math is hard, but you don't have to do it alone!

October 4, 2017
Please send comments and corrections to Anil Ada (aada@cs.cmu.edu).
Foreword

These notes are based on the lectures given by Anil Ada and Ariel Procaccia for the Fall 2017 edition of the course 15-251 “Great Ideas in Theoretical Computer Science” at Carnegie Mellon University. They are also closely related to the previous editions of the course, and in particular, lectures prepared by Ryan O’Donnell.

WARNING: The purpose of these notes is to complement the lectures. These notes do not contain full explanations of all the material covered during lectures. In particular, the intuition and motivation behind many concepts and proofs are explained during the lectures and not in these notes.

There are various versions of the notes that omit certain parts of the notes. Go to the course webpage to access all the available versions.

In the main version of the notes (i.e. the main document), each chapter has a preamble containing the chapter structure and the learning goals. The preamble may also contain some links to concrete applications of the topics being covered. At the end of each chapter, you will find a short quiz for you to complete before coming to recitation, as well as hints to selected exercise problems.

Note that some of the exercise solutions are given in full detail, whereas for others, we give all the main ideas, but not all the details. We hope the distinction will be clear.
Acknowledgements

The course 15-251 was created by Steven Rudich many years ago, and we thank him for creating this awesome course. Here is the webpage of an early version of the course:
http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15251-s04/Site/.
Since then, the course has evolved. The webpage of the current version is here:
http://www.cs.cmu.edu/~15251/.

Thanks to the previous instructors of 15-251, who have contributed a lot to the development of the course: Victor Adamchik, Luis von Ahn, Anupam Gupta, Venkatesan Guruswami, Bernhard Haeupler, John Lafferty, Ryan O’Donnell, Ariel Procaccia, Daniel Sleator and Klaus Sutner.

Thanks to Eric Bae, Seth Cobb, Teddy Ding, Ellen Kim, Aditya Krishnan, Xinran Liu, Matthew Salim, Ticha Sethapakdi, Vanessa Siriwatlothakul, Natasha Vasthare, Jenny Wang, Ling Xu, Ming Yang, Stephanie You, Xingjian Yu and Nancy Zhang for sending valuable comments and corrections on an earlier draft of the notes. And thanks to Rosie Sun for sending valuable comments and corrections on the current draft.
Contents

1 Strings and Encodings 1
2 Deterministic Finite Automata 5
3 Turing Machines 11
4 Countable and Uncountable Sets 17
5 Undecidable Languages 21
6 Time Complexity 25
7 The Science of Cutting Cake 31
8 Introduction to Graph Theory 33
Chapter 1

Strings and Encodings
Exercise 1.1 (Structural induction on words).
Let language $L \subseteq \{0, 1\}^*$ be recursively defined as follows:

- $\epsilon \in L$;
- if $x, y \in L$, then $0xy0 \in L$.

Show, using (structural) induction, that for any word $w \in L$, the number of 0’s in $w$ is exactly twice the number of 1’s in $w$.

Solution. Let $0(w)$ denote the number of 0’s in $w$ and let $1(w)$ denote the number of 1’s in $w$. Given $L$ as defined above, the question asks us to show that for any $w \in L$, $0(w) = 2 \cdot 1(w)$. We will do so by structural induction.¹

The base case corresponds to $w = \epsilon$, and in this case, $0(w) = 1(w) = 0$, and therefore $0(w) = 2 \cdot 1(w)$ holds.

To carry out the induction step, consider an arbitrary word $w \neq \epsilon$ in $L$. Then by the definition of $L$, we know that there exists $x$ and $y$ in $L$ such that $w = 0xy0$. Furthermore, by induction hypothesis,

$$0(x) = 2 \cdot 1(x) \quad \text{(1.1)}$$

and

$$0(y) = 2 \cdot 1(y). \quad \text{(1.2)}$$

We are done once we show $0(w) = 2 \cdot 1(w)$. We establish this via the following chain of equalities:

$$0(w) = 2 + 0(x) + 0(y) \quad \text{since } w = 0xy0$$
$$= 2 + 2 \cdot 1(x) + 2 \cdot 1(y) \quad \text{by (1.1) and (1.2)}$$
$$= 2 \cdot (1 + 1(x) + 1(y))$$
$$= 2 \cdot 1(w).$$

Exercise 1.2 (Can you distribute star over intersection?).
Prove or disprove: If $L_1, L_2 \subseteq \{a, b\}^*$ are languages, then $(L_1 \cap L_2)^* = L_1^* \cap L_2^*$.

Solution. We disprove the statement by providing a counterexample. Let $L_1 = \{a\}$ and $L_2 = \{aa\}$. Then $L_1 \cap L_2 = \emptyset$, and so $(L_1 \cap L_2)^* = \{\epsilon\}$. On the other hand, $L_1^* \cap L_2^* = L_2^* = \{aa\}^*$.

Exercise 1.3 (Can you interchange star and reversal?).
Is it true that for any language $L$, $(L^*)^R = (L^R)^*$? Prove your answer.

Solution. We will prove that for any language $L$, $(L^*)^R \subseteq (L^R)^*$. To do this, we will first argue $(L^*)^R \subseteq (L^R)^*$ and then argue $(L^R)^* \subseteq (L^*)^R$.

To show the first inclusion, it suffices to show that any $w \in (L^*)^R$ is also contained in $(L^R)^*$. We do so now. Take an arbitrary $w \in (L^*)^R$. Then for some $n \in \mathbb{N}$, $w = (u_1 u_2 \ldots u_n)^R$, where $u_i \in L$ for each $i$. Note that $w = (u_1 u_2 \ldots u_n)^R = u_n^R u_{n-1}^R \ldots u_1^R$, and $u_i^R \in L^R$ for each $i$. Therefore $w \in (L^R)^*$.

To show the second inclusion, it suffices to show that any $w \in (L^R)^*$ is also contained in $(L^*)^R$. We do so now. Take an arbitrary $w \in (L^R)^*$. This means that for some $n \in \mathbb{N}$, $w = v_1 v_2 \ldots v_n$, where $v_i \in L^R$ for each $i$. For each $i$, define $u_i = v_i^R$ (and so $u_i^R = v_i$). Note that each $u_i \in L$ because $v_i \in L^R$. We can now rewrite $w$ as $w = u_1^R u_2^R \ldots u_n^R$, which is equal to $(u_n u_{n-1} \ldots u_1)^R$. Since each $u_i \in L$, this shows that $w \in (L^*)^R$.

Since we have shown both $(L^*)^R \subseteq (L^R)^*$ and $(L^R)^* \subseteq (L^*)^R$, we conclude that $(L^*)^R = (L^R)^*$.

¹This means that implicitly, the parameter being inducted on is the number of applications of the recursive rule to create a word $w$ in $L$. 

2
Exercise 1.4 (Unary encoding of integers).
Describe an encoding of \( \mathbb{Z} \) using the alphabet \( \Sigma = \{1\} \).

Solution. Let \( \text{Enc} : \mathbb{Z} \to \{1\}^* \) be defined as follows:

\[
\text{Enc}(x) = \begin{cases} 
2^x - 1 & \text{if } x > 0, \\
-2^x & \text{if } x \leq 0.
\end{cases}
\]

This solution is inspired by thinking of a bijection between integers and naturals. Indeed, the function \( f : \mathbb{Z} \to \mathbb{N} \) defined by

\[
f(x) = \begin{cases} 
2x - 1 & \text{if } x > 0, \\
-2x & \text{if } x \leq 0,
\end{cases}
\]

is such a bijection. ■
Chapter 2

Deterministic Finite Automata
Exercise 2.1 (Draw DFAs).
For each language below (over the alphabet $\Sigma = \{0, 1\}$), draw a DFA recognizing it.

(a) $\{110, 101\}$
(b) $\{0, 1\}^* \setminus \{110, 101\}$
(c) $\{x \in \{0, 1\}^* : x \text{ starts and ends with the same bit}\}$
(d) $\{\epsilon, 110, 110110, 110110110, \ldots \}$
(e) $\{x \in \{0, 1\}^* : x \text{ contains 110 as a substring}\}$

Solution. (a) Below, all missing transitions go to a rejecting sink state.

(b) Take the DFA above and flip the accepting and rejecting states.

(c)

(d) Below, all missing transitions go to a rejecting sink state.

(e)

Exercise 2.2 (Finite languages are regular).
Let $L$ be a finite language, i.e., it contains a finite number of words. Show that there is a DFA recognizing $L$. 
Exercise 2.3 (Equal number of 01's and 10's).
Is the language
\[ \{ w \in \{0,1\}^* : w \text{ contains an equal number of occurrences of } 01 \text{ and } 10 \text{ as substrings.} \} \]
regular?

Solution. The answer is yes because the language is exactly same as the language in Exercise (Draw DFAs), part (c).

Exercise 2.4 \((a^n b^n c^n \text{ is not regular})\).
Let \( \Sigma = \{a, b, c\} \). Prove that \( L = \{a^n b^n c^n : n \in \mathbb{N}\} \) is not regular.

Solution. Our goal is to show that \( L = \{a^n b^n c^n : n \in \mathbb{N}\} \) is not regular. The proof is by contradiction. So let’s assume that \( L \) is regular.

Since \( L \) is regular, by definition, there is some deterministic finite automaton \( M \) that recognizes \( L \). Let \( k \) denote the number of states of \( M \). For \( n \in \mathbb{N} \), let \( r_n \) denote the state that \( M \) reaches after reading \( a^n \) (i.e., \( r_n = \delta(q_0, a^n) \)). By the pigeonhole principle, we know that there must be a repeat among \( r_0, r_1, \ldots, r_k \).

In other words, there are indices \( i, j \in \{0, 1, \ldots, k\} \) with \( i \neq j \) such that \( r_i = r_j \). This means that the string \( a^i \) and the string \( a^j \) end up in the same state in \( M \).

Therefore \( a^i w \) and \( a^j w \), for any string \( w \in \{0,1\}^* \), end up in the same state in \( M \). We’ll now reach a contradiction, and conclude the proof, by considering a particular \( w \) such that \( a^i w \) and \( a^j w \) end up in different states.

Consider the string \( w = b^j c^i \). Then since \( M \) recognizes \( L \), we know \( a^i w = a^i b^j c^i \) must end up in an accepting state. On the other hand, since \( i \neq j \), \( a^j w = a^j b^i c^j \) is not in the language, and therefore cannot end up in an accepting state. This is the desired contradiction.

Exercise 2.5 \((c^{251} a^n b^{2n} \text{ is not regular})\).
Let \( \Sigma = \{a, b, c\} \). Prove that \( L = \{c^{251} a^n b^{2n} : n \in \mathbb{N}\} \) is not regular.

Solution. Our goal is to show that \( L = \{c^{251} a^n b^{2n} : n \in \mathbb{N}\} \) is not regular. The proof is by contradiction. So let’s assume that \( L \) is regular.

Since \( L \) is regular, by definition, there is some deterministic finite automaton \( M \) that recognizes \( L \). Let \( k \) denote the number of states of \( M \). For \( n \in \mathbb{N} \), let \( r_n \) denote the state that \( M \) reaches after reading \( c^{251} a^n \). By the pigeonhole principle, we know that there must be a repeat among \( r_0, r_1, \ldots, r_k \).

In other words, there are indices \( i, j \in \{0, 1, \ldots, k\} \) with \( i \neq j \) such that \( r_i = r_j \). This means that the string \( c^{251} a^i \) and the string \( c^{251} a^j \) end up in the same state in \( M \).

Therefore \( c^{251} a^i w \) and \( c^{251} a^j w \), for any string \( w \in \{a, b, c\}^* \), end up in the same state in \( M \). We’ll now reach a contradiction, and conclude the proof, by considering a particular \( w \) such that \( c^{251} a^i w \) and \( c^{251} a^j w \) end up in different states.

Consider the string \( w = b^{251} c^i \). Then since \( M \) recognizes \( L \), we know \( c^{251} a^i w = c^{251} a^i b^{2} c^i \) must end up in an accepting state. On the other hand, since \( i \neq j \), \( c^{251} a^j w = c^{251} a^j b^{2} \) is not in the language, and therefore cannot end up in an accepting state. This is the desired contradiction.

Exercise 2.6 (Are regular languages closed under complementation?).
Is it true that if \( L \) is regular, then its complement \( \Sigma^* \setminus L \) is also regular? In other words, are regular languages closed under the complementation operation?
Solution. Yes. If $L$ is regular, then there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $L$. The complement of $L$ is recognized by the DFA $M = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Take a moment to observe that this exercise allows us to say that a language is regular if and only if its complement is regular. Equivalently, a language is not regular if and only if its complement is not regular.

Exercise 2.7 (Are regular languages closed under subsets?).
Is it true that if $L \subseteq \Sigma^*$ is a regular language, then any $L' \subseteq L$ is also a regular language?

Solution. No. For example, $L = \Sigma^*$ is a regular language (construct a single state DFA in which the state is accepting). On the other hand, by Theorem (Are regular languages closed under complementation?), $\{0^n1^n : n \in \mathbb{N}\} \subseteq \Sigma^*$ is not regular.

Exercise 2.8 (Direct proof that regular languages are closed under difference).
Give a direct proof (without using the fact that regular languages are closed under complementation, union and intersection) that if $L_1$ and $L_2$ are regular languages, then $L_1 \setminus L_2$ is also regular.

Solution. The proof is very similar to the proof of Theorem (Regular languages are closed under union). The only difference is the definition of $F''$, which now needs to be defined as

$$F'' = \{(q, q') : q \in F \text{ and } q' \in Q \setminus F'\}.$$ 

The argument that $L(M'') = L(M) \setminus L(M')$ needs to be slightly adjusted in order to agree with $F''$.

Exercise 2.9 (Finite vs infinite union).
(a) Suppose $L_1, \ldots, L_k$ are all regular languages. Is it true that their union $\bigcup_{i=0}^{k} L_i$ must be a regular language?
(b) Suppose $L_0, L_1, L_2, \ldots$ is an infinite sequence of regular languages. Is it true that their union $\bigcup_{i \geq 0} L_i$ must be a regular language?

Solution. In part (a), we are asking whether a finite union of regular languages is regular. The answer is yes, and this can be proved using induction, with the base case corresponding to Theorem (Regular languages are closed under union). In part (b), we are asking whether a countably infinite union of regular languages is regular. The answer is no. First note that any language of cardinality 1 is regular, i.e., $\{w\}$ for any $w \in \Sigma^*$ is a regular language. In particular, for any $n \in \mathbb{N}$, the language $L_n = \{0^n1^n\}$ of cardinality 1 is regular. But

$$\bigcup_{n \geq 0} L_n = \{0^n1^n : n \in \mathbb{N}\}$$

is not regular.

Exercise 2.10 (Union of irregular languages).
Suppose $L_1$ and $L_2$ are not regular languages. Is it always true that $L_1 \cup L_2$ is not a regular language?

Solution. The answer is no. Consider $L = \{0^n1^n : n \in \mathbb{N}\}$, which is a non-regular language. Furthermore, the complement of $L$, which is $\overline{L} = \Sigma^* \setminus L$, is non-regular. This is because regular languages are closed under complementation (Exercise (Are regular languages closed under complementation?)), so if $\overline{L}$ was regular, then $\overline{\overline{L}} = L$ would also have to be regular. The union of $L$ and $\overline{L}$ is $\Sigma^*$, which is a regular language.
Exercise 2.11 (Regularity of suffixes and prefixes).
Suppose $L \subseteq \Sigma^*$ is a regular language. Show that the following languages are also regular:

- $\text{SUFFIXES}(L) = \{x \in \Sigma^* : yx \in L \text{ for some } y \in \Sigma^*\}$,
- $\text{PREFIXES}(L) = \{y \in \Sigma^* : yx \in L \text{ for some } x \in \Sigma^*\}$.

Solution. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing $L$. Define the set

$$S = \{s \in Q : \exists y \in \Sigma^* \text{ such that } \delta(q_0, y) = s\}.$$ 

Now we define a DFA for each $s \in S$ as follows: $M_s = (Q, \Sigma, \delta, s, F)$. Observe that

$$\text{SUFFIXES}(L) = \bigcup_{s \in S} L(M_s).$$

Since $L(M_s)$ is regular for all $s \in S$ and $S$ is a finite set, using Exercise (Finite vs infinite union) part (a), we can conclude that $\text{SUFFIXES}(L)$ is regular.

For the second part, define the set

$$R = \{r \in Q : \exists x \in \Sigma^* \text{ such that } \delta(r, x) \in F\}.$$ 

Now we can define the DFA $M_R = (Q, \Sigma, \delta, q_0, R)$. Observe that this DFA recognizes $\text{PREFIXES}(L)$, which shows that $\text{PREFIXES}(L)$ is regular. ■
Chapter 3

Turing Machines
Exercise 3.1 (Practice with configurations).

(a) Suppose $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ is a Turing machine. We want you to formally define $\alpha \vdash_M \beta$. More precisely, suppose $\alpha = uqv$, where $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$. Precisely describe $\beta$.

(b) Let $M$ denote the Turing machine shown below, which has input alphabet $\Sigma = \{0\}$ and tape alphabet $\Gamma = \{0, x, \sqcup\}$. (Note on notation: A transition label usually has two symbols, one corresponding to the symbol being read, and the other corresponding to the symbol being written. If a transition label has one symbol, the interpretation is that the symbol being read and written is exactly the same.)

We want you to prove that $M$ accepts the input $0000$ using the definition on the previous page. More precisely, we want you to write out the computation trace

$$\alpha_0 \vdash_M \alpha_1 \vdash_M \cdots \vdash_M \alpha_T$$

for $M(0000)$. You do not have to justify it; just make sure to get $T$ and $\alpha_0, \ldots, \alpha_T$ correct!

Solution. Part (a): Let $\alpha = uqv$, where $u = u_1 \cdots u_m$ and $v = v_1 \cdots v_n$ ($u$ and $v$ possibly empty). Let $v'_1$ be $v_1$ if it exists or $\sqcup$ otherwise. Let $\delta(q, v'_1) = (q', x, D)$ (where $D$ is either $L$ or $R$). We write $\alpha \vdash_M \beta$, where $\beta$ is defined as follows:

- if $D = L, m > 0$, then $\beta = u_1 \cdots u_{m-1}q'u_mxv_2 \cdots v_n$;
- if $D = L, m = 0$, then $\beta = q' \sqcup xv_2 \cdots v_n$;
- if $D = R$, then $\beta = u_1 \cdots u_mxq'v_2 \cdots v_n$.

Part (b): Below is the trace for the execution of the Turing Machine. Read down
first and then to the right.

\[
\begin{array}{llll}
q_0000 & \sqcup q_40x\sqcup & \sqcup xq_4xx\sqcup \\
\sqcup q_1000 & q_4 \sqcup x0\sqcup & \sqcup xq_4xx\sqcup \\
\sqcup xq_200 & \sqcup q_10x\sqcup & q_4 \sqcup xxx\sqcup \\
\sqcup x0q_30 & \sqcup xq_10x\sqcup & \sqcup q_1xxx\sqcup \\
\sqcup x0xq_2 & \sqcup xxq_2x\sqcup & \sqcup xxq_1x\sqcup \\
\sqcup x0q_4x & \sqcup xxxq_2x\sqcup & \sqcup xxxq_1x\sqcup \\
\sqcup xq_40x & \sqcup xxq_4x\sqcup & \sqcup xxx \sqcup q_{acc}
\end{array}
\]

Exercise 3.2 (A simple decidable language).
Give a description of the language decided by the TM shown in the example corresponding to Definition (Turing machine).

Solution. The language decided by the TM is

\[L = \{w \in \{a,b\}^* : |w| \geq 2 \text{ and } w_1 = w_2\}.\]

Exercise 3.3 (Drawing TM state diagrams).
For each language below, draw the state diagram of a TM that decides the language. You can use any finite tape alphabet \(\Gamma\) containing the elements of \(\Sigma\) and the symbol \(\sqcup\).

(a) \(L = \{0^n1^n : n \in \mathbb{N}\}\), where \(\Sigma = \{0,1\}\).

(b) \(L = \{0^n : n \text{ is a nonnegative integer power of } 2\}\), where \(\Sigma = \{0\}\).

Solution. Part (a):

\[
\Sigma = \{0,1\} \quad \Gamma = \{0,1,\#,\sqcup\}
\]

Part (b): See the figure in Exercise (Practice with configurations), part (b).

Exercise 3.4 (Decidability is closed under intersection and union).
Let \(L\) and \(K\) be decidable languages. Show that \(L \cap K\) and \(L \cup K\) are also decidable by presenting high-level descriptions of TMs deciding them.

Solution. Since \(L_1\) and \(L_2\) are decidable, there are decider TMs \(M_1\) and \(M_2\) such that \(L(M_1) = L_1\) and \(L(M_2) = L_2\). To show \(L_1 \cup L_2\) is decidable, we present a high-level description of a TM \(M\) deciding it:
It is pretty clear that this decider works correctly. However, in case you are wondering how in general (with more complicated examples) we would prove that a decider works as desired, here is an example argument.

We want to show that $x \in L_1 \cup L_2$ if and only if it is accepted by the above TM $M$. If $x \in L_1 \cup L_2$, then it is either in $L_1$ or in $L_2$. If it is in $L_1$, then $M_1(x)$ accepts (since $M_1$ correctly decides $L_1$) and therefore $M$ accepts on line 1. If, on the other hand, $x \in L_2$, then $M_2(x)$ accepts. This means that if $M$ does not accept on line 1, then it has to accept on line 2. Either way $x$ is accepted by $M$. For the converse, assume $x$ is accepted by $M$. Then it must be accepted on line 1 or line 2. If it is accepted on line 1, then this implies that $M_1(x)$ accepted, i.e., $x \in L_1$. If it is accepted on line 2, then $M_2(x)$ accepted, i.e., $x \in L_2$. So $x \in L_1 \cup L_2$, as desired.

To show $L_1 \cap L_2$ is decidable, we present a high-level description of a TM $M$ deciding it:

\[
\begin{align*}
x: \text{string} \\
M(x): \\
1 & \text{ Run } M_1(x), \text{ if it accepts, accept.} \\
2 & \text{ Run } M_2(x), \text{ if it accepts, accept.} \\
3 & \text{ Reject.}
\end{align*}
\]

Once again, it is clear that this decider works correctly.

\[\blacksquare\]

**Exercise 3.5 (Decidable language based on pi).**

Let $L \subseteq \{3\}^*$ be defined as follows: $x \in L$ if and only if $x$ appears somewhere in the decimal expansion of $\pi$. For example, the strings $\epsilon$, 3, and 33 are all definitely in $L$, because

$\pi = 3.1415926535897932384626433 \ldots$

Prove that $L$ is decidable. No knowledge in number theory is required to solve this question.

**Solution.** The important observation is the following. If, for some $m \in \mathbb{N}$, $3^m$ is not in $L$, then neither is $3^k$ for any $k > m$. Additionally, if $3^m \in L$, then so is $3^\ell$ for every $\ell < m$. For each $n \in \mathbb{N}$, define

$L_n = \{3^m : m \leq n\}.$

Then either $L = L_n$ for some $n$, or $L = \{3\}^*$.

If $L = L_n$ for some $n$, then the following TM decides it.

\[
\begin{align*}
x: \text{string} \\
M(x): \\
1 & \text{ If } |x| \leq n, \text{ accept.} \\
2 & \text{ Else, reject.}
\end{align*}
\]

If $L = \{3\}^*$, then it is decided by:
Exercise 3.6 (Practice with decidability through reductions).

(a) Let $L = \{(D_1, D_2) : D_1$ and $D_2$ are DFAs with $L(D_1) \subseteq L(D_2)\}$. Show that $L$ is decidable.

(b) Let $K = \{\langle D \rangle : D$ is a DFA that accepts $w^R$ whenever it accepts $w\}$, where $w^R$ denotes the reversal of $w$. Show that $K$ is decidable. For this question, you can use the fact given a DFA $D$, there is an algorithm to construct a DFA $D'$ such that $L(D') = L(D)^R = \{w^R : w \in L(D)\}$.

Solution. Part (a): To show $L$ is decidable, we are going to use the fact that $\text{EMPTY}_{\text{DFA}}$ is decidable (Theorem ($\text{EMPTY}_{\text{DFA}}$ is decidable)) and $\text{EQ}_{\text{DFA}}$ is decidable (Theorem (EQ_{DFA} is decidable)). Let $M_{\text{EMPTY}}$ denote a decider TM for $\text{EMPTY}_{\text{DFA}}$ and let $M_{\text{EQ}}$ denote a decider TM for $\text{EQ}_{\text{DFA}}$.

A decider for $L$ takes as input $\langle D_1, D_2 \rangle$, where $D_1$ and $D_2$ are DFAs. It needs to determine if $L(D_1) \subseteq L(D_2)$ (i.e. accept if $L(D_1) \subseteq L(D_2)$ and reject otherwise). To determine this we do two checks:

(i) Check whether $L(D_1) = L(D_2)$.

(ii) Check whether $L(D_1) \subseteq L(D_2)$. Observe that this can be done by checking whether $L(D_1) \cap \overline{L(D_2)} = \emptyset$.

Note that $L(D_1) \subseteq L(D_2)$ if and only if $L(D_1) \neq L(D_2)$ and $L(D_1) \cap \overline{L(D_2)} = \emptyset$. Using the closure properties of regular languages, we can construct a DFA $D$ such that $L(D) = L(D_1) \cap \overline{L(D_2)}$. Now the decider for $L$ can be described as follows:

\[ D_1: \text{DFA}, \quad D_2: \text{DFA}. \]
\[ M(\langle D_1, D_2 \rangle); \]
\[ 1 \quad \text{Construct DFA } D \text{ as described above.} \]
\[ 2 \quad \text{Run } M_{\text{EQ}}(\langle D_1, D_2 \rangle). \]
\[ 3 \quad \text{If it accepts, reject.} \]
\[ 4 \quad \text{Else:} \]
\[ 5 \quad \text{Run } M_{\text{EMPTY}}(\langle D \rangle) \]
\[ 6 \quad \text{If it accepts, accept.} \]
\[ 7 \quad \text{Else, reject.} \]

Observe that this machine accepts $\langle D_1, D_2 \rangle$ if and only if $M_{\text{EQ}}(\langle D_1, D_2 \rangle)$ rejects and $M_{\text{EMPTY}}(\langle D \rangle)$ accepts. In other words, it accepts $\langle D_1, D_2 \rangle$ if and only if $L(D_1) \neq L(D_2)$ and $L(D_1) \cap \overline{L(D_2)} = \emptyset$, which is the desired behavior for the machine.

Part (b): We sketch the proof. To show $L$ is decidable, we are going to use the fact that $\text{EQ}_{\text{DFA}}$ is decidable (Theorem (EQ_{DFA} is decidable)). Let $M_{\text{EQ}}$ denote a decider TM for $\text{EQ}_{\text{DFA}}$. Observe that $\langle D \rangle$ is in $K$ if and only if $L(D) = L(D)^R$ (prove this part). Using the fact given to us in the problem description, we know that there is a way to construct $\langle D' \rangle$ such that $L(D') = L(D)^R$. Then all we need to do is run $M_{\text{EQ}}(\langle D, D' \rangle)$ to determine whether $\langle D \rangle \in K$ or not. \(\blacksquare\)

---

1Note on notation: for sets $A$ and $B$, we write $A \subseteq B$ if $A \subseteq B$ and $A \neq B$.  

\[ x: \text{string} \]
\[ M(x): \]
\[ 1 \quad \text{Accept.} \]
Chapter 4

Countable and Uncountable Sets
Exercise 4.1 (Exercise with injections and surjections).
Prove parts (a) and (b) of the above theorem.

Solution. Unfortunately we currently do not have the solution to this exercise.

Exercise 4.2 (Proof of the characterization of countably infinite sets).
Prove the above theorem.

Solution. We need to show that $A$ is countably infinite if and only if $|A| = |\mathbb{N}|$, so we will argue the two direction separately.

If $|A| = |\mathbb{N}|$, then $|A| \leq |\mathbb{N}|$, and $A$ is infinite because a finite set cannot be in one-to-one correspondence with an infinite set. Therefore $A$ is countably infinite.

For the other direction, assume $A$ is such that $|A| \leq |\mathbb{N}|$ and $A$ is infinite. Since $|A| \leq |\mathbb{N}|$, there is an injection $f : A \to \mathbb{N}$. This $f$ allows us to define an ordering on $A$. For $a, b \in A$, write $a < b$ if $f(a) < f(b)$. Using this ordering, we can define a bijection $g : \mathbb{N} \to A$, where $g(n)$ is defined to be the $n$'th smallest element in $A$ (and we start counting from 0). Since $A$ is infinite, $g(n)$ is well-defined for all $n$. Clearly $g$ is injective since we cannot have $g(n) = g(n')$ for $n \neq n'$. Furthermore $g$ is surjective because for every $a \in A$, the pre-image is $g^{-1}(a) = \{x \in A : f(x) < f(a)\}$. ■

Exercise 4.3 (Practice with countability proofs).
Show that the following sets are countable.

(a) $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$.

(b) The set of all functions $f : A \to \mathbb{N}$, where $A$ is a finite set.

Solution. Part (a): We want to show that $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ is countable. We use the CS method with $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, #\}$. Note that any element of $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ can be written uniquely as a finite word over $\Sigma$ (we use the hashtag as a separator between the integers). As an illustration, $(9234851, -1234, 0) \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ can be encoded as the string 9234851#-1234#0. Every integer has finite length, so the string encoding is always of finite length.

Part (b): Let $S$ be the set of all functions $f : A \to \mathbb{N}$, where $A$ is a finite set. We want to show that $S$ is countable.

We first make an observation about the elements of $S$. Take a function $f : A \to \mathbb{N}$, where $A$ is a finite set. Let $k$ be the size of $A$ and let $a_1, a_2, \ldots, a_k$ be its elements. Then $f$ can be uniquely represented by the tuple

$$ (f(a_1), f(a_2), \ldots, f(a_k)), $$

where each element of the tuple is an element from $\mathbb{N}$.

We now show that $S$ is countable using the CS method with the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, #\}$. The observation above shows that any element of $S$ can be uniquely represented with a finite length string where commas are replaced with #. (Note that there is no need to put the opening and closing parentheses.) This suffices to conclude that $S$ is countable. ■

Exercise 4.4 (Uncountable sets are closed under supersets).
Prove that if $A$ is uncountable and $A \subseteq B$, then $B$ is also uncountable.
Solution. We want to show that if $B$ is a superset of an uncountable set $A$, then $B$ must be uncountable.

If $A$ is uncountable, by definition, $|A| > |\mathbb{N}|$. If $A \subseteq B$, then there is a clear injection from $A$ to $B$ (map $a \in A$ to $a \in B$), so $|A| \leq |B|$. Combining this with $|A| > |\mathbb{N}|$, we have $|B| \geq |A| > |\mathbb{N}|$, and therefore $B$ is uncountable. ■

Exercise 4.5 (Practice with uncountability proofs).
Show that the following sets are uncountable.

(a) The set of all bijective functions from $\mathbb{N}$ to $\mathbb{N}$.

(b) $\{x_1 x_2 x_3 \ldots \in \{1, 2\}^\infty : \text{ for all } n \geq 1, \ \sum_{i=1}^{n} x_i \neq 0 \mod 4\}$

Solution. Part (a): Let $A$ be the set of all bijective functions from $\mathbb{N}$ to $\mathbb{N}$. We want to show that $A$ is uncountable, and we will do so by showing that $\{0, 1\}^\infty \hookrightarrow A$, establishing $|\{0, 1\}^\infty| \leq |A|$.

We now describe this injective mapping. Given $x \in \{0, 1\}^\infty$, we map it to a bijection $f_x : \mathbb{N} \rightarrow \mathbb{N}$ as follows. Let $x_n$ be the $n$'th bit of $x$, and assume the indexing starts from 0. Then for all $n \in \mathbb{N}$,

- if $x_n = 0$, $f_x$ maps $2n$ to $2n$ and $2n + 1$ to $2n + 1$;
- if $x_n = 1$, $f_x$ maps $2n$ to $2n + 1$ and $2n + 1$ to $2n$.

The below picture illustrates the construction of $f_x$. If $x_n = 0$, we pick the black arrows to map $2n$ and $2n + 1$, and if $x_n = 1$ we pick the red/dashed arrows to map $2n$ and $2n + 1$.

![Arrow Diagram]

Observe that for any $x \in \{0, 1\}^\infty$, the corresponding function $f_x$ is indeed a bijection. It is also clear that if $x \neq x'$, then $f_x \neq f_x'$. So this mapping from $\{0, 1\}^\infty$ to $A$ is indeed an injection. This completes the proof.

Part (b): Let $A = \{x_1 x_2 x_3 \ldots \in \{1, 2\}^\infty : \text{ for all } n \geq 1, \ \sum_{i=1}^{n} x_i \neq 0 \mod 4\}$. We want to show that $A$ is uncountable, and we will do so by identifying a subset of $A$ that is in one-to-one correspondence with $\{0, 1\}^\infty$.

Let $a = 22$ and $b = 112$. Define the set $A' = \{1w : w \in \{a, b\}^\infty\}$. Observe that $A' \subseteq A$ (this needs a short argument that we skip). Furthermore, it is clear that there is a bijection between $A'$ and $\{0, 1\}^\infty$. So we have identified a subset of $A$ that is in one-to-one correspondence with $\{0, 1\}^\infty$, which allows us to conclude that $A$ is uncountable. ■
Chapter 5

Undecidable Languages
Exercise 5.1 (Practice with undecidability proofs).
Show that the following languages are undecidable.

(a) $\text{EMPTY-HALTS} = \{ \langle M \rangle : M$ is a TM and $M(\epsilon)$ halts}.

(b) $\text{FINITE} = \{ \langle M \rangle : M$ is a TM that accepts finitely many strings}.

Solution. Part (a): We want to show $\text{EMPTY-HALTS}$ is undecidable. To proof is by contradiction, so assume that $\text{EMPTY-HALTS}$ is decidable, and let $M_{\text{EMPTY-HALTS}}$ be a decider for it. We will use $M_{\text{EMPTY-HALTS}}$ to show that $\text{HALTS}$ is decidable and reach a contradiction. The description of $M_{\text{HALTS}}$, the decider for $\text{HALTS}$, is as follows.

$M$: TM. $x$: string.
$M_{\text{HALTS}}(\langle M, x \rangle)$:
1. Construct the following string, which we call $\langle M' \rangle$.
2. 
   "$M'(y) :$
3. Run $M(x)$.
4. Accept."
5. Run $M_{\text{EMPTY-HALTS}}(\langle M' \rangle)$.
6. If it accepts, accept.
7. If it rejects, reject.

To see that this is a correct decider for $\text{HALTS}$, first consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \in \text{HALTS}$, i.e., $M(x)$ halts. By the construction of $M'$, this implies that $M'(y)$ halts (and accepts) for any string $y$. So $M_{\text{EMPTY-HALTS}}(\langle M' \rangle)$ accepts, and our decider above accepts as well. So in this case, the decider gives the correct answer.

Now consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \notin \text{HALTS}$, i.e., $M(x)$ loops. Then for any input $y$, $M'(y)$ would get stuck on line 3, and would never halt. This means $M_{\text{EMPTY-HALTS}}(\langle M' \rangle)$ rejects, and our decider rejects as well, as desired.

For any input, our decider gives the correct answer, and the proof is complete.

Part (b): Our goal is to show that $\text{FINITE}$ is undecidable. To proof is by contradiction, so assume that $\text{FINITE}$ is decidable, and let $M_{\text{FINITE}}$ be a decider for it. We will use $M_{\text{FINITE}}$ to show that $\text{HALTS}$ is decidable and reach a contradiction. The description of $M_{\text{HALTS}}$, the decider for $\text{HALTS}$, is as follows.

$M$: TM. $x$: string.
$M_{\text{HALTS}}(\langle M, x \rangle)$:
1. Construct the following string, which we call $\langle M' \rangle$.
2. "$M'(y) :$
3. Run $M(x)$.
4. Accept."
5. Run $M_{\text{FINITE}}(\langle M' \rangle)$.
6. If it accepts, reject.
7. If it rejects, accept.

To see that this is a correct decider for $\text{HALTS}$, first consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \in \text{HALTS}$, i.e., $M(x)$ halts. By the construction of $M'$, this implies that $M'(y)$ accepts for any string $y$. So $L(M') = \Sigma^*$ (an infinite set), and therefore $M_{\text{FINITE}}(\langle M' \rangle)$ rejects. In this case, our decider for $\text{HALTS}$ accepts and gives the correct answer.

Now consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \notin \text{HALTS}$, i.e., $M(x)$ loops. Then for any input $y$, $M'(y)$ would get stuck on line 3, and would never halt.
So \( L(M') = \emptyset \) (a finite set), and therefore \( M_{\text{FINITE}}(\langle M' \rangle) \) accepts. In this case, our decider for HALTS rejects and gives the correct answer.

For any input, our decider gives the correct answer, and the proof is complete.

\[ \square \]

Exercise 5.2 (Practice with reduction definition).
Let \( A, B \subseteq \{0, 1\}^* \) be languages. Prove or disprove the following claims.

(a) If \( A \leq B \) then \( B \leq A \).

(b) If \( A \leq B \) and \( B \) is regular, then \( A \) is regular.

Solution. Part (a): The claim is false. Let \( A \) be any decidable language. For example, we can take \( A = \emptyset \). The decider for \( A \) is a machine that rejects no matter what the input is. Let \( B = \text{HALTS} \). Then to establish \( A \leq B \), we need to argue that given a decider for \( \text{HALTS} \), we can decide \( \emptyset \). Since \( \emptyset \) is decidable, this is true (and we don’t even need to make use of a decider for \( \text{HALTS} \)). On the other hand, it is not true that \( \text{HALTS} \leq \emptyset \). For the sake of contradiction, if it was true, then this would mean that using a decider for \( \emptyset \), we can decide \( \text{HALTS} \). And this would imply that \( \text{HALTS} \) is decidable, a contradiction.

Part (b): The claim is false. Consider \( A = \{0^n1^n : n \in \mathbb{N}\} \) and \( B = \emptyset \). We have \( A \leq B \) because \( A \) is a decidable language (we don’t even need to make use of the decider for \( B \)). Furthermore, \( B \) is regular, but \( A \) is not.

\[ \square \]

Exercise 5.3 (Practice with reduction proofs).
Show the following.

(a) \( \text{ACCEPTS} \leq \text{HALTS} \).

(b) \( \text{HALTS} \leq \text{EQ} \).

Solution. Part (a): We want to show that \( \text{ACCEPTS} \) reduces to \( \text{HALTS} \). To do this, we assume that \( \text{HALTS} \) is decidable. Let \( M_{\text{HALTS}} \) be a decider for \( \text{HALTS} \). We now need to construct a TM that decides \( \text{ACCEPTS} \) (which will make use of \( M_{\text{HALTS}} \)). Here is the description of the decider:

\[
M: \text{TM. } x: \text{string,}
M_{\text{ACCEPTS}}(\langle M, x \rangle):
\begin{align*}
1 & \text{ Run } M_{\text{HALTS}}(\langle M, x \rangle). \\
2 & \text{ If it rejects, reject. } \\
3 & \text{ Else: } \\
4 & \text{ Run } M(x) \\
5 & \text{ If it accepts, accept. } \\
6 & \text{ If it rejects, reject. }
\end{align*}
\]

We now argue that this machine indeed decides \( \text{ACCEPTS} \). Note that given \( \langle M, x \rangle \), there are three possibilities: \( M(x) \) accepts; \( M(x) \) rejects, \( M(x) \) loops. And \( \langle M, x \rangle \in \text{ACCEPTS} \) if and only if \( M(x) \) accepts.

Let’s first consider an input \( \langle M, x \rangle \) such that \( \langle M, x \rangle \in \text{ACCEPTS} \). Then \( M_{\text{HALTS}}(\langle M, x \rangle) \) must accept, i.e. \( M(x) \) halts. So the decider above safely simulates \( M(x) \) on line 4 and accepts (on line 5).

If the the input is such that \( \langle M, x \rangle \notin \text{ACCEPTS} \), then there are two cases. Either \( M(x) \) halts but rejects, or \( M(x) \) loops. If it is the latter, then \( M_{\text{HALTS}}(\langle M, x \rangle) \) rejects, and therefore our decider rejects as well. If on the other hand \( M(x) \) halts but rejects, then our decider safely simulates \( M(x) \) on line 4 and rejects (on line 6).
Whatever the input is, our decider gives the correct answer. The proof is complete.

Part(b): (This can be considered as an alternative proof of Theorem (EQ is undecidable).) We want to show that HALTS reduces to EQ. To do this, we assume that EQ is decidable. Let $M_{EQ}$ be a decider for EQ. We now need to construct a TM that decides HALTS (which will make use of $M_{EQ}$). Here is the description of the decider:

$M$: TM. $x$: string.
$M_{HALTS}(\langle M, x \rangle)$:
1. Construct the string $\langle M' \rangle$ where $M'$ is a TM that rejects every input.
2. Construct the following string, which we call $\langle M'' \rangle$.
3. "$M''(y) :$
   4. Run $M(x)$.
   5. Ignore the output and accept.”
6. Run $M_{EQ}(\langle M', M'' \rangle)$.
7. If it accepts, reject.
8. If it rejects, accept.

We now argue that this machine indeed decides HALTS. Notice that no matter what the input is, $L(M') = \emptyset$. Let’s first consider an input $\langle M, x \rangle$ such that $\langle M, x \rangle \in$ HALTS. Then $M''$ accepts every input, so $L(M') = \Sigma^*$. In this case, $M_{EQ}(\langle M', M'' \rangle)$ rejects, and so our machine accepts as desired. Next, consider an input $\langle M, x \rangle$ such that $\langle M, x \rangle \notin$ HALTS. Then whatever input is given to $M''$, it gets stuck in an infinite loop when it runs $M(x)$. So $L(M'') = \emptyset$. In this case $M_{EQ}(\langle M', M'' \rangle)$ accepts, and so our machine rejects, as it should. Thus we have a correct decider for HALTS. ■
Chapter 6

Time Complexity
Exercise 6.1 (Practice with big-O).
Show that $3n^2 + 10n + 30$ is $O(n^2)$.

Solution. Proof 1: To show that $3n^2 + 10n + 30$ is $O(n^2)$, we need to show that there exists $C > 0$ and $n_0 > 0$ such that

$$3n^2 + 10n + 30 \leq Cn^2$$

for all $n \geq n_0$. Pick $C = 4$ and $n_0 = 13$. Note that for $n \geq 13$, we have

$$10n + 30 \leq 10n + 3n = 13n \leq n^2.$$

This implies that for $n \geq 13$ and $C = 4$,

$$3n^2 + 10n + 30 \leq 3n^2 + n^2 = Cn^2.$$

Proof 2: Pick $C = 43$ and $n_0 = 1$. Then for $n \geq 1 = n_0$, we have

$$3n^2 + 10n + 30 \leq 3n^2 + 10n^2 + 30n^2 = 43n^2 = Cn^2.$$

Exercise 6.2 (Practice with big-Omega).
Show that $n!^2$ is $\Omega(n^n)$.

Solution. To show $(n!)^2 = \Omega(n^n)$, we'll show that choosing $c = 1$ and $n_0 = 0$ satisfies the definition of Big-Omega. To see this, note that for $n \geq 1$, we have:

$$(n!)^2 = ((n)(n-1)\cdots(1))((n)(n-1)\cdots(1))$$

(by definition)

$$= (n)(n)(n-1)(n-1)(n-2)\cdots(1)(1)$$

(re-ordering terms)

$$\geq (n)(n)(n)\cdots(n)$$

(pairing up consecutive terms)

(by the Claim below)

$$= n^n.$$

Claim: For $n \geq 1$ and for $i \in \{0, 1, \ldots, n-1\}$,

$$(n-i)(i+1) \geq n.$$

Proof: The proof follows from the following chain of implications.

$$n - (n-1) - 1 = 0 \implies n - i - 1 \geq 0 \quad \text{(since $i \leq n - 1$)}$$

$$\implies i(n - i - 1) \geq 0 \quad \text{(since $i \geq 0$)}$$

$$\implies ni - i^2 - i \geq 0$$

$$\implies ni - i^2 - i + n \geq n$$

$$\implies (n-i)(i+1) \geq n.$$

This completes the proof.

Exercise 6.3 (Practice with Theta).
Show that $\log_2(n!) = \Theta(n \log n)$.

Solution. We first show $\log_2(n!) = O(n \log n)$. Observe that

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1 \leq n \cdot n \cdot n \cdots n = n^n,$$

as each term on the RHS (i.e. $n$) is greater than or equal to each term on the LHS. Taking the log of both sides gives us $\log_2 n! \leq \log_2 n^n = n \log_2 n$ (here, we
are using the fact that $\log a^b = b \log a)$. Therefore taking $n_0 = C = 1$ satisfies the definition of big-$O$, and $\log_2 n! = O(n \log n)$.

Now we show $\log_2 n! = \Omega(n \log n)$. Assume without loss of generality that $n$ is even. In the definition of $n!$, we’ll use the first $n/2$ terms in the product to lower bound it:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdots 2 \cdot 1 \geq \left(\frac{n}{2}\right)^{n/2} \cdot \frac{n}{2} \cdots \frac{n}{2} = \left(\frac{n}{2}\right)^{2 \cdot \frac{n}{2}}.$$  

Taking the log of both sides gives us $\log_2 n! \geq \frac{n}{2} \log_2 \frac{n}{2}$.

Claim: For $n \geq 4$, $\frac{n}{2} \log_2 \frac{n}{2} \geq \frac{n}{4} \log_2 n$.

The proof of the claim is not difficult (some algebraic manipulation) and is left for the reader. Using the claim, we know that for $n \geq 4$, $\log_2 n! \geq \frac{n}{4} \log_2 n$. Therefore, taking $n_0 = 4$ and $c = 1/4$ satisfies the definition of big-Omega, and $\log_2 n! = \Omega(n \log n)$. ■

Exercise 6.4 (Composing polynomial time algorithms).
Suppose that we have an algorithm $A$ that runs another algorithm $A'$ once as a subroutine. We know that the running time of $A'$ is $O(n^k)$, $k \geq 1$, and the work done by $A$ is $O(n^t)$, $t \geq 1$, if we ignore the subroutine $A'$ (i.e., we don’t count the steps taken by $A'$). What kind of upper bound can we give for the total running-time of $A$ (which includes the work done by $A'$)?

Solution. Let $n$ be the length of the input to algorithm $A$. The total work done by $A$ is $f(n) + g(n)$, where $g(n)$ is the work done by the subroutine $A'$ and $f(n) = O(n^t)$ is the work done ignoring the subroutine $A'$.

We analyze $g(n)$ as follows. Note that in time $O(n^t)$, $A$ can produce a string of length $cn^t$ (for some constant $c$), and feed this string to $A'$. The running time of $A'$ is $O(m^k)$, where $m$ is the length of the input for $A'$. When $A$ is run and calls $A'$, the length of the input to $A'$ can be $m = cn^t$. Therefore, the work done by $A'$ inside $A$ is $O(m^k) = O((cn^t)^k) = O(c^k n^{tk}) = O(n^{tk})$.

Since $f(n) = O(n^t)$ and $g(n) = O(n^{tk})$, we have $f(n) + g(n) = O(n^{tk})$. ■

Exercise 6.5 (TM complexity of $\{0^k1^k : k \in \mathbb{N}\}$).
In the TM model, a step corresponds to one application of the transition function. Show that $L = \{0^k1^k : k \in \mathbb{N}\}$ can be decided by a TM in time $O(n \log n)$. Is this statement directly implied by Proposition (Intrinsic complexity of $\{0^k1^k : k \in \mathbb{N}\}$)?

Solution. First of all, the statement is not directly implied by Proposition (Intrinsic complexity of $\{0^k1^k : k \in \mathbb{N}\}$) because that proposition is about the RAM model whereas this question is about the TM model.

We now sketch the solution. Below is a medium-level description of a TM deciding the language $\{0^k1^k : k \in \mathbb{N}\}$.

Repeat while both 0s and 1s remain on the tape:
- Scan the tape.
- If (# of 1s + # of 0s) is odd, reject.
- Scan the tape.
- Cross off every other 0 starting with first 0.
- Cross off every other 1 starting with first 1.

If no 0s and no 1s remain accept.
Else, reject.
Let \( n \) be the input length. Observe that in each iteration of the loop, we do \( O(n) \) work, and the number of iterations is \( O(\log n) \) because in each iteration, half of the non-crossed portion of the input gets crossed off (i.e. in each iteration, the number of 0’s and 1’s is halved). So the total running time is \( O(n \log n) \).

**Exercise 6.6** (Is polynomial time decidability closed under concatenation?). Assume the languages \( L_1 \) and \( L_2 \) are decidable in polynomial time. Prove or give a counter-example: \( L_1L_2 \) is decidable in polynomial time.

**Solution.** Let \( M_1 \) be a decider for \( L_1 \) with running-time \( O(n^k) \) and let \( M_2 \) be a decider for \( L_2 \) with running-time \( O(n^t) \). We construct a polynomial-time decider for \( L_1L_2 \) as follows:

On input \( x \):

1. For each of the \( |x| + 1 \) ways to divide \( x \) as \( yz \):
   - Run \( M_1(y) \)
   - If \( M_1 \) accepts:
     - Run \( M_2(z) \)
     - If \( M_2 \) accepts, accept
   - Reject

The input length is \( n = |x| \). The for-loop repeats \( n + 1 \) times. In each iteration of the loop, we do at most \( cn^k + c'n^t + c'' \) work, where \( c, c', c'' \) are constants independent of \( n \). So the total running-time is \( O(n^{\max\{k,t\} + 1}) \).

**Exercise 6.7** (Running time of the factoring problem). Consider the following problem: Given as input a positive integer \( N \), output a non-trivial factor\(^3\) of \( N \) if one exists, and output False otherwise. Give a lower bound using the \( \Omega(n) \) notation for the running-time of the following algorithm solving the problem:

\[
N: \text{natural number.}

\text{Non-Trivial-Factor}(\langle N \rangle):
1. \text{For } i = 2 \text{ to } N - 1:\n2. \quad \text{if } N \% i == 0: \text{Return } i.
3. \text{Return False.}
\]

**Solution.** The input is a number \( N \), so the length of the input is \( n \), which is about \( \log_2 N \). In other words, \( N \) is about \( 2^n \). In the worst-case, \( N \) is a prime number, which would force the algorithm to repeat \( N - 2 \) times. Therefore the running-time of the algorithm is \( \Omega(N) \). Writing \( N \) in terms of \( n \), the input length, we get that the running-time is \( \Omega(2^n) \).

**Exercise 6.8** (251st root). Consider the following computational problem. Given as input a number \( A \in \mathbb{N} \), output \( \lfloor A^{1/251} \rfloor \). Determine whether this problem can be computed in worst-case polynomial-time, i.e. \( O(n^k) \) time for some constant \( k \), where \( n \) denotes the number of bits in the binary representation of the input \( A \). If you think the problem can be solved in polynomial time, give an algorithm in pseudocode, explain briefly why it gives the correct answer, and argue carefully why the running time is polynomial. If you think the problem cannot be solved in polynomial time, then provide a proof.

**Solution.** First note that the following algorithm, although correct, is exponential time.

---

\(^3\)A non-trivial factor is a factor that is not equal to 1 or the number itself.
\( A \): natural number.

Linear-Search(\( (A) \)):
1. For \( B = 0 \) to \( A \):
2. \( \text{If } B^{251} > A: \text{Return } B - 1 \).

The length of the input is \( n \), which is about \( \log_2 A \). In other words, \( A \) is about \( 2^n \). The for loop above will repeat \( \lfloor A^{1/251} \rfloor \) many times, so the running-time is \( \Omega(A^{1/251}) \), and in terms of \( n \), this is \( \Omega(2^{n/251}) \).

To turn the above idea into a polynomial-time algorithm, we need to use binary search instead of linear search.

\( A \): natural number.

Binary-Search(\( (A) \)):
1. \( lo = 0 \).
2. \( hi = A \).
3. While \( (lo < hi) \):
4. \( B = \lfloor \frac{lo + hi}{2} \rfloor \).
5. \( \text{If } B^{251} > A: \text{hi} = B - 1 \).
6. \( \text{Else lo} = B \).
7. Return \( lo \).

Since we are using binary search, we know that the loop repeats \( O(\log A) \) times, or using \( n \) as our parameter, \( O(n) \) times. All the variables hold values that are at most \( A \), so they are at most \( n \)-bits long. This means all the arithmetic operations (plus, minus, and division by 2) in the loop can be done in linear time. Computing \( B^{251} \) is polynomial-time because we can compute it by doing integer multiplication a constant number of times, and the numbers involved in these multiplications are \( O(n) \)-bits long. Thus, the total work done is polynomial in \( n \). \( \blacksquare \)
Chapter 7

The Science of Cutting Cake
Exercise 7.1 (Practice with cutting cake).
Design a cake cutting algorithm for a set of players $N = \{1, \ldots, n\}$ that finds an allocation $A$ with the property that there exists a permutation/bijection $\pi_A : N \rightarrow N$ such that for all $i \in N$, $V_i(A_i) \geq \frac{1}{2^{1+n}}$. In words, there is an order on the players such that the first player has value at least $1/2$ for her piece, the second player has value at least $1/4$, and so on. The complexity of your algorithm in the Robertson-Webb model should be $O(n^2)$.

Solution. We can modify the Dubins-Spanier algorithm to solve this exercise question. The referee first makes $n$ queries: Cut$_i(0, 1/2)$ for all $i$. She computes the minimum among these values, which we’ll denote by $y$. Let’s assume $j$ is the player that corresponds to the minimum value. Then the referee assigns $A_j = [0, y]$. So player $j$ gets a piece that she values at $1/2$. After this, we remove player $j$, and repeat the process on the remaining cake. So in the next stage, the referee makes $n-1$ queries, Cut$_i(y, 1/4)$ for $i \neq j$, figures out the player corresponding to the minimum value, and assigns her the corresponding piece of the cake, which she values at $1/4$. This repeats until there is one player left. The last player gets the piece that is left.

The analysis of the running time of this algorithm is exactly the same as in the original Dubins-Spanier algorithm.

We need to show that the allocation produced by the algorithm is such that the first player (in the order the algorithms picks players) has value at least $1/2$ for her piece, the second player has value at least $1/4$ for her piece, and so on. This pretty much follows from the way the algorithm is set up and how allocations are made. The only things we need to check are:

(i) the queries never return “None”,

(ii) the last player, call it $\ell$, gets $A_\ell$ such that $V_\ell(A_\ell) \geq 1/2^n$.

To show (i), assume we have just completed iteration $k$, where $k \in \{1, 2, \ldots, n-1\}$. Let $j$ be an arbitrary player who has not been removed yet. The important observation is that the piece of cake remaining at this point has value at least $1/2^k$ to player $j$ (take a moment to verify this). Since this is true for any $k \in \{1, 2, \ldots, n-1\}$ and any player $j$ that remains after iteration $k$, the queries never return “None”. Part (ii) actually follows from the same argument. The cake remaining after iteration $n-1$ has value at least $1/2^{n-1}$ for the last player (which is indeed better than $1/2^n$). This completes the proof.

Exercise 7.2 (Finding an envy-free allocation).
We say that a valuation function $V$ is piecewise constant if there are points $x_1, x_2, \ldots, x_k \in [0,1]$ such that $0 = x_1 < x_2 < \cdots < x_k = 1$ and for each $i \in \{1, 2, \ldots, k-1\}$, $V([x_i, x_{i+1}])$ is uniformly distributed over $[x_i, x_{i+1}]$. Suppose we have $n$ players such that each player has a piecewise constant valuation function. Show that in this case, an envy-free allocation always exists.

Solution. We sketch the idea, but do not prove the correctness. Each player’s valuation function is associated with a set of points. Make a mark for each such point on the cake $[0,1]$. The subinterval between any two adjacent marks $x$ and $y$ is such that for all players $i$, $V_i([x,y])$ is uniformly distributed. Divide each such subinterval into $n$ pieces of equal length, and give one to each player.

---

1Uniformly distributed means that if we were to take any subinterval $I$ of $[x_i, x_{i+1}]$ whose density/size is $\alpha$ fraction of the density/size of $[x_i, x_{i+1}]$, then $V(I) = \alpha \cdot V([x_i, x_{i+1}])$. 

32
Chapter 8

Introduction to Graph Theory
Exercise 8.1 (Max number of edges in a graph).
In an \( n \)-vertex graph, what is the maximum possible value for the number of edges in terms of \( n \)?

Solution. An edge is a subset of \( V \) of size 2, and there are at most \( \binom{n}{2} \) possible subsets of size 2.

Exercise 8.2 (Application of Handshake Theorem).
Is it possible to have a party with 251 people in which everyone knows exactly 5 other people in the party?

Solution. Create a vertex for each person in the party, and put an edge between two people if they know each other. Note that the question is asking whether there can be a 5-regular graph with 251 nodes. We use Theorem (Handshake Theorem) to answer this question. If such a graph exists, then the sum of the degrees would be \( 5 \times 251 \), which is an odd number. However, this number must equal \( 2m \) (where \( m \) is the number of edges), and \( 2m \) is an even number. So we conclude that there cannot be a party with 251 people in which everyone knows exactly 5 other people.

Exercise 8.3 (Equivalent definitions of a tree).
Show that if a graph has two of the properties listed in Definition (Tree, leaf, internal node), then it automatically has the third as well.

Solution. If a graph is connected and satisfies \( m = n - 1 \), then it must be acyclic by Theorem (Min number of edges to connect a graph). If a graph is connected and acyclic, then it must satisfy \( m = n - 1 \), also by Theorem (Min number of edges to connect a graph). So all we really need to prove is if a graph is acyclic and satisfies \( m = n - 1 \), then it is connected. For this we look into the proof of Theorem (Min number of edges to connect a graph). If the graph is acyclic, this means that every time we put back an edge, we put one that satisfies (i) (here “(i)” is referring to the item in the proof of Theorem (Min number of edges to connect a graph)). This is because any edge that satisfies (ii) creates a cycle. Every time we put an edge satisfying (i), we reduce the number of connected components by 1. Since \( m = n - 1 \), we put back \( n - 1 \) edges. This means we start with \( n \) connected components (\( n \) isolated vertices), and end up with 1 connected component once all the edges are added back. So the graph is connected.

Exercise 8.4 (A tree has at least 2 leaves).
Let \( T \) be a tree with at least 2 vertices. Show that \( T \) must have at least 2 leaves.

Solution. We use Theorem (Handshake Theorem) to prove this (i.e. \( \sum \deg(v) = 2m \)). If a tree has less than 2 leaves, then the sum of the degrees of the vertices would be at least

\[
1 + 2(n - 1) = 2n - 1
\]

(in the worst-case, we have 1 leaf and \( n - 1 \) vertices with degree 2). This value must equal \( 2m \), which is always equal to \( 2(n - 1) = 2n - 2 \) in a tree. This is a contradiction since \( 2n - 1 > 2n - 2 \).

Exercise 8.5 (Max degree is at most number of leaves).
Let \( T \) be a tree with \( L \) leaves. Let \( \Delta \) be the largest degree of any vertex in \( T \). Prove that \( \Delta \leq L \).
Solution. It is instructive to recall all 3 proofs.

Proof 1: We use Theorem (Handshake Theorem). The degree sum in a tree is always \(2n - 2\) since \(m = n - 1\). Let \(v\) be the vertex with maximum degree \(\Delta\). The vertices that are not \(v\) or leaves must have degree at least 2 each, so the degree sum is at least \(\deg(v) + L + 2(n - L - 1)\). So we must have \(2n - 2 \geq \deg(v) + L + 2(n - L - 1)\), which simplifies to \(L \geq \deg(v) = \Delta\), as desired.

Proof 2: We induct on the number of vertices. For \(n \leq 3\), this follows by inspecting the unique tree on \(n\) vertices. For \(n > 3\), pick an arbitrary leaf \(u\) and delete it (and all the edges incident to \(u\)). Let \(T - u\) denote this graph, which is a tree (it is connected and acyclic). Also, we let \(L(T)\) denote the number of leaves in \(T\) and \(L(T - u)\) to denote the number of leaves in \(T - u\). We make similar definitions for \(\Delta(T)\) and \(\Delta(T - u)\) regarding the maximum degrees. Note that \(L(T) \geq L(T - u)\). There are two cases to consider:

1. \(\Delta(T - u) = \Delta(T)\)
2. \(\Delta(T - u) = \Delta(T) - 1\)

If case 1 happens, then by the induction hypothesis \(L(T - u) \geq \Delta(T - u) = \Delta(T)\). But this implies \(L(T) \geq \Delta(T)\) (since \(L(T) \geq L(T - u)\)), as desired.

Let \(v\) be the neighbor of \(u\). If case 2 happens, then \(v\) is the only vertex of maximum degree in \(T\). In particular, \(v\) cannot be a leaf in \(T - u\). So \(L(T) = L(T - u) + 1\). The induction hypothesis yields \(L(T - u) \geq \Delta(T - u) = \Delta(T) - 1\). Combining this with \(L(T) = L(T - u) + 1\) we get \(L(T) \geq \Delta(T)\), as desired.

Proof 3: Let \(v\) be a vertex in the tree such that \(\deg(v) = \Delta\). Consider the graph \(T - v\) obtained by deleting \(v\) and all the edges incident to it. Since \(T\) is a tree, we know that \(T - v\) contains \(\Delta\) connected components; let us denote them \(T_1, \ldots, T_\Delta\). Since \(T\) is acyclic, each of the \(T_i\)'s are also acyclic. Since each \(T_i\) is connected and acyclic, each one is a tree. There are two possibilities for each \(T_i\):

(i) \(T_i\) consists of a single vertex. Then that vertex is a leaf in \(T\).

(ii) \(T_i\) is not a single vertex, and so has at least 2 leaves (by Exercise (A tree has at least 2 leaves)). At least one of these leaves is not connected to \(v\) and therefore must be a leaf in \(T\).

In either case, one vertex in \(T_i\) is a leaf in \(T\). This is true for all \(T_1, \ldots, T_\Delta\). Hence we have at least \(\Delta\) leaves in \(T\).

Exercise 8.6 (MST with negative costs).
Suppose an instance of the Minimum Spanning Tree problem is allowed to have negative costs for the edges. Explain whether we can use the Jarník-Prim algorithm to compute the minimum spanning tree in this case.

Solution. Yes, we can. Assign a rank to each edge of the graph based on its cost: the highest cost edge gets the highest rank and the lowest cost edge gets the lowest rank. When making its decisions, the Jarník-Prim algorithm only cares about the ranks of the edges, and not the specific costs of the edges. The algorithm would output the same tree even if we add a constant \(C\) to the costs of all the edges since this would not change the rank of the edges. And indeed, adding a constant to the cost of each edge does not change what the minimum spanning tree is. Hence, we can turn any instance with negative costs into an equivalent one with non-negative costs by adding a large enough constant to all the edges without changing the tree that is output.
(Note: In fact the original algorithm would output the minimum cost spanning tree even if the edge costs are allowed to be negative. There is not even a need to add a constant to the edge costs.)
Exercise 8.7 (Maximum spanning tree).
Consider the problem of computing the maximum spanning tree, i.e., a spanning tree that maximizes the sum of the edge costs. Explain whether the Jarník-Prim algorithm solves this problem if we modify it so that at each iteration, the algorithm chooses the edge between $V'$ and $V \setminus V'$ with the maximum cost.

Solution. Let $(G, c)$ be the input, where $G = (V, E)$ is a graph and $c : E \rightarrow \mathbb{R}^+$ is the cost function. Let $c' : E \rightarrow \mathbb{R}^-$ be defined as follows: for all $e \in E$, $c'(e) = -c(e)$. Let $A_{\min}$ be the original Jarník-Prim algorithm and let $A_{\max}$ be the Jarník-Prim algorithm where we pick the maximum cost edge in each iteration. There are a couple of important observations:

1. The minimum spanning tree for $(G, c')$ is the maximum spanning tree for $(G, c)$.
2. Running $A_{\max}(G, c)$ is equivalent to running $A_{\min}(G, c')$, and they output the same spanning tree.

From Exercise (MST with negative costs), we know $A_{\min}(G, c')$ gives us a minimum cost spanning tree. So $A_{\max}(G, c)$ gives the correct maximum cost spanning tree.

Exercise 8.8 (Kruskal’s algorithm).
Consider the following algorithm for the MST problem (which is known as Kruskal’s algorithm). Start with MST being the empty set. Go through all the edges of the graph one by one from the cheapest to the most expensive. Add the edge to the MST if it does not create a cycle. Show that this algorithm correctly outputs the MST.

Solution. We do not have the solution to this problem at this time.

Exercise 8.9 (Cycle implies no topological order).
Show that if a directed graph has a cycle, then it does not have a topological order.

Solution. Let $G = (V, A)$ be a directed graph and suppose $u_1, u_2, \ldots, u_k, u_1$ is a cycle in $G$. This means that for all $i \in \{1, 2, \ldots, k-1\}$, $(u_i, u_{i+1}) \in A$, and $(u_k, u_1) \in A$. If there is a topological order $f$ of $G$, then by definition, it must be the case that

$$f(u_1) < f(u_2) < f(u_3) < \cdots < f(u_k) < f(u_1).$$

This implies $f(u_1) < f(u_k) < f(u_1)$, which is impossible.

Exercise 8.10 (Topological sort, correctness of naïve algorithm).
Show the algorithm above correctly solves the topological sorting problem, i.e., show that for $(u, v) \in A$, $f(u) < f(v)$. What is the running time of this algorithm?

Solution. We will use the following observation: if an algorithm removes an edge $(u, v) \in A$, then it must be because $v$ is chosen as a sink vertex and removed from the graph.

We now prove that the algorithm is correct by a proof by contradiction. Suppose $(u, v) \in A$ such that $f(u) > f(v)$. This means that $u$ was removed from the graph before $v$ was removed. At the moment that $u$ is removed, $u$ must be a sink (i.e. it must not have any outgoing edges). This implies the edge $(u, v)$ must have been removed at a previous iteration. But the only way
$(u, v)$ would be removed is if $v$ was chosen to be a sink vertex and removed. This implies that $v$ must have been removed before $u$, which is the desired contradiction.

A straightforward implementation of the algorithm would result in a running time of at least $\Omega(n^2)$ since the algorithm has $n$ iterations, and in each iteration, a sink vertex must be found and removed (which takes $\Omega(n)$ steps).