Complexité avancée - TD 1

Benjamin Bordais

September 23, 2020

Exercise 1: One-minute-long exercise
Prove that any language $L \subset \{0,1\}^*$ that is neither empty nor $\{0,1\}^*$ is hard for NL for polynomial-time reductions.

Solution Consider such a language L with $w_y \in L$ and $w_n \notin L$ and an arbitrary problem L' in NL. We know that $\mathsf{NL} \subseteq P$. We modify a TM running in polynomial time deciding P by setting the output to w_y instead of accepting and w_n instead of rejecting.

Exercice 2: Graph representation and why it does not matter. Let $\Sigma = \{\mathtt{0}, \mathtt{1}, /, \bullet, \#\}$ with $\#$ the end-of-word symbol. For a directed graph $G = (V, E)$ with $V = [0, n-1]$ for some $n \in \mathbb{N}$ and $E \subseteq V \times V$, we consider the following two representations of G by a word in Σ^*:

- By its adjacency matrix $m_G \in \Sigma^*$:

$$m_G \overset{\text{def}}{=} m_{0,0}m_{0,1}\ \dots m_{0,n-1} \bullet \dots \bullet m_{n-1,0} \dots m_{n-1,n-1}\#$$

  where for all $0 \leq i,j < n$, $m_{i,j}$ is $\mathtt{1}$ if $(i,j) \in E$, $\mathtt{0}$ otherwise.

- By its adjacency list $l_G \in \Sigma^*$:

$$l_g \overset{\text{def}}{=} k_0^0 / \dots / k_{m_1}^0 \bullet \dots \bullet k_0^{n-1} / \dots / k_{m_{n-1}}^{n-1}\#$$

  where for all $0 \leq i < n$, $k_0^i, \dots, k_{m_i}^i$ are binary words listing the (codes of) right neighbors of vertex i.

1. Show that it is possible to check in logarithmic space that a word $w \in \Sigma^*$ is a well-formed description of a graph (for any of the two representations).
2. Describe a logarithmic space bounded deterministic Turing machine taking as input a graph G, represented by its adjacency matrix, and computing the adjacency list representation of G.

Solution. We make two preliminary observations.
– Firstly, it is easy to implement a binary counter on an empty worktape T_B. E.g., for little-endian representation we increment by reading T_B left to right, replacing `1`s by `0`s, stopping at the first `0` and replacing it with a `1`. If we reach the end of T_B without seeing a `0`, we add a `1` to the right. Then, we reset the TM head to the start of the tape, and the incrementation is over.
– Secondly, comparing two numbers written in binary (with no extra `0`s on the right) just

needs logspace. If the counters are on different worktapes one needs no extra space and only linear time: one checks that the counters have different sizes and deduces which is larger, and if they have same size one checks if they coincide or reading from the right, finds the first bit where they differ and deduces which counter is the largest. If the counters are on the same worktape (usually the input tape) one just needs logarithmic space, e.g., to store pointers to the extremities of the counters before performing the comparison, or to copy them on different auxiliary worktapes and reuse the previous method.

In the following, N will denote the size of the input.

1. First, consider the adjacency matrix case. Checking that $w \in \Sigma^*$ consists in sequences of 0s and 1s separated by the symbol $\bullet$ and ending with the symbol $\#$ is straightforward[1]. But we also have to check that between any two consecutive $\bullet$s (and also before the first $\bullet$) there are exactly $m+1$ symbols, where m is the number of $\bullet$s. To do so, we may use an additional tape T_B as a counter, incrementing it each time the symbol $\bullet$ or $\#$ is seen, so that in the end T_B stores $m+1$. Note that $m+1$ written in binary takes $O(\log N)$ bits. Then, we consider another working tape T_W where, for each sequence in $\{0,1\}^*$ we increment a counter until we reach the next $\bullet$ symbol. Then, we can check that it is equal to the number $m+1$ on T_B.

   The case of adjacency list is analogous, but instead of checking the number of symbols between two $\bullet$s, we have to check that every number written (between /) is smaller than the value of the counter written on the tape T_B.

2. We first use the Turing Machine we described in the previous question to check that the word considered is well-formed. Then, we consider a Turing Machine M with a tape T_i where a counter will loop between 0 and $k-1$ (where k is the number of $\bullet$ symbol plus 1) assuming the word we consider is well formed. The counter i is incremented whenever a new symbol 0 or 1 is seen and reinitialized at each $\bullet$ symbol. Furthermore, whenever $\bullet$ is seen, $\bullet$ is written on the output tape and whenever a 1 is seen, the counter i is copied on the output tape, preceded by symbol / if it is not the first time the counter i is copied since the last time a $\bullet$ symbol was written. The space taken by this TM is in $O(\log N)$ since we only use a counter whose value, written in binary is at most k and the logarithmic space taken by the TM described in the first question. Then, we conclude with the speedup theorem.

Exercise 3: A few NL-complete problems

Show that the following problems are NL-complete for logspace reductions (you may use the fact that REACH is NL-hard for logspace reductions):

1. Deciding if a non-deterministic automaton $\mathcal{A}$ accepts a word w.

2. Deciding if a directed graph has a cycle.

Solution

1. The problem is in NL as we can simply guess the path in the automaton corresponding to the word and arrive in an accepting state. To be in logspace, we do not however guess the whole path at once, but guess transition by transition while keeping a counter to the current state.

[1] This is a "regular" constraint and any regular language is in $\mathsf{TIME}(n) \cap \mathsf{SPACE}(0)$.

Then, to establish NL-hardness, given a graph G, a starting node and an ending node, we label all the edges with the lettre ϵ in order to create an automaton $\mathcal{A}$, with initial state (resp. final) the starting (resp. ending) node. Now, REACH is equivalent to whether $\mathcal{A}$ accepts ϵ, we thus reduced REACH to our problem, which is then NL-complete.

2. The problem is in NL, given G we guess an edge (x, y) of the cycle and run REACH on (G, y, x).

   Consider now the NL-hardness. Given an instance of REACH (G, s, t), we may create G' by first adding an edge between t and s, creating a cycle inside G' if s and t are connected in G. This is not enough, because G may have other cycles and the equivalence would not hold. Thus, we must first eliminate all the cycles in G. Let m be the number of nodes in G. We create m copies of G, which can be seen as m levels. For every edge from i to j in G, we draw an edge from node i at each level to node j at the next level. Additionally, we draw an edge from each node i at each level to node i at the next level. We call s' the s of the first level, and t' the t of the last level. Now, there is a path from s to t in G if and only if there is path from s' to t' in G'. Moreover, in G' path are only "going up" into the levels, so there cannot be any circle. Thus if we add an edge from t' to s', we now have that there is a path from s to t in G if and only if there is a cycle in G'. We can check that this operation can be done in logarithmic space.