University of Nevada, Las Vegas Computer Science 456/656 Fall 2018

Decidability, Etc.

Let $\mathcal{N}$ be the set of *natural numbers*$= \{1, 2, \ldots\}$, or the set of positive integers. In class, I gave a proof that there is a function $f : \mathcal{N} \to \mathcal{N}$ which is eventually greater than any computable function.

Definition 1 *If $f, g : \mathcal{N} \to \mathcal{N}$ are functions, we say that f is eventually greater than g if there is some integer i such that $f(n) > g(n)$ for all $n \geq i$.*

Theorem 1 *There is a function $f : \mathcal{N} \to \mathcal{N}$ which is eventually greater than any computable function.*

Proof: Each computable function can be implemented as a Turing machine (or a C++ program, if you prefer). There are only countably many Turing machines, hence there are only countably many computable functions.

Let $f_1, f_2, \ldots$ be an enumeration of all computable functions from $\mathcal{N}$ to $\mathcal{N}$. Now define a function f as follows: for any $n \in \mathcal{N}$, let $f(n) = 1 + \sum_{i=1}^{n} f_i(n)$. We claim that f is eventually greater than any computable function.

For all $i \geq 1$ and $n \geq i$, $f(n)$ equals $1 + f_i(n)$ plus possibly additional positive terms. Therefore, f is eventually greater than f_i. Since every computable function is f_i for some i, we are done. ∎

$\mathcal{P}$, $\mathcal{NP}$, Etc.

Definition 2 *We say that a function f is in the class $\mathcal{P}$-TIME, or f is polynomial time, if there is a constant k and a machine M which computes $f(w)$ for any string w of length n in at most n^k steps. We say that a problem P is in the class $\mathcal{P}$-TIME if there is a constant k and an algorithm $\mathcal{A}$ for P which takes at most n^k steps for any input of size n. A language L is in the class $\mathcal{P}$-TIME if the membership problem for L is in the class $\mathcal{P}$-TIME.*

Size of an input is defined to be the number of bits needed to express the input. For example, the primality problem is to decide whether a given numeral represents a prime number. The size of the input is not the number, but the number of bits in the numeral for the number. If N is an integer and $\langle N \rangle$ is the numeral for N is base b, the size of $\langle N \rangle$ is $\Theta(\log N)$ if $b \geq 2$.

A 0/1 problem is a problem such that the answer is either 0 or 1 (false or true) for each instance. For example, the membership problem for a language L is a 0/1 problem. In fact, every 0/1 problem is the membership problem for some language.

We say that L is $\mathcal{NP}$-TIME if there is an NTM (non-deterministic Turing Machine) which accepts L in polynomial time. (We could simply say non-deterministic machine ... it doesn't have to be a Turing Machine.)

$\mathcal{P}$-TIME and $\mathcal{NP}$-TIME are usually abbreviated as $\mathcal{P}$ and $\mathcal{NP}$, respectively.

Theorem 2 *A language L is $\mathcal{NP}$-TIME if and only if, given any string $w \in L$, it can be proven that $w \in L$ in polynomial time.*

That is, there is a constant k such that, for any $w \in L$ of length n, there is a proof that $w \in L$ whose length (number of symbols in the proof) is at most n^k.

Trivially, every $\mathcal{P}$-TIME language is also $\mathcal{NP}$-time. The converse is perhaps the most important open problem in all of computation theory, and perhaps the most important unsolved problem in all of mathematics.

Conjecture 1 *If L is an $\mathcal{NP}$-TIME language, then L is $\mathcal{P}$-TIME.*

All (as far as I know) experts are of the opinion that Conjecture 1 is false. The usual statement of this conjecture is, "$\mathcal{P} = \mathcal{NP}$."

Definition 3 *A language L is co-$\mathcal{NP}$ if its complement is $\mathcal{NP}$.*

Definition 4 *A language L (or equivalently, a 0/1 problem) is said to be $\mathcal{NP}$-complete if, given any $\mathcal{NP}$-TIME language L_2 there is a polynomial time reduction of L to L_2.*

Go to the internet and look up the definition of SAT, the Boolean Satisfiability problem, as well as the special form called 3-SAT.

Briefly, a boolean expression is satisfiable if it is not a contradiction. For example, "$x = y$ and $x! = y$" is a contradiction, hence not satisfiable, while "$x = y$ and $y = z$" is satisfiable. SAT is the language consisting of all satisfiable Boolean expressions.

We will not give the proof of the following theorem. You can find it on the internet.

Theorem 3 *SAT is $\mathcal{NP}$-complete.*

Once we prove a problem to be $\mathcal{NP}$-complete, we can use reduction to prove other problems $\mathcal{NP}$-complete.

Lemma 1 *If there is a polynomial time reduction of L_1 to L_2, and if L_2 is $\mathcal{NP}$ and L_1 is $\mathcal{NP}$-complete, then L_2 is $\mathcal{NP}$-complete.*

The proof is a trivial, given the rule that there can be no "easy" reduction of a "hard" problem to an "easy" problem.

Theorem 4 *3-SAT is $\mathcal{NP}$-complete.*

Proof: It is trivial that 3-SAT is $\mathcal{NP}$. We can define a polynomial time reduction of SAT to 3-SAT. (We skip that construction: you can find it on the internet, and I might do it in class.) Since SAT is $\mathcal{NP}$-complete so is 3-SAT by Lemma 1. ∎

Well-Known $\mathcal{NP}$-Complete Problems

In the orginal paper on the subject, a number of well-known problems were proved to be $\mathcal{NP}$-complete. Now, there are thousands of $\mathcal{NP}$-complete problems known.

1. Partition. Given any set of weighted objects, does there exist a partition of that set into two subsets of equal weight? For example, can there be a tie in the Electoral College?

2. Knapsack. Given any set of weighted objects, and given a knapsack with capacity K, does there exist a subset of objects that exactly fills the knapsack? That is, a subset whose total weight is exactly K?

3. Traveling Salesman. Given n cities with various distances between them, and given a distance D, can a salesman, starting at one city, visit all of the cities, each exactly once, while traveling a total distance of no more than D?

4. Integer Programming. Linear Programming can be solved in polynomial time, where the variables have real type. But if the variables are required to have integer type, the problem is $\mathcal{NP}$-complete.

5. Bounded Degree Minimum Spanning Tree. Given a weighted graph G, and given a weight W, can you find a spanning tree of weight at most W? Kruskal's algorithm solves this problem in polynomial time. But if we impose the condition that the spanning tree can have degree at most D, the problem is $\mathcal{NP}$-complete.

6. Independent Set. A set I of vertices of a graph G is *independent* if no two vertices of that set are neighbors. Given a graph G and a number k, does G have an independent set of size k?

Guide Strings

Let M be some non-deterministic machine. Any computation of M requires picking one of finitely many choices at each step. Without loss of generality, M never has more than two choices at each step, since k choices at a step can be emulated by a sequence of at most $\log_2 k$ steps with 2 choices at each step. We can deterministically emulate any finite computation of M by providing a binary string, called a *guide string*, of sufficient length. At each step, the emulation reads the guide string to determine the next choice. The emulation halts when the end of the guide string is reached.

Theorem 5 *If a language L is accepted by some non-deterministic machine M_1, then L is accepted by some deterministic machine M_2.*

Proof: Let $\Sigma = \{0, 1\}$, the binary alphabet, and let $g_1, g_2 \ldots$ be a canonical order enumeration of Σ^*. Let M_2 be the following program.

1. Read w.
2. For $i = 1$ to ∞:
   (a) Emulate M_1 with input w using g_i as a guide string.
   (b) If that emulation outputs "1" before the guide string is exhausted, write "1" and **break**.

If w is accepted by some computation of M_1, let g be a binary string which encodes the necessary choices of that computation. When $g_i = g$ in the main loop of the code, M_2 halts and accepts w. On the other hand, M_2 will never halt if w is not accepted by any computation of M_1. ∎

We define $\mathcal{EXP}$ to be the set of all functions $f : \mathcal{N} \to \mathcal{N}$ such that $f(n) = O\left(2^{n^k}\right)$ for some constant k. We define $\mathcal{EXP}$-TIME to be the class of all languages which are accepted in *exponential time*. That is, $L \in \mathcal{EXP}$-TIME if there is a deterministic machine M and a constant k such that for every $w \in L$, M accepts w in at $\left(2^{n^k}\right)$ steps where n is the length of w, and M does not accept any string not in L.

Theorem 6 $\mathcal{NP}$-TIME $\subseteq \mathcal{EXP}$-TIME.

Proof: Suppose $L \in \mathcal{NP}$-TIME. Let M be an NTM that accepts L in polynomial time, *i.e.,* M does not accept any string not in L, and there is a constant k such that every $w \in L$ is accepted by M in at most n^k steps. For each $w \in L$, let g_w be the guide string which encodes the choices that M makes while accepting w. The length of g_w does not exceed the number of steps M needs to accept w. During the loop of M_2 given in the proof of Theorem 5, we can halt M_2, and output "0" once $g_i > g_w$ in the canonical order. There are at most 2^k guide strings that are less than g_w in canonical order, hence the program, which is deterministic, decides L in exponential time. ∎

Witnesses, Certificates

If we have an instance I of a problem P, we say that a string w is a *witness*, or *certificate* for I if it shows that I is a solution of the problem.

For example, if $I = (K, x_1, x_2, \ldots x_m)$ is an instance of the Knapsack problem, a witness for I is any subsequence of $\{x_i\}$ whose sum is K.

Theorem 7 *If $L \in \mathcal{NP}$, each member of L has a witness of polynomial length.*

More formally, if $L \in \mathcal{NP}$, there is a deterministic machine V, called the *verifier*, and an integer k such that

1. The input of V is an ordered pair of strings (u, v). With input (u, v), V either *accepts* or *rejects.*
2. If $u \notin L$ and v is any string, V rejects (u, v).
3. If $u \in L$ and $n = |u|$, there is some string v such that
   (a) $|v| \leq n^k$,
   (b) V accepts (u, v) in at most n^k steps.

Note that V could reject (u, v) even if $u \in L$. The verifier is easy, but finding the correct certificate could be hard.

Space Complexity

$\mathcal{P}$-SPACE is the class of all lenguages L which are accepted in polynomial space. That is, $L \in \mathcal{P}$-SPACE if there is some integer k and some machine M which has at most n^k states which accepts L.

We can also define the non-deterministic version, but we don't get a new class, since $\mathcal{NP}$-SPACE $= \mathcal{P}$-SPACE.

The proof of the following theorem is pretty easy. Do you see it?

Theorem 8 $\mathcal{NP}$-TIME $\subseteq \mathcal{P}$-SPACE.

The following conjecture is also considered very important, and experts also believe that it is false.

Conjecture 2 $\mathcal{NP}$-TIME $= \mathcal{P}$-SPACE.