a current vertex v and a count i. Initially set $v := u$ and $i := 0$. While $i < m$ (where m is the number of vertices in G) repeat:

If v has degree 0, reject, else nondeterministically choose an edge $(v, v') \in E(G)$. Increment i and set $v := v'$. If i is odd and $v = u$, accept. Otherwise, repeat the loop with the new v. If the loop terminates (i.e., if $i \geqslant m$), reject.

It is clear that this nondeterministic algorithm detects odd cycles in G. On the other hand, if G does not contain an odd cycle, all branches of computation reject. We need to provide storage only for the vertices u, v, v' and for the counter i. This can be achieved in log-space only.

4. The idea is similar to the proof of the theorem: "If $A \leqslant_L B$ and $B \in \text{L}$, then $A \in \text{L}$". For the computation of $(g \circ f)(\alpha) = g(f(\alpha))$, don't compute the full of $f(\alpha)$, because that may require super-log (as high as poly) space. Whenever a new symbol of $f(\alpha)$ is required for the computation of g, recompute (only) that symbol from α.

5. Since $\text{NL} = \text{coNL}$, it suffices to show that $\overline{\text{2SAT}}$ in NL-complete. In order to prove $\overline{\text{2SAT}} \in \text{NL}$, construct the graph G as in Exercise 2.2.3. Convince yourself that this construction can be done in (deterministic) log-space. Now nondeterministically choose a vertex x in G. Use an NL algorithm for PATH to check if G contains both $x, \bar{x}$ and $\bar{x}, x$ paths. If both paths exist, accept, else reject. The correctness of this algorithm is established by the claim in Exercise 2.2.3.

In order to prove the NL-hardness of $\overline{\text{2SAT}}$, I show that $\text{PATH} \leqslant_L \overline{\text{2SAT}}$. Given G, s, t we have to construct a 2-cnf formula ϕ such that G has an s,t-path if and only if ϕ is unsatisfiable. Let $V(G) = \{s, t, y_1, \ldots, y_m\}$. The resulting formula ϕ consists of $m+1$ variables $x, y_1, \ldots, y_m$. The vertex s is identified (relabeled) with x and the vertex t with $\bar{x}$. The clauses of ϕ will be $x \vee x$ and $\bar{u} \vee v$ for every edge (u, v) of G. That completes the construction of ϕ.

In order to show that this construction works, first assume that G has an s,t-path. Let this path be $s, u_1, \ldots, u_k, t$. Then ϕ contains the clauses $(\bar{x} \vee u_1), (\bar{u}_1 \vee u_2), \ldots, (\bar{u}_k, \bar{x})$. If we assign $x = 1$, at least one of these clauses is not satisfied (easy check). On the other hand, if $x = 0$, then the clause $(x \vee x)$ of ϕ is not satisfied. So ϕ is zero for any assignment of $x, y_1, \ldots, y_m$.

Conversely, suppose that G contains no s,t-path. Let U be the set of vertices in G reachable from s, V the set of vertices from which t is reachable and $W := V(G) \setminus (U \cup V)$. By hypothesis and construction, U, V, W are pairwise disjoint, and there are no edges from U to $V \cup W$ or from $U \cup W$ to V. Let me assign the true value to every variable in $U \cup W$ (including x) and the false value to every variable in V (including the *literal* $\bar{x}$). It is now simple to check that ϕ is satisfied for this truth assignment.

6. I first show that $\overline{\text{STRONGLY-CONNECTED}}$ is in NL. Nondeterministically choose a pair of vertices s, t of the input graph G and output the decision of an NL algorithm for $\overline{\text{PATH}}$ on the input $\langle G, s, t \rangle$.

Next I reduce PATH to STRONGLY-CONNECTED. Let $\langle G, s, t\rangle$ be an instance for PATH. Convert it to a graph G' as follows. First copy G to G' and then for every $u \in V(G)$ add the edges (u, s) and (t, u) (if not already present in G). Avoid loops, if they are aesthetically unpleasant to you.

Assume that G has an s, t-path; call it P. Take any two distinct vertices $u, v \in V(G') = V(G)$. If both u and v are outside P, then the edge (u, s), the path P and the edge (t, v) constitute an u, v-path in G'. If u is outside P and v is on P, then the edge (u, s) and the s, v-subpath of P gives a u, v-path in G'. If u is on P and v is outside P, use the u, t-subpath of P and the edge t, v. Finally, consider that u and v are both on P. If u appears earlier than v in P, use the u, v-subpath of P. If v appears earlier than u, use the u, t-subpath of P and then the edge (t, v).

Conversely, if G does not contain an s, t-path, neither does G', since all the edges added to G (to manufacture G') are either into s or out of t.

7. In order to see TQBF is PSPACE-complete under log-space reduction look at Papadimitriou (Theorem 19.1). The reduction TQBF (actually FORMULA-GAME) to GA, as we did in the class, is clearly doable in log-space.

If any of these languages is in NL, it is in $\mathrm{SPACE}(\lg^2 n)$ (by Savitch's theorem) and so in PolyL. By the previous result every language in PSPACE is then in PolyL, implying that PolyL = PSPACE, a contradiction.