CMSC 28100-1 / MATH 28100-1
Introduction to Complexity Theory
Fall 2017 – Homework 6
Solution

November 9, 2017

Exercise 1. Let $\mathbf{P}$ be the complexity class of all languages decidable in polynomial time, that is, we have $\mathbf{P} = \bigcup_{k\geqslant 1} \mathrm{TIME}(n^k)$. Prove that $\mathbf{P} \neq \mathrm{TIME}(n^3)$. (Hint: use the Time Hierarchy Theorem.)

Solution. Since n^4 is fully time constructible and $\lim_{n\to\infty} n^3 \log n^3/n^4 = 0$, by the Time Hierarchy Theorem, we know that there exists a language $L \in \mathrm{TIME}(n^4)\setminus\mathrm{TIME}(n^3)$. Since $\mathrm{TIME}(n^4) \subseteq \mathbf{P}$, it follows that $\mathbf{P} \supsetneq \mathrm{TIME}(n^3)$. $\triangleleft$

Exercise 2. The number of *head reversals* of a single-tape Turing machine M on input x is the number of times the tape head changes direction, that is, it was moving left and it moves right, or vice-versa. (Not moving does not count as reversing direction, but e.g. moving right, not moving then moving left does.)

Let $\mathrm{REVERSAL}(r(n))$ denote the class of languages decidable by Turing machines that on inputs of length n use at most $r(n)$ head reversals.

A function $R\colon \mathbb{N} \to \mathbb{N}$ is said to be *fully reversal constructible* if there exists a Turing machine M_R that always halts and takes exactly $R(n)$ head reversals on inputs of length n.

Prove the following head reversal hierarchy theorem: if $\lim_{n\to\infty} r(n)^2/R(n) = 0$ and R is fully reversal constructible, then there exists a language in $\mathrm{REVERSAL}(kR(n))$ that is not in $\mathrm{REVERSAL}(r(n))$ for some fixed $k > 0$. You can assume the following fact: if a Turing machine M on input w requires at most r head reversals, then the simulation of M on w can be done by a Universal Turing Machine using less than r^2 head reversals. (Hint: mimic the proof of the Time Hierarchy Theorem.)

Solution. Let us first prove a small lemma.

Lemma 2.1. *Let $R\colon \mathbb{N} \to \mathbb{N}$. If M is a Turing machine that uses at most $R(n)$ head reversals on inputs of length n, then there exists a Turing machine that uses at most $O(R(n))$ head reversals on inputs of length n and such that $L(M) = L(N)$ and N always halts.*

Proof. Without loss of generality, we assume that M always moves its head (otherwise, we can recode M to do so).

Let q be the number of states of M and let γ be the size of the tape alphabet of M. We construct N to mimic M, except for two differences:

- Whenever M writes a blank B, we make N write a new symbol $\widetilde{B}$. This new symbol is treated as a blank for the execution of M, but is a different symbol that allows us to identify blank parts of the tape that were not yet visited by N (as these will have the true blank symbol B rather than $\widetilde{B}$).

- N has an extra counter C (on an extra tape) that counts how many times a (true) blank symbol has been read since the last head reversal; if this counter C ever gets to $q+2$, the machine N halts and rejects.

Note that since q is constant (i.e., does not depend on the input) and since C always gets reset on every reversal, it follows that N uses at most $K \cdot R(n) = O(R(n))$ head reversals on inputs of length n (where K is the number of head reversals required to increase a counter from 0 to $q+2$ and comparing it each time with $q+2$ and resetting it at the end).

Let us prove that N always halts.

Clearly if M halts on a certain input x, then N will also halt in x. So suppose x is an input in which M does not halt. Since M can only use $R(|x|)$ head reversals on input x, we know there is some time t, one of two things happen:

- M keeps moving right after t.
- M keeps moving left after t.

If the first of the above happens, then N will eventually reach the (true) blank symbols B on the right end of the tape and the counter C will eventually reach $q+2$, making N halt.

If the second of the above happens, then N will eventually reach the (true) blank symbols B on the left endo of the tape and the counter C will eventually reach $q+2$, making N halt.

Therefore N always halts.

Let us now prove that $L(N) = L(M)$. If M rejects x, then N clearly rejects x (either because N sees this in the execution of M or because the counters make it reject).

Let us prove that if N rejects x, then $x \notin L(M)$.

If N rejects x because the execution of M by N rejected x, then clearly M rejects x, so suppose this is not the case.

Then N must reject x because of the counter C.

But if N rejects x because of the counter C, then we know that in the execution of M by N, we at some point read $q+2$ (true) blank symbols without making any head reversals between them. Then there is a direction $D \in \{L, R\}$ such that along the time when these $q+2$ blank symbols are read, the machine M always moves its head in the direction D.

Let us consider the case $D = R$ (the case $D = L$ is symmetric). Let $t_1, \ldots, t_{q+2}$ be the times when these (true) blank symbols are read.

We claim that at time t_2, the head is at the right end of the tape. Indeed, for this to not happen, at time t_1 the head must have been at the left end of the tape (recall that true blank symbols are only present at the ends of the tape), but the head cannot move left and it replaces the true blank symbol with $\widetilde{B}$, so the next true blank symbol (i.e., the one read at time t_2) must be at the right end of the tape.

But then we know that between these times there must be a repeated state r among the times $t_2, \ldots, t_{q+2}$, i.e., the computation must be of the form

$$wrB \vdash zrB,$$

for some strings w and z between two times t_i and t_j and in the computation above no left movements are performed. But this means that between times t_j and $2t_j - t_i$, the same computation will take place (as symbols to the left of the head become irrelevant), hence M loops forever, which implies $x \notin L(M)$.

Therefore $L(N) = L(M)$ as desired. ■

Fix a computable encoding of all Turing machines such that each machine has infinitely many encodings and such that a Universal Turing Machine U can simulate machines in this encoding in a way that if M requires r reversals on input w, then U requires r^2 reversals to simulate M on w (including the number of reversals it takes to "decode" x into M).

Let M_R be a Turing machine that uses exactly $R(n)$ head reversals on inputs of length n (and always halts).

Consider the following algorithm.

Algorithm 2.1: Diagonal head reversal language for r.

```
1 On input $x$, let $M$ be the Turing machine encoded by $x$.
2 Copy input $x$ to an extra tape and rewind the heads to the beginning of the two copies.
3 Run $M_R$ in the extra copy of $x$ while writing a 1 in an extra tape $C$ whenever $M_R$ does a
    head reversal.
4 Simulate $M$ on input $x$ while erasing a 1 in $C$ whenever one needs to do a head reversal,
    halting the simulation if there are no 1's left in $C$.
5 if The simulation is halted because $C$ is empty then Reject.
6 if $M$ accepts then Reject.
7 else Accept.
```

Let us estimate the number of head reversals that the algorithm above takes. Copying x then rewinding the heads take 2 head reversals. Running M_R takes $R(|x|)$ reversals. To start erasing the 1's on C, we use one head reversal. The simulation of M on x takes at most $R(|x|)$ head reversals since we bound the simulation. Hence the total number of head reversals is

$$3 + 2R(|x|) \leqslant 5R(|x|).$$

Even though the algorithm above does not necessarily halt on every input, by Lemma 2.1, we know that there is a Turing machine N that decides (i.e., always halts) precisely the language recognized by the algorithm above and uses at most $O(5R(n))$ reversals on inputs of length n.

Let $L = L(N)$. We have just proved that $L \in \mathrm{REVERSAL}(O(R(n)))$.

We claim that $L \notin \mathrm{REVERSAL}(r(n))$. Suppose not, that is, suppose that there exists a Turing machine D that decides L and takes at most $r(n)$ reversals on inputs of length n.

Since $\lim_{n\to\infty} r(n)^2/R(n) = 0$, it follows that there exists a large enough encoding x of D such that $r(n)^2 < R(n)$, where $n = |x|$.

Consider the execution of the algorithm for the input x. Since D takes at most $r(n)$ head reversals to run on x and $r(n)^2 < R(n)$, it follows that the simulation of D runs completely.

If $x \in L$, then D accepts x, so the algorithm rejects x, hence $x \notin L$, a contradiction.

On the other hand, if $x \notin L$, then D rejects x so the algorithm accepts x, hence $x \in L$, a contradiction.

Therefore $L \in \mathrm{REVERSAL}(O(R(n))) \setminus \mathrm{REVERSAL}(r(n))$ as desired. $\triangleleft$

Exercise 3 (Based on Exercise 9.13 of Sipser's "Introduction to Theory of Computation")**.** Consider the function $\mathrm{pad}\colon \Sigma^* \times \mathbb{N} \to \Sigma^*\#^*$ that is defined as follows. Let $\mathrm{pad}(s,\ell) = s\#^j$, where $j = \max\{0, \ell - |s|\}$ and $|s|$ is the length of s. Thus $\mathrm{pad}(s,\ell)$ simply adds enough copies of the new symbol # to the end of s so that the length of the resulting string is at least ℓ. For any language A and function $f\colon \mathbb{N} \to \mathbb{N}$, define the language $\mathrm{pad}(A, f(m))$ as

$$\mathrm{pad}(A, f(m)) = \{\mathrm{pad}(s, f(|s|)) : s \in A\}.$$

(a) Prove that if $A \in \mathrm{TIME}(n^6)$, then $\mathrm{pad}(A, n^2) \in \mathrm{TIME}(n^3)$.

(b) Let Fac be the language

$$\text{Fac} = \{(x,y) : x, y \in \mathbb{N} \text{ are written in } \textit{binary} \text{ and } x \text{ has a non-trivial factor that is less or equal to } y\}.$$

A non-trivial factor of x is a positive divisor of x that is neither 1 nor x. Show that Fac can be decided in time $O(x(\log x)^2) = O(2^n n^2)$, where $n = |x|$ is the length of x. You can assume the following fact: given two integers x and y in binary, one can decide whether y is a factor of x in $O(n^2)$ steps, where n is the number of digits in x.

(c) Let UnaryFac be the language

$$\text{UnaryFac} = \{(x,y) : x, y \in \mathbb{N} \text{ are written in } \textit{unary} \text{ and } x \text{ has a non-trivial factor that is less or equal to } y\}.$$

Show that UnaryFac $\in \text{TIME}(n(\log n)^2)$, where n is the length of x. (The *unary* representation of an integer x is the string 1^x.)

(d) Show that $\text{pad}(\text{Fac}, 2^n) \in \text{TIME}(n(\log n)^2)$.

Solution. We start by solving item (a).

Let M_A be a Turing machine that decides A in time n^6.

Consider the following algorithm.

Algorithm 3.1: Algorithm for $\text{pad}(A, n^2)$.

```
1 On input $x$, check to see if $x = y\#^j$ for some $y$ that does not contain any $\#$.
2 **if** *$x$ is not of this form* **then** Reject.
3 **if** $|x| = |y|^2$ **then** Reject.
4 Erase all $\#$ from the input so that $y$ remains and position the head in its first character.
5 Run $M_A$ on $y$.
6 **if** *$M_A$ accepts* **then** Accept.
7 **else** Reject.
```

Clearly the algorithm above always halts and accepts precisely the language $\text{pad}(A, n^2)$.

Note now that if the input x has length $N = |x|$, then checking the formatting of x takes time at most $O(N^2)$ (the square gives us enough leeway to compute $|y|^2$).

If x is well-formatted, that is, it is of the form $x = y\#^j$ with $|x| = |y|^2$, then the execution of M_A takes time at most $|y|^6 = |x|^3 = N^3$.

Therefore the whole algorithm runs in time $O(N^3)$ and by linear acceleration, we know that there exists an algorithm that decides $\text{pad}(A, n^2)$ in time N^3.

For item (b), consider the following algorithm.

Algorithm 3.2: Algorithm for Fac

```
1 On input $(x, y)$, do the following.
2 Let $m \leftarrow \min\{x - 1, y\}$.
3 **for** $i \leftarrow 2$ **to** $m$ **do**
4     **if** *$i$ is a factor of $x$* **then** Accept.
5 Reject.
```

Clearly the algorithm above always halts and decides Fac.

Note that computing m takes time $O(\min\{\log x, \log y\})$, which is $O(\log x)$. Incrementing i costs $O(\log m)$ each time (since i is bounded by m). Testing if i is a factor of x can be done in time $O((\log x)^2)$, hence the total running time is

$$O(\log x) + m \cdot (O(\log m) + O((\log x)^2)),$$

which is $O(x(\log x)^2)$ since $m \leqslant x$.

For item (c), we could use Algorithm 3.2 preceded by a small computation that converts x and y from unary to binary, which can be done in time $O(x \log x)$ and $O(y \log y)$ by progressively incrementing a counter. However, this is not good enough because of y, so we slightly change this pre-computation so that each time the counter for the conversion of y is incremented, we compare the length of the current binary string with the length of the binary representation of x, stopping the conversion if it is longer. This converts y to the binary representation of

$$z = \min\{y, 2^{\lceil \log_2 x \rceil}\}$$

in time $O(x \log x)$.

Since we clearly have $(x, y) \in \text{Fac}$ if and only if $(x, z) \in \text{Fac}$, the resulting algorithm solves the problem in time $O(x(\log x)^2)$. By linear acceleration, we get $\text{UnaryFac} \in \text{TIME}(n(\log n)^2)$ where $n = |x|$ as desired.

Finally, let us solve item (d).

Let M_F be the Turing machine associated with Algorithm 3.2 and consider the following algorithm.

Algorithm 3.3: Algorithm for $\text{pad}(\text{Fac}, 2^n)$.

```
1 On input $w$, check to see if $w = z\#^j$ for some $z$ that does not contain any #.
2 if $w$ is not of this form then Reject.
3 if $|w| = 2^{|z|}$ then Reject.
4 Erase all # from the input so that $z$ remains and position the head in its first character.
5 Run $M_F$ on $z$.
6 if $M_F$ accepts then Accept.
7 else Reject.
```

Clearly the algorithm above decides the language $\text{pad}(\text{Fac}, 2^n)$.

Note now that if the input w has length $n = |w|$, then checking the formatting of w takes time at most $O(n(\log n))$ (the extra $\log n$ factor gives us enough leeway to compute 2^n).

If w is well-formatted, it is of the form $z\#^j$ where z is the encoding of (x, y) and the execution of M_F takes time $O(x(\log x)^2)$ which is $O(2^{|z|}|z|^2)$, which in turn is $O(n(\log n)^2)$.

Hence the whole algorithm takes time $O(n(\log n)^2)$ and by linear acceleration we get $\text{pad}(\text{Fac}, 2^n) \in \text{TIME}(n(\log n)^2)$. ◁

Exercise 4 (Undirected Hamiltonian Circuit is **NP**-complete)**.** Prove that the undirected Hamiltonian Circuit Problem UHC (i.e., the Hamiltonian Circuit Problem for undirected graphs) is **NP**-complete. You can assume that the Directed Hamiltonian Circuit Problem HC is **NP**-complete. (Hint: do a polynomial (many-to-one) reduction to HC. Remember to state what the input of your reduction is, what it produces and why this implies that UHC is **NP**-complete.)

Solution. Given a directed graph G, define the undirected graph G' by letting

$V(G') = V(G) \times \{-1, 0, 1\}$;
$E(G') = \{\{(v, -1), (v, 0)\} : v \in V(G)\} \cup \{\{(v, 0), (v, 1)\} : v \in V(G)\} \cup \{\{(v, 1), (w, -1)\} : (v, w) \in E(G)\}$;

that is, the graph G' has three copies indexed by $\{-1,0,1\}$ of each vertex of G, copies of the same vertex are connected as $-1 \sim 0 \sim 1$ (but not $-1 \sim 1$), and for every (directed) edge from v to w in G, we put an (undirected) edge between $(v,1)$ and $(w,-1)$.

We claim that G has a (directed) Hamiltonian Circuit if and only if G' has a (undirected) Hamiltonian Circuit.

Suppose $(v_0, v_1, v_2, \ldots, v_n)$ is a Hamiltonian Circuit of G ($v_0 = v_n$), then (since $(v_i, v_{i+1}) \in E(G)$ for every $i \in \{0,1,\ldots,n-1\}$) by our construction of G', the following is a circuit of G'.

$$((v_0,0),(v_0,1),(v_1,-1),(v_1,0),(v_1,1),(v_2,-1),(v_2,0),(v_2,1),\ldots,(v_n,-1),(v_n,0)).$$

Note also that every vertex of G' appears exactly once in the above except for $(v_0,0)=(v_n,0)$ (since every vertex of G appears exactly once in $(v_0,v_1,\ldots,v_n)$ except for $v_0=v_n$), hence this is a Hamiltonian Circuit in G'.

Suppose now that

$$C = ((v_0,b_0),(v_1,b_1),(v_2,b_2),\ldots,(v_m,b_m))$$

is a Hamiltonian Circuit of G'. Note that $m = |V(G')| = 3|V(G)|$. By possibly changing the initial vertex (and since all vertices of G' appear in the above), we may suppose that $b_0 = 0$.

Note that $(v_0,0)$ has exactly two neighbors in G', namely $(v_0,-1)$ and $(v_0,1)$. By possibly reversing the cycle (recall that G' is undirected), we may suppose that $b_1 = 1$.

Let us now prove by induction in $k \in \{0,1,\ldots,m\}$ that $b_k \equiv k \pmod 3$.

This is clearly true for $k \in \{0,1\}$, so suppose $k \geqslant 2$ and that the assertion holds for $k-1$ and $k-2$.

If $k \equiv 0 \pmod 3$, then we know that $b_{k-2} = 1$ and $b_{k-1} = -1$ by inductive hypothesis. We claim that $(v_k,b_k) = (v_{k-1},0)$. First note that since the only neighbors of $(v_{k-1},0)$ are $(v_{k-1},-1)$ and $(v_{k-1},1)$ both edges $\{(v_{k-1},0),(v_{k-1},-1)\}$ and $\{(v_{k-1},0),(v_{k-1},1)\}$ must appear in the circuit C and since $b_{k-2} = 1$ and C does not repeat vertices (except for the initial and final vertices, which are not $(v_{k-1},-1)$), it follows that the first edge must be between $(v_{k-1},-1)$ and (v_k,b_k), hence $(v_k,b_k) = (v_{k-1},0)$ as desired and we get $b_k = 0 \equiv k \pmod 3$.

Suppose now that $k \equiv 1 \pmod 3$, then we know that $b_{k-2} = -1$ and $b_{k-1} = 0$ by inductive hypothesis. Since the only neighbors of $(v_{k-1},0)$ are $(v_{k-1},-1)$ and $(v_{k-1},1)$, it follows that $(v_{k-2},b_{k-2}) = (v_{k-1},-1)$ and $(v_k,b_k) = (v_{k-1},1)$, hence $b_k = 1 \equiv k \pmod 3$.

Finally, suppose $k \equiv -1 \pmod 3$, then we know that $b_{k-2} = 0$ and $b_{k-1} = 1$ by inductive hypothesis. Since $(v_{k-1},1)$ has a unique neighbor (w,b) with $b=0$ (namely $(v_{k-1},0)$, which must be (v_{k-2},b_{k-2})) and no neighbors (w,b) with $b=1$, it follows that $b_k = 1$ (since C can only repeat vertices in the beginning and the end and k is not the end as it is not divisible by 3 whereas m is).

Therefore $b_k \equiv k \pmod 3$.

Since the only neighbors of a vertex of the form $(v,0)$ are $(v,-1)$ and $(v,1)$, this implies that C is actually of the form

$$C = ((w_0,0),(w_0,1),(w_1,-1),(w_1,0),(w_1,1),(w_2,-1),(w_2,0),(w_2,1),\ldots,(w_n,-1),(w_n,0)),$$

which implies by the construction of G' that

$$\widetilde{C} = (w_0,w_1,\ldots,w_n)$$

is a circuit of G. Furthermore, since every vertex of G' appears exactly once in C except for $(w_0,0) = (w_n,0)$, it follows that every vertex of G appears exactly once in $\widetilde{C}$ except for $w_0 = w_n$, i.e., the circuit $\widetilde{C}$ is a Hamiltonian Circuit in G.

Therefore we have proved that $G \in$ HC if and only if UHC.

Note now that G' can be constructed in polynomial time from G by the following algorithm.

Algorithm 4.1: Constructing G' from G.

```
1 Given the adjacency matrix $A$ of $G$, construct the adjacency matrix $A'$ of $G'$ as follows.
2 for $v \in V(G)$ do
3     $A'[(v,-1),(v,0)] \leftarrow A'[(v,0),(v,-1)] \leftarrow 1$.
4     $A'[(v,0),(v,1)] \leftarrow A'[(v,1),(v,0)] \leftarrow 1$.
5     for $w \in V(G)$ do
6         if $A[v,w] = 1$ then
7             $A'[(v,1),(w,-1)] \leftarrow A'[(w,-1),(v,1)] \leftarrow 1$.
8 return $A'$.
```

Hence we get HC $\leqslant_p$ UHC. Since HC is **NP**-complete, this implies that UHC is **NP**-hard since for every **NP**-problem P, we have $P \leqslant_p$ HC, hence $P \leqslant_p$ UHC as $\leqslant_p$ is transitive (explicitly, just compose the algorithm above with the algorithm that reduces P to HC).

To show that UHC is **NP**-complete, it remains to show that UHC is in **NP**. But indeed, a non-deterministic polynomial algorithm for UHC is the following. Given an undirected graph G, non-deterministically pick a sequence $C = (v_0, v_1, \ldots, v_n)$ such that $v_0 = v_n$ and $V(G) = \{v_0, v_1, \ldots, v_{n-1}\}$ and check whether it is a circuit in G; if it is, accept; otherwise, reject. $\triangleleft$