Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 11

Lecture 55

Welcome to the 55th lecture of the second-level algorithms course. Till now, we have seen various algorithms for very important problems. From this class, we start the intractability results—how to show that for some problems, it is unlikely to admit polynomial-time algorithms, ok? So, let us begin. So, this theory is called NP-completeness, and in this theory or in this framework, we assume that the problems are decision problems, that is, the output is either yes or no. The instances whose output is yes are called yes instances. Similarly, the instances whose output is no are called no instances. We will see later that this assumption—that the problems are only decision problems—is almost without loss of generality, ok. Now, we will categorize the problems into various classes. The first class is P; it is the set of problems which admit polynomial-time algorithms, ok.

So, what are some examples of the problems that admit polynomial-time algorithms? All the problems we have seen so far, for example, max flow, min cut, maximum matching. So, all these problems belong to the complexity class P. Our next complexity class is NP. NP stands for nondeterministic polynomial.

Formally, a decision problem belongs to NP if there exists what is called a nondeterministic Turing machine. that solves the problem. Because we have not seen nondeterministic Turing machines, we will use an equivalent definition, which is as follows. Equivalently, a decision problem belongs to NP if there exists a polynomial-time algorithm that can verify all yes instances given the input instance and a polynomial-size string, which is called a certificate in this context. So, let us see some examples. For example, the max-cut problem belongs to NP. What is the max-cut problem?

In the decision version of the max cut problem The input are the inputs are a graph G and an integer k. And the question is: does there exist a cut in G of size at least k? In this case, a cut of size at least k is a certificate.

So, given a cut of size at least k, we can check whether the size of the cut is indeed at least k or not, and if it is at least k, then we can we are convinced that the input instance is a yes instance. So, for this reason, for the max cut problem, there is a certificate of polynomial size, namely a cut of size k, which establishes the fact that the input instance is a yes instance. Observe that NP demands a certificate only for yes instances, existence of polynomial size certificate for every yes instance. that can be verified in polynomial time. For no instances, there is no condition. that needs to be satisfied, ok. Some other examples of problems in NP are satisfiability where the input is a Boolean formula with n variables and the question is does there exist an assignment of the Boolean variables which satisfy the formula. This satisfiability problem is also belongs to NP. In the The input is a Boolean formula. and the question is does there exist an assignment of the Boolean variables that evaluate the formula. to true . Here also for every ES instance a satisfying assignment can be used as a certificate. that can be verified in polynomial time.

How? Given an assignment you simply evaluate the formula at that assignment and check whether the ah formula turns out to be true or not. Hence the satisfiability problem also belongs to NP. Here is a easy observation every problem in P belongs to NP in particular P is a subset of NP. Why? Because the certificate could be empty, and in polynomial time, you can solve the problem in P and check whether it is a yes instance or not. Because Every problem in P can be solved, and thus every yes instance of the problem can be verified with an empty certificate. In polynomial time. So, pictorially, here is P and here is NP. A long-standing open question in computer science is whether P is a strict subset of NP or not. So, we will assume that P is not equal to NP. This is called the P versus NP conjecture. So, this is one of the most challenging long-standing open questions in computer science. Now, we define two important concepts. The first one is what is called a polynomial-time reduction or reduction first. So, for reduction, let us first fix two decision problems: let π_1 and π_2 be two decision problems.

We say that the problem π_1 many-to-one reduces to the problem π_2 if there exists a polynomial-time algorithm that takes as input, say I_1 of the problem π_1 and outputs an equivalent input instance I_2. Here, also let us call it an instance of the problem π_2. That is the meaning of equivalence: I_1 is a yes-instance of the problem π_1 if and only if I_2 is a

yes-instance of the problem π_2. We denote the polynomial-time reduction from π_1 to π_2 as this notation, OK?

Now, with the help of reduction, we define an important complexity class called NP-hard. A decision problem is called NP-hard. If every problem So, let us give a name to the first decision problem; let us call it π. A decision problem π is called NP-hard if every problem π_1 in NP polynomial-time many-to-1 reduces to π_2. You observe that if a decision problem π_1 many-to-1 reduces to another decision problem π_2, then we can say that π_2 is at least as hard as π_1. Indeed, if there exists a polynomial-time algorithm for π_2, then using the reduction algorithm, which takes polynomial time, we can design a polynomial-time algorithm for π_1 also. So, in that sense, the NP-hard problems are the hardest problems of the class NP. The other class is NP-complete.

This is also a very important class. A decision problem π is called NP-complete if π belongs to NP and π is NP-hard. So, an NP-hard problem need not belong to NP, but if there is a problem in NP which is also NP-hard, then that problem is among the hardest problems of the class NP, and such problems are called NP-complete problems.

We will see a lot of NP-complete problems in the coming lectures, okay? So, let us stop here. Thank you.