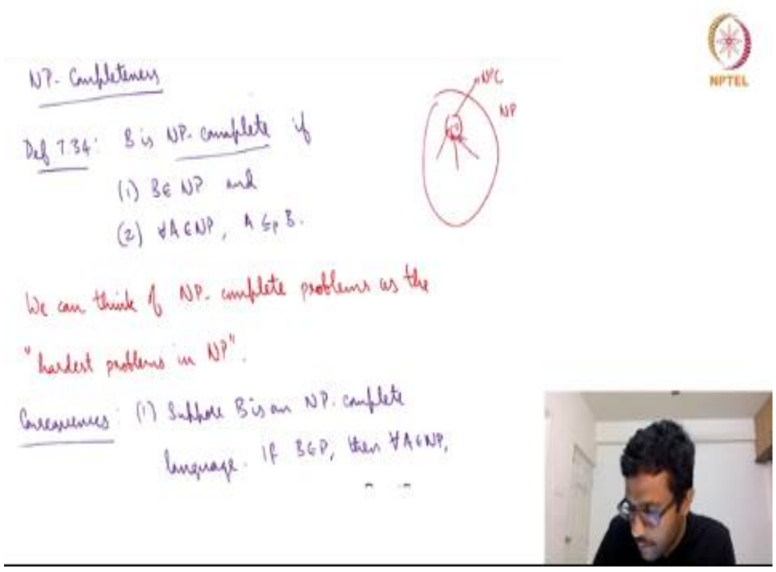
Computational Complexity

Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture -06
NP- Completeness

(Refer Slide Time: 01:43)



NP- Completeness

Def 7.34: B is NP-complete if
(1) $B \in NP$ and
(2) $\forall A \in NP$, $A \leq_p B$.

We can think of NP-complete problems as the "hardest problems in NP".

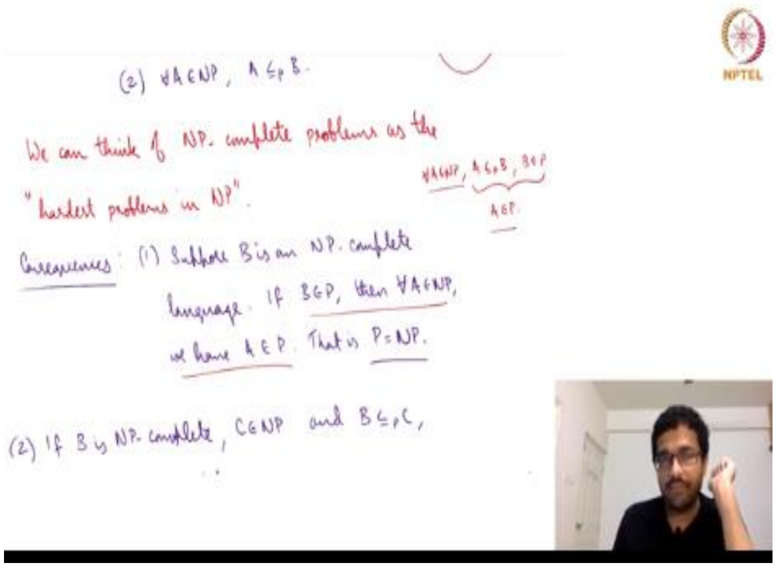
Consequences: (1) Suppose B is an NP-complete language. If $B \in P$, then $\forall A \in NP$,

Welcome to lecture 6 of computation complexity. In fact this will be a very brief lecture, so we have already seen P, NP and what are polynomial time reductions, so in this mini lecture I will just describe what is NP complete or what is the definition of NP complete. The definition, again the numbers are from Sipser, 7.34 a language B is NP complete if first of all B must be in NP and all the a in NP must be reducible to B in polynomial time.

So what does it mean? So all the languages in A should be reducible to B in polynomial time and there are several problems or several languages that are known to be NP complete. The one reason is that you can think of these problems as the hardest problems in NP, so we know that NP contains P and these are the probably, the more easily solvable problems. However NP complete problems are those for which we do not know of a polynomial time algorithm yet.

So we, in a moment we will see why I call them hardest problems in NP, in a way if we solve these problems if you have a solution for these problems then we will have solutions for all the problems in NP In other solutions. I mean efficient polynomial time deciders, so just to pictorially depict so suppose this is NP and this is NP complete. So now if B is in NP complete then not only this B NP complete all the other languages can be reduced to B.

(Refer Slide Time: 02:23)



(2) $\forall A \in NP$, $A \leq_p B$.

We can think of NP-complete problems as the "hardest problems in NP".

$\underbrace{\forall A \in NP, A \leq_p B, B \in P}_{A \in P}$

Consequences: (1) Suppose B is an NP-complete language. If $B \in P$, then $\forall A \in NP$, we have $A \in P$. That is $P = NP$.

(2) If B is NP-complete, $C \in NP$ and $B \leq_p C$,

So the main consequences is that if B is an NP complete language, and if B is shown to have a polynomial time algorithm. B is in P B is shown to be in P then it means that for all A in NP A has a polynomial time algorithm. That means all the languages in NP have polynomial time algorithms that means that NP is equal to P or P is equal to NP. Why is this the case? Because if B is in P, then for all A in NP.

We know that A reduces to B because of the definition of NP completeness, we said we started by saying that B is an NP complete and now we are saying that by assumption B is NP. Now together we saw in the last lecture that these two together imply that A is NP, now this means that for all A in NP, A is in B which means that NP is a subset of P which again is same as saying that P is equal to NP.

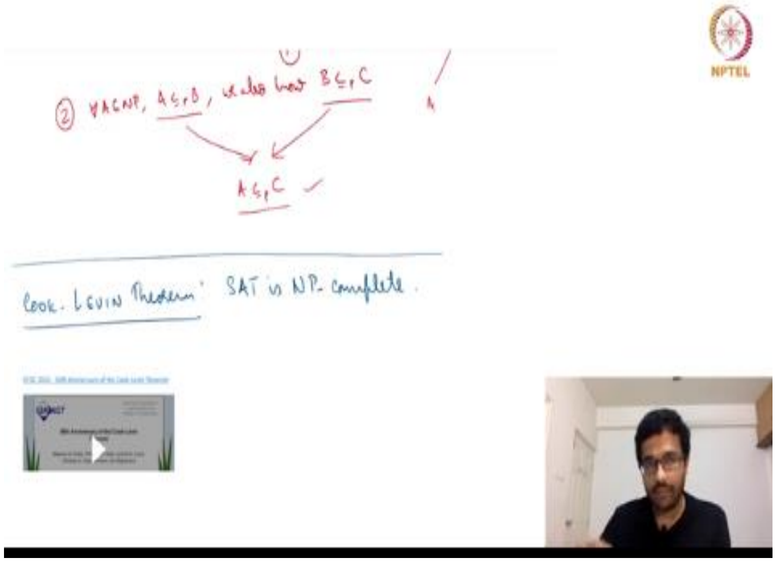
So again, this is an intuition or this is the reason why we call them as the hardest problems in NP, the NP complete languages because if any of these has a polynomial time solution then we have shown that P is equal to NP. There are multiple NP complete problems known so one of which is SAT, 3 SAT, clique, vertex cover independent set, subset sum, integer linear programming many such problems are there.

So, in fact to show that P is equal to NP all you have to do is for any of these problems to demonstrate a polynomial time algorithm. But it has so turned out that even doing that has proved very tricky, nobody has been able to do it so far and to show that P is not equal to NP we have to show that none like we pick some problem and it does not have a polynomial time algorithm.

Some problem in NP or some problem in NP complete does not have a polynomial time algorithm, and showing something does not have a polynomial time algorithm is not easy, because how do you say something cannot be solved in polynomial time, because given a problem you could approach the same problem in many different ways, how do you say that none of these approaches work?

So it is very difficult to show that it is not possible to do something in this much time or polynomial time. This much so this so called lower bound is what they are called, you cannot solve this problem in this much time these kind of statements are very hard to prove, anyway so the point I want to make here is that if B is an NP complete language and if you show that B is in polynomial time then this implies automatically that P is equal to NP.

(Refer Slide Time: 05:44)



② $\forall A \in NP$, $\underline{A \leq_p B}$, we also have $\underline{B \leq_p C}$

$\underline{A \leq_p C}$ ✓

$\underline{\text{Cook-Levin Theorem}}$: SAT is NP-complete.

The next consequence I want to mention is suppose B is a NP complete language and suppose C is another language in NP and suppose we can reduce B to C in polynomial time, then C is also NP complete, B is in NP complete C is an NP and B reduces to C in polynomial time, then C is also NP complete y so we have to show two things one is that so now to show the C is NP complete.

We have to show that C is in NP that is already shown this is given so condition one is satisfied, two is that for all A in NP, A reduces to C, so 0.2 for all A in NP we know that B is an NP complete, so we know that A reduces to B, because B is NP complete but we also know that B reduces to C so we have this and we have this A reduces to B and B reduces to C so together these imply that A reduced to C.

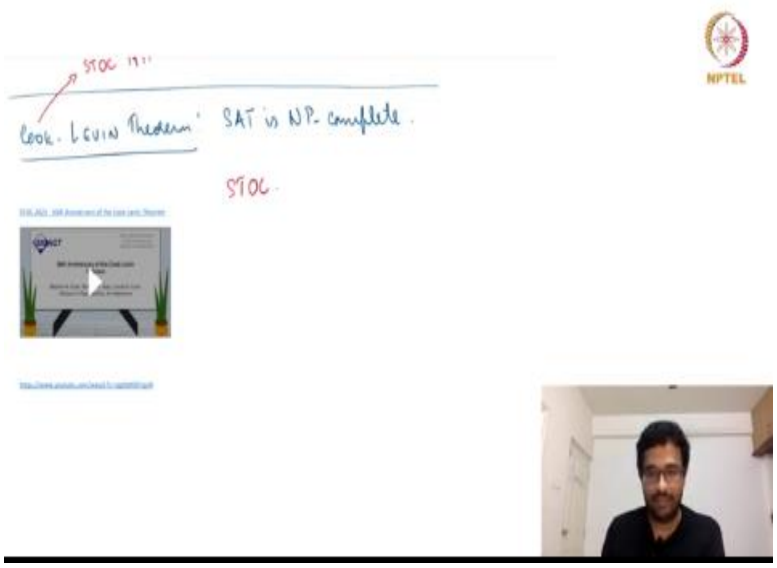
Again this is something that I mentioned in the previous lecture but I did not give a proof, I asked you to think about it so this is something that you can take for granted, now but you can work out the proof so this together means that for all A in NP, A is reducible to C in polynomial time which is a condition 2. So this is also verified so B is NP complete B reduces to C where C is in NP implies that C is also NP complete so it is like this suppose B is in P complete sorry B is NP complete B is reducible to C, B reduces to C.

But we already know that all the other languages in NP reduces to B, so by transitivity of reductions A reduced to B, B reduces to C it follows that A reduces to C, so again the definition of NP completeness is that A language B is NP complete if B is an NP and all the other languages NP are reducible to B and they are called the hardest problems in NP, because if we show that any one of them has a polynomial time algorithm it follows that p is equal to NP.

So if we solve them we have automatically solved all the problems in NP and finally again how do you show that something is NP complete we have to show that first of all we have to show that that language is an NP, second we have to show that all the other languages reduce to this so that does not seem like an easy task, like how do you show that every other language reduces to this, like because there are so many languages in NP.

How do you show that all of them reduce to this so this is what is known as Cook Levin theorem. And this is what really started off the theory of NP completeness, so which is what I mentioned in one of the earlier lectures so they showed that Cook and Levin independently, Cook was in I think Canada and Levin was in Russia, they independently born in Canada one in Russia, they independently showed that SAT is NP complete. SAT or three SAT is an NP complete, so the details of this we will see in the next lecture.

(Refer Slide Time: 09:37)



How it is done so now just a small bit of history I want to mention this was done in 71, so now it is 2021, so we have exactly 50 years from when this was proved to be NP complete that and therefore starting off the area of NP completeness so there is a well-known conference in computer science theory called STOC and recently the 2021 edition of STOC happened which is happened to be the 53rd edition of STOC and Steve Cook's paper appeared in stock 1971.

So again he this was independent so they did not publish together, Steve Cook's paper appeared in 1971, STOC which was a third STOC. So STOC stands for Symposium on Theory of Computation and there he proved that SAT NP is complete and in the 53rd STOC that happened maybe few weeks back, there was a discussion by Steve Cook, Levin then Richard Carr who also did a bunch of early results in the theory of NP completeness with a couple of other computer scientists.

First Steve Cook talks about his experience then Levin talks about his experience then at the end they together discuss and take some questions, so it is like I think one and a half hour video and if you are interested in the history of how this evolved and how maybe some of the technical details maybe not familiar right now but if you can at least watch the interesting history parts.

How obviously Cook and Levin are much older Cook age is 81 years old but it is an interesting watch if you have the time, so I have pasted the Youtube link here perhaps or you could just search for this in Youtube STOC 2021, 50th anniversary of Cook Levin theorm and with that I conclude this mini lecture, thank you.