**Second Level Algorithms**

**Prof. Palash Dey**

**Department of Computer Science and Engineering**

**Indian Institute of Technology, Kharagpur**

**Week – 12**

**Lecture 60**

Welcome to the last lecture of the second-level algorithm course. In this lecture, we will also see NP-hardness and NP-completeness of some very important problems. So, let us begin. The first theorem that we will prove today is that the vertex cover problem is NP-complete.

The vertex cover problem clearly belongs to NP because a vertex cover of size at most k or exactly k can be verified in polynomial time for every YES instance. To prove NP-hardness, we reduce from the independent set problem. Let $I_1$ equal to $(G, k)$ be an arbitrary instance of vertex cover.

We claim that the instance $I_2$ equal to $(G, n-k)$, where n is the number of vertices in G. The instance $I_2$ of independent set—sorry, this is an instance we have to take as an arbitrary instance of independent set, not vertex cover. So, $I_1$ is an instance of independent set, and $I_2$ is an instance of vertex cover, OK. We claim that $I_1$ and $I_2$ are equivalent.

So, let us prove it. Let $I_1$ be an independent set $I_1$, YES instance of independent set. That means there exists an independent set of size k. So, S is a subset of vertices of size k, and S is an independent set of G. So, this is $I_1$—it is a yes instance. We claim that V minus S, let us call it W, is of G. Indeed, this must be the case. If there exists an edge E not covered by W, then both the endpoints of E belong to S, contradicting our assumption that S is an independent set, OK. Hence, $I_2$ must be an YES instance of vertex cover, OK. Similarly, show this—I let me give it as homework—that if $I_2$ is then $I_1$ is also an YES instance, OK. This concludes the NP-hardness proof of the vertex cover. So, we have seen some graph-theoretic problems, ah, which are NP-complete; we have seen satisfiability problems which are NP-complete.

Now, let us see a problem of different flavor involving numbers which are NP complete. So, that problem is called subset sum. what is input? The first one is set of n integers n positive integers let us call them $x_1$ to and a target sum B what is the question does there exist a subset of the input integers whose numbers sum up to B. So, we will now show that the subset sum problem is NP completed. The subset sum problem clearly belongs to the complexity class NP because the subset whose sum is b can be verified in polynomial time for YES instances. To prove NP hardness of subset sum.

we reduce an arbitrary instance of 3SAT to subset sum ok. So, let arbitrary instance of 3 set to an equivalent instance of subset So, let $I_1$ equal to $C_1$ and $C_2$ and $C_m$ be an arbitrary instances of 3SAT.

We construct an instance $I_2$ of subset sum as follows. As usual, let us understand this with an example. So, suppose $I_1$ is. So, let us go to the next page.

$x_1$ or $x_2$ or $x_3$ So, any three-set clause has at most 3 literals. So, it can be 2 or 1 also. What do we do? We will create big integers, which will be the set for the subset sum problem, and those integers are indexed by the variables and clauses.

So, we will have n plus m digit integers in $I_2$. So, the digits will be indexed by the n variables. In this toy example, we have 5 variables, and the rest m digits will be indexed by the clauses. We have 4 clauses.

$C_1, C_2, C_3, C_4$, OK. So, first, for each variable, we will have two numbers. So, for $x_1$, I have two numbers. Both of them are 1. Let us call them $v_1$ and $v_1^{'}$. Both of them are 1 at the $x_1$-th digit and 0 in the rest of the digits indexed by variables. For the first number, $v_1$, we will see which clauses are satisfied if $x_1$ is set to true. So, if $x_1$ is set to true, then $C_1$ is satisfied. So, for $C_1$, I will write 1. $C_2$—setting $x_1$ to true does not necessarily satisfy C2. So, it is 0, 0, 0. Similarly, for $v_1^{'}$, I will look at the clauses which will be satisfied by setting x1 to false, which is $C_2$.

That clause—the corresponding digit—I will make it 1. This way, I will have two numbers for each variable. Setting $x_2$ to 1 satisfies clause 1 and clause 3. Setting $x_2$ to 0 satisfies clause 2, and so on. So, this way, I have 2n integers, and now we will have one number for each clause or two numbers each.

So, for clause 1, let us have $s_1, s_1^{'}$, which will have 0 everywhere, but $s_1$ has 1 in the digit corresponding to $C_1$. And $s_2$ has 2 in the digit corresponding to $C_1$ and 0 everywhere else. So, we have 2m integers, OK? And what is our target? Target B, so let us have that index

in general: $x_1,\ldots,x_n$ and $C_1,\ldots,C_m$. This is the indexing, and the target B has 1 in every digit corresponding to variables $x_1,\ldots,x_n$ and 4 in every digit corresponding to clauses $C_1,\ldots,C_m$. Now, we claim that we have a total of $2n+2m$ integers, each integer is of— the number of digits in each integer is at most $n+m$. So, the size of the instance $I_2$ is polynomial in the size of instance $I_1$. We define $I_2$ as this whole set—let us call it X— this X and B. So, $I_2$ is our reduced and the size of $I_2$, which is the sum of the number of digits in every integer in $I_2$ times some constant because the size of the instance is the number of bits needed to write down the input, which is bounded by or is $O(n+m)$ — these are the number of digits of every number, and the number of numbers in the set is $2n+2m$, and there is one number B. So, this times $2n+2m+1$, OK? So, the size of $I_2$ is polynomially bounded by the size of $I_1$. So, these are polynomial-time reductions And the equivalence—I give you this as homework. Let me give you the idea.

If there is a satisfying assignment for $I_1$, you see which variables are set to true. For each variable, say $x_1$, if $x_1$ is set to true, then I will pick $v_1$ in my set. If $x_1$ is set to false, I will pick $v_1^{'}$ in my set. So, this is how I pick the integers among the first 2 n integers. Now, because it is a satisfying assignment for every clause at every number we have picked has at least 1 in that clause.

So, if we just add this n numbers corresponding to the n variables we have picked then they the digits corresponding to the variables will have 1, but the digits corresponding to the clauses will either have 1 or 2 in every clause we will have either 1 or 2 as a sum 1 or 2 or 3 because it has 3 literals ok. So, now, but the that is why we have set the target sum to be 4 for the clauses. So, then for any clause if the sum is 1 then I pick both $s_1$ and $s_1^{'}$ to make it 4, if the sum is 2 then I pick only $s_1^{'}$, if the sum is 3 if I will pick only $s_1$ ok.

So, by choosing $s_1$ and $s_1^{'}$ accordingly we will ensure that the sum becomes 1 to 4 and see the sum of every column does not exceed 9. The sum of every column corresponding or every digit corresponding to variables is exactly 2 and sum of the digits corresponding to any clause it has 3 ones among first 2m integers. So, that 3 and then 3 from below corresponding to 2m integers. So, the sum of the digits corresponding to the clauses are at most 4, okay? not 4, 6, 3 plus 3, 6. So, an important point is even if we add all integers, in X, there is no carry. This is very important to argue each column separately and, using this, you prove the equivalence of the instances as homework. Prove that

I1 is a yes instance of 3SAT if and only if I2 is a yes instance of subset sum, okay? So, this concludes the NP-hardness proof of subset sum, and we have already argued the membership in NP; hence, subset sum is NP-complete, okay? So, let us stop here.

Thank you.