Chapter 2

Complexity Measures

Having established in Chapter 1 how and what a Turing machine computes, we now turn to the definition of how efficiently it computes.

2.1. Definitions

The first two definitions establish the amount of time and space used by a particular Turing machine on a particular input.

Definition 2.1: An alternating Turing machine M on input x *runs in time at most* t if and only if there is an accepting subtree of M on x whose height is at most t.

Definition 2.2: An alternating Turing machine M on input x *runs in space at most* t if and only if there is an accepting subtree of M on x each of whose nodes is labeled by a configuration of length at most t.

The next definition specifies the amount of time or space used by a particular Turing machine on its worst case inputs.

Definition 2.3: An alternating Turing machine M runs in *time* (*space*) $T(n)$ if and only if, for every $x \in L(M)$, M on input x runs in time (respectively, space) at most $T(|x|)$.

Notice that this definition puts no bound on the time or space used by M on any x not accepted by M.

Definition 2.4: DTIME($T(n)$) = $\{L \mid$ there is a multitape deterministic Turing machine that accepts L and runs in time $T(n)\}$.

Definition 2.5: DSPACE($T(n)$) = $\{L \mid$ there is a multitape deterministic Turing machine that accepts L and runs in space $T(n)\}$.

Definition 2.6: NTIME($T(n)$) = $\{L \mid$ there is a multitape nondeterministic Turing machine that accepts L and runs in time $T(n)\}$.

Definition 2.7: NSPACE($T(n)$) = $\{L \mid$ there is a multitape nondeterministic Turing machine that accepts L and runs in space $T(n)\}$.

Definition 2.8: ATIME($T(n)$) = $\{L \mid$ there is a multitape alternating Turing machine that accepts L and runs in time $T(n)\}$.

Definition 2.9: ASPACE($T(n)$) = $\{L \mid$ there is a multitape alternating Turing machine that accepts L and runs in space $T(n)\}$.

Convention 2.10: Throughout this text, n will refer to the length of the input string.

2.2. Review of Order Notation

Let $\mathcal{R}$ be the set of real numbers, and $f, g : \mathcal{R} \to \mathcal{R}$ be two functions. The following definitions provide a convenient notation for comparing the rates of growth of f and g. (See Knuth [25] for more discussion.)

- $f(n) = O(g(n))$ if and only if $(\exists c)(\exists n_0)(\forall n \geq n_0)\ |f(n)| \leq cg(n)$.
- $f(n) = \Omega(g(n))$ if and only if $(\exists c)(\exists n_0)(\forall n \geq n_0)\ f(n) \geq cg(n)$.
- $f(n) = \Theta(g(n))$ if and only if $(\exists c)(\exists c')(\exists n_0)(\forall n \geq n_0)\ cg(n) \leq f(n) \leq c'g(n)$.
- $f(n) = o(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) = \omega(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

2.3. Sublinear Space Bounds

The problem with the definition of space as given in Definition 2.2 is that there are no machines with sublinear (i.e., $o(n)$) space bounds: the entire input, whose length is n, is included in the initial configuration, and hence in the space bound. In order to extend these definitions to sublinear space bounds, we will assume the following modifications from now on, unless specified otherwise:

Convention 2.11:

1. There will be a separate input tape of length n plus two endmarkers. This tape is read-only.
2. Configurations contain the position h of the input tape head in binary, but not the contents of the input tape. Therefore, a configuration of a machine with an input tape and k additional worktapes looks like $(h, u_1qv_1, u_2qv_2, \ldots, u_kqv_k)$.
3. Since the input x is no longer part of the configuration, the notation $P \vdash_M Q$ will be replaced by $P \vdash_{M,x} Q$.

4. The definition of space will not include the length of the input tape, nor the length of the input head position h.

Example 2.12: $L = \{ww \mid w \in \{0,1\}^*\}$ can be accepted by a deterministic Turing machine in space $O(\log n)$. This can be accomplished as follows. First M uses a counter to calculate the length n of the input. This takes $O(\log n)$ space on a worktape. If n is odd, M rejects the input. If n is even, M starts at the first input symbol, storing it in the finite control, and uses a second counter to move the input head $n/2$ cells to the right. M rejects if this is not the same symbol as in the finite control. Otherwise M moves the input head left $n/2 - 1$ cells and repeats this check for the second input symbol. M performs this check for each symbol until the input head moves off the right end of the input, at which point M accepts. Each counter contains an index between 1 and n, so M uses space $O(\log n)$.

2.4. Sublinear Time Bounds

The extension to sublinear space bounds suggests doing the same for time bounds. One might argue that any interesting Turing machine cannot have a sublinear time bound, as it takes at least n steps just to read the input. However, this argument fails for nondeterministic or alternating Turing machines, provided they are given random access to the input tape rather than sequential access. Consider, for instance, the language that consists of all strings that contain a 1. A nondeterministic machine could guess and record the position of the 1 in time $O(\log n)$, and use its supposed random access to the input to verify that there is a 1 in that position. This idea of random access or indexing is formalized in the following definition.

Definition 2.13: An *indexing Turing machine* has an input tape with no head, and a special "index tape" in addition to its other worktapes. The next move depends on the ith input symbol if the nonblank portion of the index tape to the left of the head is the binary encoding of i for $1 \leq i \leq n$, and depends on the endmarker symbol otherwise. The length of the nonblank portion of the index tape is included in the machine's space bound.

Convention 2.14: Unless explicitly stated otherwise, we will assume from now on that all deterministic and nondeterministic Turing machines are not indexing machines, but that all other alternating Turing machines are.

Example 2.15: An indexing nondeterministic Turing machine can calculate the length n of the input as follows. On the index tape guess n one bit at a time, verify that the nth input symbol is in the input alphabet, and verify that the $(n+1)$st symbol is an endmarker, rejecting if either of these is not the case. It takes time $O(\log n)$ to do this.

Example 2.16: The complement of the language $L = \{ww \mid w \in \{0,1\}^*\}$ from Example 2.12 can be accepted by an indexing nondeterministic Turing machine in time $O(\log n)$. Compute the length n of the input as in Example 2.15. If n is odd, accept. Otherwise, guess i satisfying $1 \leq i \leq n/2$, and accept if and only if $x_i \neq x_{i+n/2}$, where $x_1 x_2 \ldots x_n$ is the input. The sum $i + n/2$ and the other computations necessary can be calculated in time $O(\log n)$.

Any nondeterministic Turing machine requires time at least n to accept the language L from Example 2.16, but it only takes an alternating Turing machine time $O(\log n)$ to do the same. These are both left as simple exercises. Here is an example of a slightly more interesting language that can be accepted by an alternating Turing machine in $O(\log n)$ time:

Example 2.17: Let L be the set of strings $A\#B$ that are encodings of two equal sets (which we will also refer to as A and B). More specifically, $A = A_0\$A_1\$\ldots\$A_a$ and $B = B_0\$B_1\$\ldots\$B_b$, where for simplicity $A_i, B_j \in \{0,1\}^m$ for some $m = 2^p - 1$.

The alternating Turing machine first determines n, m, a, b, and h where h is the index of the marker $\#$, in a manner similar to that of Example 2.15. The goal is to accept if and only if $A \subseteq B$ and $B \subseteq A$, that is,

$$(\forall i)(\exists j)A_i = B_j \text{ and } (\forall j)(\exists i)B_j = A_i.$$

The machine will universally check each of these two conjuncts. Here is how it checks the first, the second being done in an analogous manner:

1. Universally choose and record i, with $0 \leq i \leq a$.
2. Existentially choose and record j, with $0 \leq j \leq b$.
3. Universally choose and record k, with $1 \leq k \leq m$.
4. Accept if and only if $A_{i,k} = B_{j,k}$, where these are the kth bits of A_i and B_j, respectively.

The machine requires time $O(\log a)$ to universally choose i, time $O(\log b)$ to existentially choose j, and time $O(\log m)$ to universally choose k; furthermore, a, b, and m are each at most n. To find $B_{j,k}$ in step 4, the alternating Turing machine needs to calculate $h + j2^p + k$ on its index tape, which can be done in $O(\log n)$ time.

Alternating Turing machines, and particularly indexing alternating Turing machines, may seem unmotivated at first, but there is a direct correspondence to Boolean circuits (Ruzzo [39]) that makes them an excellent model of parallel computations; this will be explored further in Section 7.5. Furthermore, they will turn out to be extremely helpful in understanding the complexity of natural problems, much as nondeterministic Turing machines have proven to be. *Add other forward pointers.*

2.5. Exercises

1. Let I be the set of invertible $\sqrt{n} \times \sqrt{n}$ matrices over $\mathcal{Z}_2$ (the integers modulo 2). Show that $I \in \text{NTIME}(O(n^{3/2})) \cap \text{NSPACE}(O(\sqrt{n}))$.

2. Prove that an indexing deterministic Turing machine can compute the length n of its input in time $O(\log n)$.

3. Show that the language L from Example 2.16 cannot be recognized by an indexing nondeterministic Turing machine in time less than n.

4. Prove that Set Equality (from Example 2.17) can be solved by an alternating Turing machine in time $O(\log n)$ for *arbitrary* subsets of $\{0,1\}^*$.

5. Show that every regular language is in $\text{DTIME}(O(n)) \cap \text{DSPACE}(O(1))$.

6. Show that every regular language is in $\text{ATIME}(O(\log n))$.