COMPSCI 330: Design and Analysis of Algorithms **Apr 13, 2020**

Lecture 21: P vs NP, Reductions

Lecturer: Rong Ge *Scribe: Haoming Li*

21.1 Reductions

Given two problems A and B, how do we know which is more difficult? *Reduction* provides a general way of comparing the difficulty of two problems. We say that problem A *can be reduced to* (or *is reducible to*) problem B if an algorithm that solves B can also be used as a subroutine to solve A.

For example, the problem of finding the median of an array (MEDIAN) can be reduced to sorting an array (SORTED). We can use an algorithm that solves SORTED as a subroutine to solve MEDIAN. Given an input array X to MEDIAN, we first sort it to array Y using the algorithm for SORTED. The middle element of Y is thus the median of X.

For example, the problem of finding the longest increasing subsequence of a sequence (LIS) can be reduced to the problem of finding the longest common subsequence between two sequences (LCS). We can use an algorithm that solves LCS as a subroutine to solve LIS. Given an input sequence X to LIS, we first sort it to sequence Y. We then find the longest common subsequence between X and Y using the algorithm for LCS. The solution to that is the longest increasing subsequence of X.

21.1.1 Poly-time Reductions

All previous examples are, in fact, *polynomial time reductions.* If A can be reduced in poly-time to B, that means we can perform the reduction process (given an input X to A, prepare an input Y to B) in poly-time, with no post-processing involved (i.e. the solution to B(Y) is the solution to A(X)).

21.2 Complexity Classes

In general, we like to think that the problems that can be solved in poly-time (e.g. $O(n), O(n \log n), O(n^3)$) are easy, and that those we believe cannot be solved in poly-time are hard. *Complexity classes* provides a general way to classify problems (into sets, or *classes*) by their difficulty.

21.2.1 P and NP

The class P contains all the decision problems that can be solved in poly-time. For example, "Is there a path from s to t that has cost at most L?" and "Is there a spanning tree with cost at most W?" are both in P. In fact, all the problems (in their decision versions) that we have covered in 330 so far are in P.

The class NP contains all the decision problems whose solution can be verified in poly-time. For example, "does this sudoku have a solution?" is in NP, because given a solution to the sudoku (not just the yes/no decision), we can verify its correctness in poly-time by checking the sum of rows, etc. "Is there a path from s to t that has cost at most L?" is also in NP, because given a solution (a path), we can verify its correctness in poly-time. For the same reason, "can this graph be 3-colored" and "is this number composite?" are both in NP.

21.2.2 P vs NP

By definition of the two classes, all problems in P are also in NP. However, whether P=NP or P⊂NP is a major open problem in computer science.

21.2.3 NP-Completeness

A problem is *NP-complete* if every problem in NP can be reduced to it in poly-time. NP-complete problems are, in other words, the hardest problems in NP (by the reducibility definition). An important corollary follows directly from this definition is that, if any of the NP complete problems can be solved in polynomial time, then P=NP.

The Cook-Levin Theorem says that Circuit-SAT is an NP-complete problem. We can show that a problem in NP is NP-complete by reducing Circuit-SAT (or any other known NP-complete problems) to it in poly-time.

21.2.4 Other Complexity Classes

There are many other complexity class. If you are interested, see Wikipedia and Complexity Zoo.