Complexity theory
Department of Mathematics and Statistics
Fall 2017
Exercise set 5

Read chapters 1.7–2.1 of the book.

Exercise 1. Consider a k-tape Turing machine running in time $T(n)$ and space $S(n)$ (with both T and S time-constructible). What can you say about the relationship between T and S, i.e., can you bound T in terms of S and vice versa?

Exercise 2. Look at the proof of Theorem 1.9 in chapter 1.7 of the book.

(a) Looking at just one tape. If the machine starts in a position with the head pointing to the leftmost symbol of the input, what are the shifts needed to simulate 7 steps to the right? How does the tape look after each shift and how many moves were required to make the shift?
(b) How many moves are needed in general to make a shift of index i?
(c) How often (at worst) is a shift of index i needed? What is this 'worst case'?
(d) Using your observations, calculate the time needed to simulate a machine running in time T.

Exercise 3. (1.10 from the book) Consider the following simple programming language. It has a single infinite array `A` of elements in $\{0, 1, \square\}$ (initialized to $\square$) and a single integer variable `i`. A program in this language contains a sequence of lines of the following form:

label: `If A[i] equals` σ `then` *cmds*

where $\sigma \in \{0, 1, \square\}$ and *cmds* is a list of one of more of the following commands

(a) `Set A[i] to` τ, where $\tau \in \{0, 1, \square\}$,
(b) `Goto` *label*,
(c) `Increment i by one`,
(d) `Decrement i by one`, and
(e) `Output` b `and halt`, where $b \in \{0, 1\}$.

A program is executed on an input $x \in \{0, 1\}^n$ by placing the ith bit of x in `A[i]` and then running the program following the obvious semantics.

Prove that for every functions $f : \{0, 1\}^* \to \{0, 1\}$ and (time-constructible) $T : \mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a program in this language, then $f \in \mathbf{DTIME}(T(n))$.

Exercise 4. (1.12 from the book) A *partial* function from $\{0, 1\}^*$ to $\{0, 1\}^*$ is a function that is not necessarily defined on all its inputs. We say that a Turing machine M computes a partial function f if for every x on which f is defined, $M(x) = f(x)$ and

for every x on which f is not defined M gets into an infinite loop when executed on input x. If S is a set of partial functions, we define f_S to be the Boolean function that on input α outputs 1 iff M_α computes a partial function in S. *Rice's Theorem* says that for every nontrivial S (i.e., neither empty nor the set of all partial functions computable by some Turing machine), the function f_s is not computable.

(a) Show that Rice's Theorem yields an alternative proof that the function `HALT` is not computable.
(b) Prove Rice's Theorem.

Exercise 5. (part of 1.14 from the book) Prove that the following languages/decision problems on graphs are in **P**. (You may pick either the adjacency matrix or adjacency list representation for graphs; it will not make a difference. Can you see why?)

(a) `CONNECTED`, the set of connected graphs. A graph is *connected* if there is a path between any two distinct vertices.
(b) `TREE`, the set of all trees. A graph is a *tree* if it is connected and contains no cycles, i.e. a graph is a tree if any two distinct vertices are connected by exactly one path that repeats no vertices.