CS 6810: Theory of Computing **Fall 2021**

Lecture 1: Aug 26, 2021

Lecturer: Eshan Chattopadhyay *Scribe: Atul Ganju*

1 Introduction

Broadly speaking, the aim of this course is understanding the "intrinsic" complexity of a wide range of computational problems. The complexity of a problem is of course dependent on the computational resources at hand (e.g. time, space, randomness etc).

In general, the results about the complexity of a problem comes in two favors: upper and lower bounds of computation.

An upper bound result implies that with a certain set of resources at your disposal, defined by your upper bound, you can solve a computational problem. As an example, the shortest path between s and t in an undirected graph $G = (V, E)$ can be found in $\mathcal{O}((|E| + |V|log(|V|)))$ time (using the Dijkstra's algorithm).

A lower bound result implies that you need a certain set of resources to solve a problem regardless of your choice of an algorithm. Thus, a statement of the form: "There exists a problem in NP for which there is no polynomial time algorithm" is an example of a lower bound result (of course note that this example would imply P$\neq$NP).

2 Models of Computation

A model of computation describes how an output of a mathematical function is computed given an input. There are two types of models of computation that we will see in this class: uniform and non-uniform.

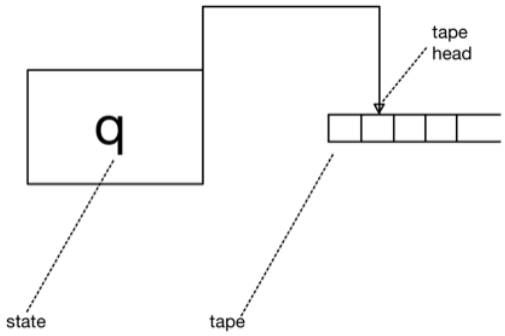
Definition 2.1 (**Uniform Model of Computation**). *a single machine is used to compute the value of a function for all possible inputs (e.g. finite automaton, Turing machines, etc).*

Definition 2.2 (**Non-Uniform Model of Computation**). *A family of programs used to compute the value of a function, where, depending on input size, a different program is used to compute the value of the function (i.e. given a family of functions $\{f_n\}$ where each only takes in inputs of length n, there exists a machine which computes f on an input x of length l where $f(x) := f_l(x)$). We will see examples of non-uniform models of computation later in the course.*

Now we discuss Turing machines (TMs). A Turing machine has an input tape, an output tape and potentially, one or more work tapes. Tapes are divided into cells, each of which can record a single symbol. There is a reading head on each tape which can be moved one step along the tape in a single operation done by the TM. It also has a finite state which can be used to store information.

Now, lets define some standard notation. We take Γ to be the tape alphabet, Σ to be the input alphabet, and Q to be the set of states for the TM. Note that $|\Gamma|, |\Sigma|, |Q|$ are all $O(1)$, or in other words the size of these sets are not dependent on input size. Finally, for every Turing machine, there is a corresponding transition function δ, which takes as input the values on each of the readable tapes and the current state of the Turing machine and outputs a new state for the Turing Machine along with what to write onto the tapes and in which direction to move each of the read/write heads on each of the tapes.

2.1 Single-Tape Turing Machine



The first computational model we discuss is the single-tape Turing machine.

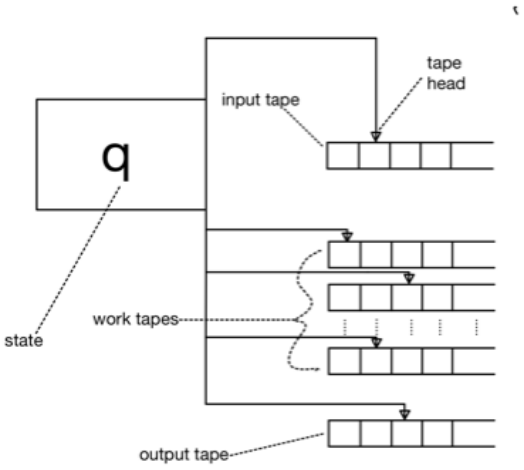
Definition 2.3 (**Single-Tape Turing Machine**)**.** *A single-tape Turing machine is defined as follows:*

- *There is a single tape that is read/write.*
- Γ *is the alphabet of the aforementioned tape (as stated before, a finite set of symbols).*
- Q *is a finite set of internal states.*
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$ *represents the state change, tape rewrite and tape head movement (Left, Stay or Right) given the current state and the characters under the reading heads.*

A drawback of this model of computation is that it loses efficiency (polynomial factors) even for rather simple problems.

Example 2.4. *Define* PALINDROME $: \{0,1\}^* \to \{yes, no\}$ *to return yes on the input* s *of length* n *if and only if* $s[i] = s[n-i]$ *for all* $i \in \{0, 1, 2, ..., n\}$*. Using this model of computation,* PALINDROME *takes time* $O(n^2)$*.*

2.2 Multi-Tape Turing Machine



Definition 2.5 (**Multi-Tape Turing Machine**). *A k-tape Turing machine for $k >= 3$ is defined as follows:*

- *There is an input tape that is read only.*
- *There are $k-2$ work tapes that are read/write.*
- *There is one output tape that is read/write.*
- *Q is a finite set of internal states*
- *$\delta : Q \times \Sigma^2 \times \Gamma^k \to Q \times \Gamma^k \times \Sigma \times \{L, S, R\}^{k+2}$ represents the state change, tape rewrite and tape head movement (Left, Stay or Right) given the current state and the characters under the reading heads.*

Using this model of computation, node that PALINDROME can be computed in $O(n)$ time.

The following theorem attests to the fact that TM model of computation is robust, and various design choices do not significantly change the notion of efficiency.

Theorem 2.6. *For every $f : \{0,1\}^* \to \{0,1\}$, time constructible $T : \mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a TM M using k tapes (plus additional input and output tapes) then it is computable in $\mathcal{O}(T(n)^2)$ time by a TM $\hat{M}$ using only a single work tape (plus additional input and output tapes).*

We provide a proof sketch of the above theorem. We start by recognizing that we have to record all k tapes of M onto one tape. To do this, we can use locations $1, k+1, 2k+1, ...$ of $\hat{M}$ to record the values that would've been on the first work tape of M, locations $2, k+2, 2k+2, ...$ to record the values that would've been on the second work tape of M, etc. We also augment the alphabet of $\hat{M}$ such that for every symbol $s \in \Gamma_M$, we have $s, \hat{s} \in \Gamma_{\hat{M}}$. For each set of locations that simulates a tape of M, only one location will have a symbol with a $\hat{}$ on it indicating the position of the corresponding tape head. $\hat{M}$ still uses the input and output tape in the same way that M does. To simulate one step of M, $\hat{M}$ sweeps through its work tape twice: once from left to right to record in its state the symbols the locations of the heads (the symbols on the single work tape marked with a $\hat{}$) and then again to update the tape according to the directions given by M's transition function. Both TMs will have the same output. Also, on an input of length n, if M only reaches location $T(n)$ on all of its tapes, $\hat{M}$ will never reach more than location $kT(n)$. So for each step of M, $\hat{M}$ performs proportional to $kT(n)$ steps ($2T(n)$ for the sweep and some extra steps for updating head movement and other book keeping). Thus, if M takes $T(n)$ time to run, $\hat{M}$ can simulate M in $\mathcal{O}(kT^2(n))$ time.