Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 12

Lecture 57

Welcome to the 57th lecture of second level algorithm course. We have been studying the theory of NP completeness. In this lecture, we will see a lots of examples of NP complete reductions, thereby proving that many problems are NP complete. So, let us begin. So, if we recall the definition of NP completeness, it has two components one is membership in NP and NP hardness Typically membership in NP is easy because all we need to show is the existence of a polynomial size certificate and a polynomial time algorithm. However, if we look at the definition of NP hardness, it says that every problem in NP should be polynomial time many to one reducible to the problem that we want to show to be NP hard, which sounds like a daunting task. So, the idea is to prove one problem to be NP-hard from the definition and to show some other problem to be NP-hard, it is enough to show reduction from a NP-hard problem.

So, this is a crucial observation. if a decision problem is a decision problem let us give it a name π is NP hard and π many to one polynomial time reduces to another decision problem π_1, then π_1 is also NP hard ok. This follows from the simple observation that if our decision problem says π_2 many-to-one polynomial time reduces to π, and π many-to-one polynomial time reduces to π_1. These two together imply that π_2 many-to-one polynomial time reduces to π_1. So, the daunting task is only proving the first NP-hard problem, and that is done by the celebrated Cook-Levin theorem. They showed that a problem called circuit satisfiability is NP-complete. They indeed showed that every problem in NP polynomial time many-to-one reduces to the circuit satisfiability problem. Although the proof is not difficult, because the ideas involved in the proof will not be used in our course, we will not prove this theorem. You can see the proof of this theorem in many standard textbooks on algorithms, for example, the Introduction to Algorithms book by CLRS. So, we will assume this theorem that circuit satisfiability is NP-complete, and from this theorem, we will reduce circuit satisfiability to various other problems to prove some other problems are also NP-hard. If they belong to NP, then they are also NP-complete. But before that, let us understand the formal definition of circuit satisfiability.

So, in this case, we are given the input is a Boolean circuit with all kinds of Boolean gates like AND, OR, NOT, and these three together form a universal gate because any Boolean formula can be written down with these three operations. So, a Boolean circuit with these operations So, let's call this circuit φ. Pictorially, how does it look like?

For example, the leaves are variable input variables. Suppose this is an AND gate. This is the NOT gate, and this is the output. So, this is how a typical Boolean circuit looks like, and the question is: does there exist an assignment to the Boolean variables such that the Boolean circuit evaluates to true. So, for this particular example, if we set x_4 to be, say, false, irrespective of how we set other variables, the output is true. So, this is the circuit satisfiability formula, the circuit satisfiability problem. Our next problem is formula satisfiability. In this case, the input is a Boolean formula say, if $x_1, \ldots, x_n$, and the question is: does there exist a satisfying assignment for $f(x_1, \ldots, x_n)$, ok. So, we will show that formula satisfiability is NP-complete. Let us recall NP-complete needs two things: one is membership in NP, and the other is NP-hardness.

So, this formula satisfies The satisfiability problem clearly belongs to the class NP because the satisfying assignment can act as a certificate. OK. So, to prove NP-hardness of formula satisfiability, we exhibit a polynomial-time many-to-one reduction from circuit satisfiability. So, let's understand this reduction with an example. So, what is the reduction? Let's recall. So, we want to show that circuit satisfiability polynomial-time many-to-one reduces to formula satisfiability. That means we need to design a polynomial-time algorithm that takes an instance of circuit satisfiability and outputs an equivalent instance of formula satisfiability. So, let us understand this with an example. Let us take an arbitrary instance of circuit satisfiability. Let's understand this algorithm with a toy example. So, this is how an arbitrary instance of circuit satisfiability looks like.

What we do first is label the output of every gate with a variable. OK, because there are polynomially many input gates or polynomially many gates in the circuit satisfiability instance, the number of variables introduced is at most the number of gates, or it is exactly the number of gates in circuit satisfiability, which is polynomial in the input size. Now, once we label this, we can write the equivalent I look at the output gate, the final output gate of the circuit, which is Y_5. Now, for this to be satisfiable, Y_5 should be satisfiable, and Y_5 should be the AND of Y_4 and X_4. So, Y_5 should be the same as the

AND of X_4 and Y_4, and Y_4 should be the NOT of Y_3, and this should be Y_4. Y_3 should be the OR of X_3 or Y_2, and Y_2 should be the NOT of Y_1, and Y_1 should be the NOT of X_1. Not, not, not—Y_1 should be the AND of X_1 and X_2, OK.

Now, if we insist that we should only use AND, OR, and NOT operators, then we can replace this 'equal to' or 'if and only if' with this formula: A if and only if B is 'A implies B' and 'B implies A,' and 'A implies B' means if A is true, then B also must be true, which is the same as saying NOT A OR B. That means if A is true, B must be true; if A is false, B can be anything, and NOT B OR A. So, we can replace each of these 'if and only if' statements with this part, and hence we get a formula. Let us call this formula F. equal to this, and clearly, F is satisfiable if and only if the Boolean circuit phi is satisfiable.

Okay? Indeed, if there is a satisfying assignment of ϕ, we can feed that satisfying assignment and we can get evaluate this y_1, y_2, y_3, y_4 and y_5 which will satisfy the formula On the other hand, if there is a satisfying assignment for if then the corresponding assignment for x_i's satisfies the Boolean circuit ϕ ok. Not only that if we look at each the formula f is an and of many clauses and each clause is an y_i if and only if something which is also an and of ors so observe that if is and of various clauses $C_1, \ldots, C_m$ where C_j is a logical or of literals when the formula is in this format this is called conjunctive normal form and when the formula should be given in conjunctive normal form the corresponding satisfiability problem is called CNF satisfiability so what we get as a corollary is that

CNF satisfiability is NP complete okay the same reduction what is CNF satisfiability problem the input is a Boolean formula in conjunctive normal form what is conjunctive normal form It is logical and of some number of clauses where each clause is a logical or or of literals. So, the input formula is not in arbitrary form, but it is in conjunctive normal form. And the question is, does there exist a satisfying assignment or not? This problem we have shown is also NP complete.

So, in the next class, we will see some more problems that are NP-complete. Okay. So, let us stop here. Thank you.