University of California, Los Angeles
CS 281 Computability and Complexity

Instructor: Alexander Sherstov
Scribe: Yiwen Kou
Date: January 17, 2024

LECTURE
3

Deterministic Computation

In this lecture, we will embark on a comprehensive journey through the world of Turing machines, beginning with a fundamental definition of a Turing machine. Following this, we will delve into a practical example, illustrating how a Turing machine can be configured to solve the palindrome recognition problem.

Next, our focus will shift to evaluating the efficiency of Turing machines, a critical aspect in understanding their computational capabilities. We will then explore various alternative definitions of Turing machines. This exploration includes discussions on machines with restricted alphabets, bidirectional Turing machines, and single-tape Turing machines. In each case, we will examine how these modifications impact the overall efficiency of the machine, especially in the context of transitioning from a multi-tape to a single-tape configuration.

Finally, we will culminate our discussion by addressing the lower bound of computational complexity involved in recognizing palindromes using a single-tape Turing machine. This will provide valuable insights into the inherent limitations and challenges associated with this specific computational task.

3.1 Turing Machines Defined

The concept of a Turing machine (TM) is fundamental in the study of computation and computational theory. A particularly versatile form of this machine is the k-tape Turing machine, as depicted in Figure 3.1. This machine is characterized by its use of k tapes, where each tape is an infinite one-directional line of cells. These cells are capable of holding a symbol from a predefined, finite set of alphabets.

In the k-tape Turing machine, the first tape is uniquely designated as the input tape. This tape is read-only, serving the sole purpose of providing input data to the machine. The remaining $k-1$ tapes are read-write tapes, among which the first $k-2$ tapes are referred to as work tapes. These tapes are used for intermediate computations and data manipulations. The final tape in this sequence is earmarked as the output tape. This tape holds particular significance as it is where the machine writes its final result before halting its computation process.

Definition 3.1. A Turing machine (TM) is described by a tuple (Γ, Q, δ) containing:

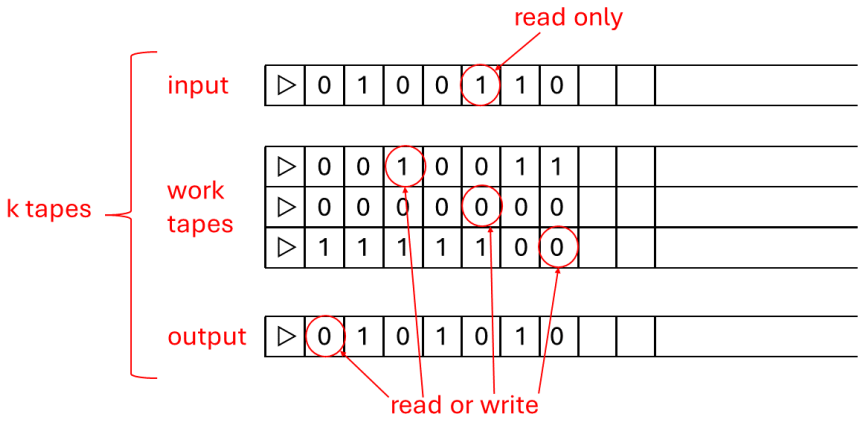

FIGURE 3.1: A 5-tape Turing machine M with an input tape, three work tapes, and an output tape.

- Γ is a finite alphabet that M's tapes can contain and $\{0, 1, \square, \triangleright\} \subseteq \Gamma$. Here, $\square$ is the "blank" symbol, $\triangleright$ is the "start" symbol.
- Q is a set of states that M's register can be in. We assume that $\{\text{START}, \text{HALTED}\} \subseteq Q$.
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\text{L}, \text{R}, \text{S}\}^k$ is the transition function of M describing the rule M uses in performing each step, where k is the number of tapes arbitrary but fixed.

If the machine is in state $q \in Q$ and $(\sigma_1, \sigma_2, \cdots, \sigma_k)$ are the symbols currently being read in the k tapes, and $\delta(q, \sigma_1, \sigma_2, \cdots, \sigma_k) = (q', \sigma'_2, \cdots, \sigma'_k, z_1, z_2, \cdots, z_k)$ where $z_i \in \{\text{L}, \text{R}, \text{S}\}$ then at the next step the $\sigma_2, \cdots, \sigma_k$ symbols in the last $k-1$ tapes will be replaced by the $\sigma'_2, \cdots, \sigma'_k$ symbols, the machine will be in state q', and the k heads will move left, right or stay in place, as given by $z_1, z_2, \cdots, z_k$.

3.2 Example TM Palindromes

In this section, we will explore a practical example of a Turing machine (TM) designed to identify palindromes. Palindromes are sequences of characters that read the same forwards and backwards, a concept we'll define more formally below.

DEFINITION 3.2 (Palindrome). A string $x \in \{0, 1\}^n$ is called a palindrome if $x_1 x_2 \cdots x_n = x_n x_{n-1} \cdots x_1$.

With this definition in mind, we propose the construction of a Turing machine specifically for recognizing palindromes. This TM will be characterized by the following features:

- It utilizes three tapes: an input tape, a work tape, and an output tape.
- The alphabet $\Gamma = \{0, 1, \square, \triangleright\}$.

- The state set $Q = \{\text{START}, \text{COPY}, \text{LEFT}, \text{TEST}, \text{HALTED}\}$.

The operational process of this TM is outlined as follows:

1. It begins by copying the input from the input tape to the work tape.
2. Subsequently, the TM returns the input tape's head to the start of the tape.
3. During operation, if the machine encounters two different values at any point, it halts and outputs 0 on the output tape, indicating the input is not a palindrome.
4. If no discrepancies are found, the machine halts and outputs 1 on the output tape, confirming the input is a palindrome.

Let us now delve into the specifics of the transition function:

- In the **START** state, the TM moves the input-tape head right, while also moving the work-tape head right and writing the start symbol $\triangleright$. It then transitions to the **COPY** state.
- In the **COPY** state, if the symbol read from the input tape is not $\square$, the TM moves both the input and work-tape heads right, replicating the input tape's symbol onto the work tape, and remains in the **COPY** state. If $\square$ is read, it moves the input-tape head left and transitions to the **RIGHT** state.
- In the **RIGHT** state, if the symbol on the input tape is not $\triangleright$, the TM moves the input head left, keeping the work-tape head stationary, and stays in the **RIGHT** state. Upon reading $\triangleright$, it shifts the input-tape head right and the work-tape head left, then changes to the **TEST** state.
- In the **TEST** state, if $\square$ is read on the input tape and $\triangleright$ on the work tape, the TM writes 1 on the output tape and transitions to **HALT**. If the symbols read from the input and work tapes differ, it writes 0 on the output tape and also transitions to **HALT**. If the symbols match, it moves the input tape head right and the work-tape head left, continuing in the **TEST** state.

This Turing machine exemplifies a methodical approach to palindrome recognition, showcasing the intricate yet logical process that TMs follow to execute seemingly simple tasks.

3.3 Efficiency

This section is dedicated to exploring the methodology for estimating the efficiency of a Turing machine when it computes a function $f : \{0,1\}^* \to \{0,1\}^*$. We begin by defining an essential function $T : \mathbb{N} \to \mathbb{N}$ for this purpose.

Definition 3.3. We define the computation of a function f by a Turing machine in terms of time complexity. Specifically, a Turing machine is considered to compute f in time $T(n)$, where n denotes the length of the input, if the machine, given an input $x \in \{0,1\}^*$ on its input tape, halts within $T(|x|)$ steps. Here, $|x|$ represents the bit length of x and upon halting, the machine outputs $f(x)$ on its output tape.

To illustrate this concept, let's consider the time complexity of the Turing machine constructed in the previous section for recognizing the palindrome.

EXAMPLE 3.4. Consider the problem of palindrome recognition. A Turing machine designed for this task demonstrates the computation within a linear time complexity $O(n)$.

Transitioning from specific cases to a broader definition, we next define the concept of a language in the context of Turing machines.

DEFINITION 3.5. A language is any subset $L \subseteq \{0,1\}^*$.

Building on the concept of languages, we now explore how a Turing machine interacts with these languages.

DEFINITION 3.6. A Turing machine M is said to decide a language L if it effectively computes the associated characteristic function as follows:

$$f_L(x) = \begin{cases} 1, & x \in L. \\ 0, & x \notin L. \end{cases}$$

This function serves as a binary indicator of whether a given input belongs to the language L.

Finally, we introduce a formal classification of languages based on the time complexity of the Turing machines that decide them.

DEFINITION 3.7. The set $\mathrm{DTIME}(T(n))$ is defined to categorize languages based on the decidability time of Turing machines. It is expressed as:

$$\mathrm{DTIME}(T(n)) = \{L : L \text{ is decidable in time } cT(n) \text{ for some constant } c > 0\}.$$

Expanding this notion, we define the complexity class P as the union of DTIME functions for polynomial time complexities:

$$P = \mathrm{DTIME}(n) \cup \mathrm{DTIME}(n^2) \cup \mathrm{DTIME}(n^3) \cup \cdots .$$

3.4 Other Definitions of TMs

In this section, we aim to broaden our understanding of Turing machines (TMs) by examining alternative definitions and configurations. We will focus on how variations in the machine's alphabet and tape structure can impact its computational efficiency.

One significant aspect of a TM's design is its alphabet. We will now consider the effects of limiting the alphabet Γ to the set $\{0, 1, \triangleright, \square\}$ and how it impacts the time complexity of computing functions.

CLAIM 3.8. *If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ with alphabet Γ, then f is computable in time $O(T(n) \log |\Gamma|)$ with alphabet $\{0, 1, \triangleright, \square\}$.*

Proof. Let M be a TM with alphabet Γ, k tapes, and state set Q that computes the function f in $T(n)$ time. Since one can encode any element of Γ using $\log |\Gamma|$ bits, we consider the

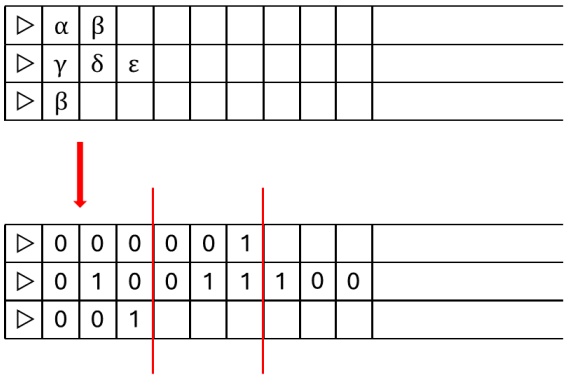

FIGURE 3.2: This illustration demonstrates the process of restricting a larger alphabet, initially comprising $\{\triangleright, \square, \alpha, \beta, \gamma, \delta, \epsilon\}$, down to a more compact set of $\{\triangleright, \square, 0, 1\}$.

following TM $\tilde{M}$ computing f with alphabet $\{0, 1, \triangleright, \square\}$, k tapes: each of $\tilde{M}$'s work tapes will simply encode one of M's tapes: for every cell in M's tape we will have $\log |\Gamma|$ cells in the corresponding tape of $\tilde{M}$, as shown in Figure 3.2. To simulate one step of M, the machine $\tilde{M}$ will:

1. use steps to read from each tape the $\log |\Gamma|$ bits encoding a symbol of Γ,
2. use its state register to store the symbols read,
3. use M's transition function to compute the symbols M writes and M's new state given this information,
4. store this information in its state register,
5. use $\log |\Gamma|$ steps to write the encoding of these symbols on its tapes.

It is not hard to see that for every input $x \in \{0,1\}^n$, if on input x the TM M outputs $f(x)$ within $T(n)$ steps, then $\tilde{M}$ will output the same value within less than $O(\log |\Gamma| T(n))$ steps. □

Another aspect worth considering is the directional nature of the tapes used by TMs. We investigate the impact of using bidirectional tapes, which are infinite in both directions, as opposed to standard unidirectional tapes.

CLAIM 3.9. *Define a bidirectional Turing machine to be a turning machine whose tapes are infinite in both directions. If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ with bidirectional tapes, f is computable in time $O(T(n))$ using standard tapes.*

Proof. If M uses alphabet Γ then $\tilde{M}$ will use the alphabet Γ^2. We encode a tape of M that is infinite in both direction using a standard tape by "folding" it in an arbitrary location, with each location of $\tilde{M}$'s tape encoding two locations of M's tape, as shown in Figure 3.3.

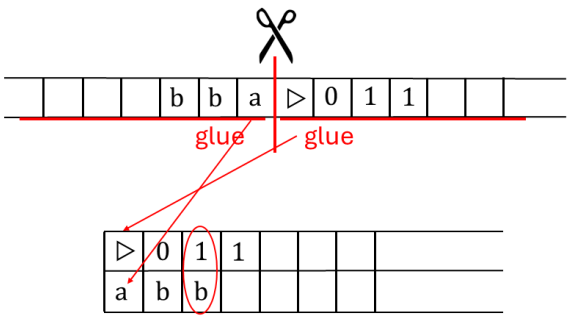

FIGURE 3.3: Illustration of "folding" a one-directional tape to create a bi-directional tape.

At first, $\tilde{M}$ will ignore the second symbol in the cell it reads and act according to M's transition function. However, if this transition function instructs $\tilde{M}$ to go "over the edge" of its tape then instead it will start ignoring the first symbol in each cell and use only the second symbol. When it is in this mode, it will translate left movements into right movements and vice versa. If it needs to go "over the edge" again then it will go back to reading the first symbol of each cell, and translating movements normally. □

Finally, we examine a single-tape Turing machine, equipped with a sole read/write tape that functions concurrently as the input, work, and output tape. We aim to demonstrate that this configuration does not significantly impact the machine's efficiency in terms of the time required to compute functions.

THEOREM 3.10. *If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ on a k-tape Turing machine, then f is computable in time $O(T(n)^2)$ on a single-tape Turing machine.*

Proof. Assume we have a k-tape Turing machine, denoted as M, that computes a function f in time $T(n)$. Our objective is to construct a single-tape Turing machine, $\tilde{M}$ which simulates M and computes the same function within $O(T(n)^2)$ time.

The machine $\tilde{M}$ uses an extended alphabet Γ^k, where each symbol in Γ^k represents a combination of symbols from the k tapes of M, as shown in Figure 3.4. This allows $\tilde{M}$ to track the state of all k tapes of M on its single tape.

To simulate a step of M, $\tilde{M}$ performs a full left-to-right scan of its tape. During this scan, $\tilde{M}$ reads and memorizes the current symbols under the heads of M (represented in the extended alphabet). After the first scan, $\tilde{M}$ performs another full scan, this time to update the tape based on the actions M would have taken.

Now we analyze the time complexity. Each of M's steps is simulated by two full scans of $\tilde{M}$'s tape. Since on n-length inputs M never reaches more than location $T(n)$ of any of its tapes, $\tilde{M}$ will never need to reach more than location $kT(n)$ of its work tape, meaning that for each the at most $T(n)$ steps of. If the length of the tape used by M is at most $T(n)$ steps of M, $\tilde{M}$ performs at most $O(kT(n))$ work steps, so the total time taken by $\tilde{M}$ will be $O(kT(n)^2)$. □

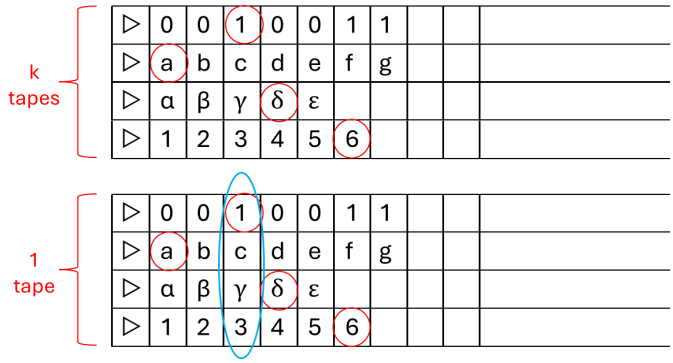

FIGURE 3.4: Conversion of a k-tape Turing machine into a single-tape Turing machine. In this transformation, the single-tape machine interprets k cells, each corresponding to the same position across the k tapes of the original machine, as one composite cell, as highlighted by the blue circle. To effectively simulate the operations of the k-tape Turing machine, the single-tape machine must account for the possibility that the heads of the original tapes may occupy different positions at any given time, as indicated by the red circles. This necessitates scanning through the entire tape of the single-tape machine to identify and appropriately modify these positions.

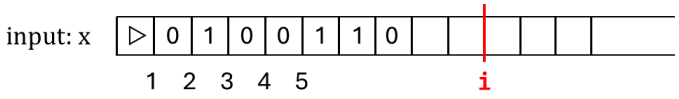

FIGURE 3.5: Illustration of i-th tape cell boundary.

3.5 Recognizing Palindromes on a Single-Tape TM

In this section, our focus shifts to the task of recognizing palindromes using a single-tape Turing machine (TM). We aim to establish the lower bound of time complexity for this specific problem. As a preliminary step, it is crucial to introduce and gain an understanding of the concept of a crossing sequence and its associated properties, which play a vital role in our analysis.

DEFINITION 3.11 (Crossing Sequence). The crossing sequence, denoted as $C_i(x)$, refers to the ordered sequence of states $q_1, q_2, q_3, \cdots$ encountered by a Turing machine M given input x. Here, q_k represents the state of M immediately after it crosses the i-th tape cell boundary for the k-th time. (See Figure 3.5)

This concept leads us to an important property of crossing sequences which is pivotal in understanding the behavior of Turing machines, particularly in the context of palindrome recognition.

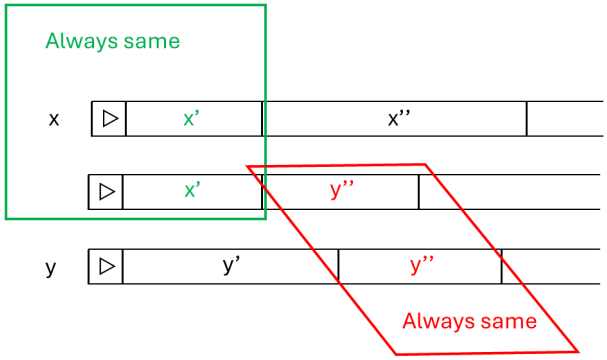

FIGURE 3.6: This figure illustrates Lemma 3.12. It demonstrates that if input x and y share an identical crossing sequence, the Turing machine's behavior on the concatenated string $x'y''$ will mirror its actions on the corresponding segments of x and y.

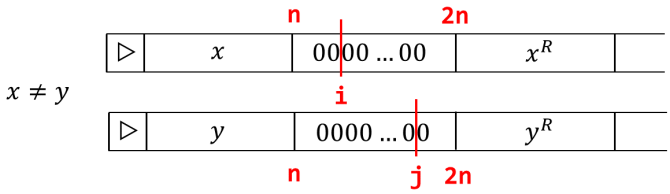

FIGURE 3.7: In this depiction, x and y represent distinct binary strings, each of length n, consisting of the digits 0 and 1. The illustration pertains to the proof of Theorem 3.13, where a crucial consideration is the analysis of crossing sequences occurring at positions $n, n+1, \cdots, 2n$.

LEMMA 3.12. *Consider two distinct inputs x and y, where x can be decomposed as $x'x''$ and y as $y'y''$. Given that the crossing sequences $C_i(x) = C_j(y)$ and that the machine M produces the same output for both inputs (i.e. $M(x) = M(y)$), it follows that $M(x'y'') = M(x) = M(y)$.*

THEOREM 3.13. *Any 1-tape Turing machine M that decides palindromes runs in time $\Omega(n^2)$.*

Proof. Here we only consider inputs of pattern $x0^n x^R$ where $x \in \{0,1\}^n$ and x^R is the reverse of x. Suppose $x \neq y$. If $C_i(x0^n x^R) = C_j(y0^n y^R)$ for some $i, j \in \{n, n+1, \cdots, 2n\}$, then $M(x00\cdots0y^R) = 1$ by Lemma 3.12 (see Figure 3.7), which is a contradiction since $x00\cdots0y^R$ is not a palindrome. This means that for any distinct x and y, $C_i(x0^n x^R) \neq C_j(y0^n y^R)$ for all $i, j \in \{n, n+1, \cdots, 2n\}$. Next, we prove that there exists some input $x0^n x^R$ with "long" crossing sequences at all of $n, n+1, n+2, \cdots, 2n$. Let t_x be the length of the shortest crossing sequence in $\{C_i(x0^n x^R) | n \leq i \leq 2n\}$, and let i_x be the crossing index among $n, n+1, \cdots, 2n$ with the shortest crossing sequence of x, that is,

$i_x = \arg\min_{n \le i \le 2n} |C_i(x0^n x^R)|$. Let $t = \max_{x \in \{0,1\}^n} t_x$ and $\tilde{x} = \arg\max_{x \in \{0,1\}^n} t_x$. The number of crossing sequences of length $\le t$ is:

$$1 + |Q| + |Q|^2 + |Q|^3 + \cdots + |Q|^t = \frac{|Q|^{t+1} - 1}{|Q| - 1}.$$

Since $C_{i_x}(x)$ and $C_{i_y}(y)$ are different for distinct $x, y \in \{0,1\}^n$, so we have

$$\frac{|Q|^{t+1} - 1}{|Q| - 1} \ge 2^n,$$

which gives us $t = \Omega(n)$ which means the length of crossing sequence for $\tilde{x}$ at $n, n+1, \cdots, 2n$ is $\Omega(n)$. Notice that each element in each crossing sequence of $\tilde{x}$ corresponds to a move of M, this implies that M runs in time $\Omega(n^2)$. □