Lecture 8: P,NP, and NP-Completeness

We began our discussion of P,NP, and NP-completeness. Informally a problem is in P if there is an algorithm which solves it in a number of steps which is polynomial in the input size. In order to make this precise we need to define formally what we mean by a problem and the size of an input, and then specify our model of computation and the definition of a simple step. We remarked that while the precise definition of the model of computation would affect the running time of algorithms, for any two reasonable models the running time in one would be polynomial if and only if the running time in the other was polynomial.

We can finesse the question of the size of the input by restricting our attention to decision problems for languages. Thus, the input is some word w in Σ^* for some finite alphabet Σ, and we are asked to determine if w is in a language over Σ(i.e. a subset of Σ^*). Thus, for example, L might be all encodings of a pair (G,k) such that G is a graph containing a clique of size k.

You have seen/will see in 330, one model of computation, the Turing machine, in detail . A Turing machine $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ has a state space Q, a tape alphabet Γ which contains an input alphabet Σ and a blank symbol ϕ which is not part of the input alphabet, a 1 way infinite tape each square of which contains a symbol of Γ and a combined CPU/IO device which has a read-write head which at any given time can read/write into exactly one of the squares of the tape. The CPU has three distinguished states, the start state q_0, and two halting states q_{accept} and q_{reject}.
The actions of the machine are controlled by its transition function δ which maps $Q\text{-}q_{accept}\text{-}q_{reject}$ X Γ to a subset of Q X Γ X(R,L). In a deterministic Turing Machine (DTM), these subsets all have size 1, in a Non-Deterministic Turing Machine (NDTM) they can have arbitrary size. In each step a DTM reads the symbol α in the tape square where it currently rests, and applies δ to this symbol and its current state q to determine the new state it enters, the new symbol to be written on the current square and whether it moves Right or Left. A NDTM can choose to use any of the elements of $\delta(\alpha,q)$ to determine its new state, the tape symbol it writes and the direction it moves.

For any word w in Σ^*, the computation performed by a Turing Machine on input w is the simple steps it carries out if it starts with w on the rightmost ||w|| tape symbols and blanks on the rest of the tape, with the CPU on the rightmost state square in q_0. For a DTM this is a sequence of simple steps. For an NDTM, this is a tree of possible computations.

For a language L over Σ, a DTM *decides* L if for any input w in L, the computation on input w ends in q_{accept} while for any input w not in L, the computation ends in q_{reject}. An NDTM *decides* L if for any w in Σ^*, the computation tree on input w is finite, and there is a leaf of this tree labeled q_{accept} precisely if w is in L (thus for w not in L, every leaf is labeled q_{reject} and for w in L it may be the case that all but one of the leaves are labeled q_{reject}).

A language L is in P, if for some a and b there is a DTM which (i) decides L and (ii) halts after at most $a||w||^b$ steps on every input w.

A language L is in NP, if for some a and b there is an NDTM which (i) decides L and (ii) has a computation tree of height at most $a||w||^b$ on every input w.

These definitions extend naturally to decision problems, i.e. problems which have a yes-no answer. The decision problem is in P or NP precisely if the languages "reasonably encoding" yes instances of the problem are. We noted that it seems likely that any reasonable encoding will have size which is at most a polynomial in the size of any other reasonable encoding, so informally, we can speak of NP and P as families of decision problems independent of the encoding.

We remarked that some people used a different notion of non-determinism which involved having a DTM satisfying the following properties (i) for each word w in L there is a certificate w* such that on input w#w* the DTM halts in q_{accept} while (ii) for every word w not in L, on any input w#w*, the DTM halts in q_{reject}. In this setting, a language L is in NP, if for some a and b there is a such a DTM which halts after at most $a||w||^b$ steps on every input w. We saw that the two definitions were equivalent.

Thus a problem is in NP precisely if there is a certificate which allows us to check that a yes answer is correct in polynomial time. A decision problem is in co-NP if there is a certificate which allows us to check that a no answer is correct in polynomial time. Clearly P is contained in both NP and co-NP. It is an open conjecture whether P is exactly the intersection of NP and co-NP. This would be true if P=NP. However,we do not know if this is true. Indeed P=NP? is a million dollar question (see http://en.wikipedia.org/wiki/P_versus_NP_problem).

We say that L is polynomially reducible to L', written $L <_P L'$ if for some a and b there is a DTM such that for any input w: (i) the machine halts after $a||w||^b$ steps, and (ii) the string written when the DTM halts consists of a word f(w) using only non-blanks followed by an infinite string of blanks, and (iii) f(w) is in L' precisely if w is in L.

A language L is NP-hard if every language in NP is polynomial reducible to L. A language L is NP-complete if (i) it is in NP, and (ii) it is NP-hard.

We remarked that some people used a different definition of polynomial reduction.

We observed that Cook and Levin had independently proved that the SAT decision problem is NP-complete (we will prove this in the next lecture). We then noted that since the "is polynomially reducible to" relation is transitive, to prove a problem π is NP-complete it is enough to show that (i) π is in NP, and (ii) for some NP-complete problem π' we have $\pi' <_P \pi$. We will adopt this approach to show that many problems are NP-complete. In doing so, we will not describe the reduction algorithm using a Turing machine but rather simply informally describe and analyze these algorithms.

I asked you to think about how to reduce SAT to the decision problem CLIQUE whose input is a graph G and integer k, and we ask: does G contain a clique of size k? Since CLIQUE is clearly in NP (given as a certificate the names of k vertices in a clique we need only ensure that this list really does have k elements and that there is an edge between every two of them- this can be done in O(|E|) time), this implies that clique is NP-complete. I also asked you to think about proving that the Decision Problem VERTEX COVER is NP-complete. An instance of this problem is a graph G and an integer k, we are asked to determine if there is a cover using at most k vertices.

Everything you need to know about Turing Machines and the formal definition of NP-completeness is given above. It meshes with what you have seen/will see in COMP330. The text for that course is Sipser's Theory of Computation, the first edition of which is readily available on the web. If you wish, you can also look at that text but this is not necessary. Pages 125-135 of that text discuss the definition of Turing machines, and provide a few examples. Sections 7.2 and 7.3 and 7.4 of the text discuss P, NP, and NP-completeness. That the two definitions of NP are equivalent is Theorem 7.17 Reducibility is discussed on pages 191 as mapping reducibility using the definition of computable function from 190. The definition of polynomial reducibility is given on page 250. The proof of the Cook-Levin Theorem given next lecture is not from Sipser, it is a much better proof from a book by Garey and Johnson. A scan of the proof is on the mycourses website.