Time Complexity

1 Time complexity

Reading: Sipser chapter 7.

From now on we will be interested not only in whether a Turing machine can compute a function, but also in how many resources (space, time) uses in order to do so.

Definition 1 (Running time). *Let M be a deterministic Turing machine that halts on all inputs. The running time (time complexity) of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M takes on an input of length n. We say that M runs in time $f(n)$ and M is an $f(n)$ time Turing machine.*

We will measure running time asymptotically.

Definition 2 (Upper bounds, Big-O). *Let $f, g : \mathbb{N} \to \Re^+$. We say that $f(n) = O(g(n))$ if there exist $c, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, we have $f(n) \leq c \cdot g(n)$. In this case the function g is said to be an asymptotic upper bound on f.*

Upper bounds of the form n^c, where $c > 0$ is a constant, are called polynomial bounds, while upper bounds of the form $2^{(n^c)}$ are called exponential bounds.

Definition 3 (Small-o). *Let $f, g :\to \mathbb{N} \to \Re$. We say that $f(n) = o(g(n))$ if*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

That is, for every $c > 0$ there exists n_0 such that $f(n) < c \cdot g(n)$ for all $n \geq n_0$.

Example 1. *What is the runtime of the machine that decides the language $A = \{0^k 1^k \mid k \geq 0\}$? Consider a TM M that solves this problem. On input w:*

1. *Check that the input has the desired form 0^*1^*.*
2. *While there are both 0s and 1s left on the tape:*
   - *Move the tape head from left to right, crossing off one zero and one one. Then go back to the leftmost end of the tape.*
3. *If no 0s and 1s are left, accept. Otherwise, reject.*

The first operation of checking the form of the input takes $O(n)$ steps. In the second step, the tape head moves from the left end of the tape to the right end of the input, and then back, which gives $2n$ steps. Each iteration of step 2 crosses off at most 2 symbols, so there will be at most $n/2$ iterations of this type. In the third step, the tape head moves across the tape one more time, for a total of n steps. Thus in total we have $O(n^2)$ steps.

Recall that a Turing machine M *decides* a language $L \in \{0,1\}^*$ if for any word $w \in \{0,1\}^*$, the machine M accepts if $w \in L$ and rejects if $w \notin L$.

Definition 4 (The class DTIME)**.** *Let $f : \mathbb{N} \to \mathbb{N}$ be a function. A language L is in $DTIME(t(n))$ if there exists a Turing machine running in $O(t(n))$ steps and decides L.*

2 The class P

Definition 5 (The class P)**.** *P is the class of languages decidable in polynomial time (on deterministic, one-tape) Turing machines:*

$$P = \bigcup_k DTIME(n^k)$$

The class P:

- remains invariant for all models of computation that are polynomially-time equivalent to deterministic one-tape Turing machines
- corresponds to problems that are realistically solvable on a real computer.

Some examples of problems in P are as follows.

Example 2 (PATH)**.** *Consider the language*

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph and there is a path from } s \text{ to } t\}$$

Theorem 2.1. *$PATH \in P$.*

Proof. Consider the following TM for solving $PATH$. On input $\langle G, s, t\rangle$,

1. Place a mark on node s.
2. Repeat until no new nodes are marked:
   - Scan all the edges of G. If an edge (a, b) is found outgoing from a marked node a to an unmarked node b, mark node b.
3. If t is marked, accept. Otherwise, reject.

□

3 The class NP

Next we will look at problems where coming up with a solution is (intuitively) harder than verifying the solution. For instance, when solving a crossword puzzle, it's harder to solve it than to check that a solution given by someone else. Similarly with a math problem where the hard part is getting the right idea and getting it to work, while checking that a solution is correct (i.e. tracing the sequence of derivations and making sure they follow logically from each other) is easier.

Thus in contrast to P, which is the class of "efficiently solvable problems", NP is the class of "efficiently verifiable solutions".

Definition 6 (Verifier)**.** *A* verifier *for a language L is an algorithm V that takes inputs of the form $\langle w, u\rangle$ and*

$$w \in L \iff \text{there exists } u \text{ such that } V \text{ accepts } \langle w, u\rangle$$

The string u with this property is called a certificate *(or witness) for w. That is,*

$$L = \{w \mid V \text{ accepts } \langle w, u\rangle \text{ for some string } u\}$$

Definition 7 (Polynomial-time verifier)**.** *A* polynomial-time verifier *for a language L is an algorithm V that takes inputs of the form $\langle w, u\rangle$ and runs in time $p(|w|)$ for some polynomial $p : \mathbb{N} \to \mathbb{N}$, such that*

$$w \in L \iff \text{there exists } u \text{ such that } V \text{ accepts } \langle w, u\rangle$$

Definition 8 (The class NP)**.** *NP is the class of languages $L \in \{0,1\}^*$ that have polynomial time verifiers.*

Example 3 (Independent Set)**.** *The language*

$$INDSET = \{\langle G, k\rangle \mid \text{there exists } S \subseteq V(G) \text{ such that } |S| \geq k \text{ and } \forall u, v \in S, (u, v) \notin E(G)\}$$

Consider the following TM M, which on input $\langle G, k, u\rangle$, outputs 1 if and only if u encodes a list of k vertices of G with no edges between them. Thus $\langle G, k\rangle \in INDSET$ if and only if there exists a string u such that $M(\langle G, k\rangle u) = 1$, so $INDSET \in NP$.

The list of vertices u represents the certificate *that $\langle G, k\rangle$ is in $INDSET$. Also note that if n is the number of vertices in G, then a list of k vertices can be encoded using $O(k \log n)$ bits. So $|u| = O(n \log n)$, which is polynomial in the size of the representation of G.*

Example 4 (Graph Isomorphism)**.** *Given two graphs G and H, is there a bijective function $f : V(G) \to V(H)$ such that $(u, v) \in E(G)$ if and only if $(f(u), f(v)) \in E(H)$?*

To show that Graph Isomorphism is in NP, we need to find a polynomial time verifier V for it. The verifier should receives as input two graphs (G, H) and a certificate u, and accept (G, H, u) if and only if G and H are isomorphic. Moreover, the runtime of V must be bounded by $p(|(G, H)|)$ for some polynomial p.

Consider the following algorithm V, which receives as input two graphs G and H on n vertices and a string u:

1. *If $u = (i_1, \ldots, i_n)$ is not a permutation of $(1, \ldots, n)$, reject.*

2. *Else, permute the vertices of G as indicated by u. Check that the permuted graph G is identical to H.*

Step 1 runs in $O(n^2)$ and step 2 runs in $O(n+m)$, where m is the number of edges. So the verifier V runs in time $O(n^2)$, thus the Graph-Isomorphism problem is in NP.

Example 5 (Subset-Sum)**.** *Given natural numbers $U = \{w_1 \dots w_n\}$ and a number W, is there a subset $S \subseteq U$ such that $\sum_{a \in S} a = W$?*

A certificate for this problem is a subset of numbers that sum up to W.