Lecture 9: 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 simple 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.

For our formal definitions, we focus on languages. 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 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 a distinguished start state q_0, and two distinguished halting states q_{accept} and q_{reject}. The actions of the machine are controlled by its transition function δ which maps $Q\text{-}q_{accept}\text{-}q_{reject} \times \Gamma$ to a subset of $Q \times \Gamma \times (R,L)$. 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 $w \in L$, every leaf is labeled q_{reject} and for $w \in L$ it may be 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 language encoding yes instances of the problem is. We noted that while the size of the encoding of a decision problem would very depending on the encoding chosen, for any two reasonable encodings the size of the larger encoding would be a polynomial function of the size of the smaller encoding so that essentially whether or not a decision problem is in P or NP is 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}.

Thus a decision problem is in NP if there is a polynomially verifiable certificate that the answer for an instance is yes. A decision problem is in co-NP if there is such a certificate that the answer is no. Clearly P is contained in NP and co-NP and hence NP∩co-NP. It is conjectured that P is equal to NP∩co-NP. It is a million dollar question whether P=NP, see http://en.wikipedia.org/wiki/P_versus_NP_problem.

My discussion of Turing Machines and the definition of NP-completeness follows what you have seen in COMP330 (my understanding is you have all taken that course). The text for COMP330 is Sipser's Theory of Computation which discusses Turing machines and introduces the notions of P and NP. Webpages from the 2012 version of that course are at http://crypto.cs.mcgill.ca/~crepeau/COMP330/. The slides introducing Turing Machines are those for Lectures 14 of that course, you may find them useful.