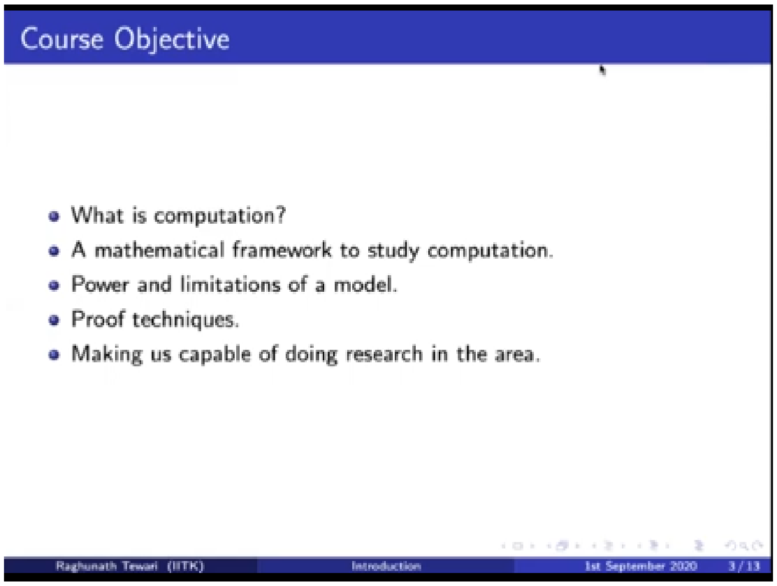
Computational Complexity Theory

Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -01
Introduction

Hello everyone, welcome to this course on Computational Complexity Theory. My name is Raghunath Tewari and I will be your instructor for this course. So, today I am going to talk about broadly two major things. I am going to talk about firstly the course details and then I will move onto the some introductory topics on complexity theory.

(Refer Slide Time: 00:43)



Course Objective

- What is computation?
- A mathematical framework to study computation.
- Power and limitations of a model.
- Proof techniques.
- Making us capable of doing research in the area.

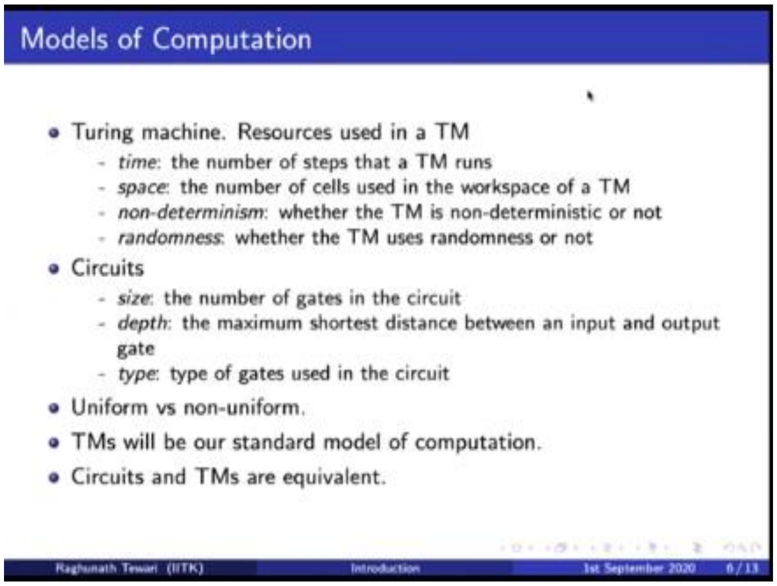
Raghunath Tewari (IITK) Introduction 1st September 2020 3 / 13

So, first of all, let me talk about the course objectives. So we are going to discuss what is Computation? So, speaking in an informal way, computation is basically a step by step process to perform a well defined task. So it can be anything it can be solving a problem or it can be any other object where you can actually give us step by step process to complete a well defined task and in this course, we are going to look at it mathematical framework to study computation.

We are not going to study computation from let us say by looking at a computational device or anything. We will look at computational device, but it will only be a mathematical device and mathematical modeling. And the most important element in this course is the following, we are going to look at the power and limitations of various computation model. So that is actually what computational complexity is where we look at how powerful a given computation model is or what are the limitations of a computational model under various restrictions.

And while learning this we also look at various proof techniques. So and finally, the goal would be by the end of this course that this course should make us capable of picking up a research problem in this area and probably working on it. So that is the; I mean high level objective that we would like to achieve by the end of this course.

(Refer Slide Time: 02:37)



Models of Computation

- Turing machine. Resources used in a TM
  - *time*: the number of steps that a TM runs
  - *space*: the number of cells used in the workspace of a TM
  - *non-determinism*: whether the TM is non-deterministic or not
  - *randomness*: whether the TM uses randomness or not
- Circuits
  - *size*: the number of gates in the circuit
  - *depth*: the maximum shortest distance between an input and output gate
  - *type*: type of gates used in the circuit
- Uniform vs non-uniform.
- TMs will be our standard model of computation.
- Circuits and TMs are equivalent.

Raghunath Tewari (IITK) Introduction 1st September 2020 6 / 13

So now we move to the introduction to complexity theory some of the introductory topics that I plan to discuss today. So in this course, we will be looking at Turing machines as one of our models of computation. So what are the resources that are used in our Turing machine? So in a Turing machine there can be many types of resources that Turing machine uses. For example time is a resource.

What is time? Time is the number of steps that a Turing machine takes to run its input. So given input how many steps are the Turing machine takes. Space can also be resource used by a Turing machine. So space is the number of cells that are used by the Turing machine in its workspace. We can actually divide the space of the Turing Machine into I mean, so there can be three different types of tapes in a Turing Machine the input tape, the work tape and the output tape.

So when we count the space it is only the space that is used in the work, so when we talk about more about space complexity, we will come back to this point. Then the type of the Turing Machine is also a resource, for example, whether it is a non deterministic Turing Machine or not, whether it is a randomized Turing Machine or not. So these are also various types of resources that a Turing Machine has access to.

Another model of computation that we will be discussing in this course is circuits. Turing Machine is more like how an algorithm is run. So, it is more like a stimulation of a computer. A circuit is slightly different. In a circuit, you have gates less such as AND gates, OR gates, NOT gates might have also some other types of gates and also there are some input gates, either can be constants like 0 or 1 or they can be some variables x1, x2 where those variables can take values 1, 0 in Boolean circuits.

Now the resources of a circuit are the following. So it can either be the size of the circuit that is how many gates are present in the circuit. It can be the depth of the circuit that is a maximum shortest distance between an input and output gate in the circuit. It can also be that types of gates that are used in the circuit, for example whether you allow the NOT gates or not, or how do you allow the NOT gates to appear AND gates, OR gates and any types of restriction that makes logical sense to apply on the circuit.

So these are the various types of resources. Now so under these resources we are going to study that for example, let us see if I am looking at Turing Machine, we are going to ask questions like okay if I allow polynomial amount of time for a Turing Machine to run, what are the types of problems that the Turing machine can solve and cannot solve? If I allow it may be let say linear amount of time, what are the kinds of problems that the Turing Machine can solve and cannot solve and linearly for space as well?

Again for circuits also if I ask them if the machine has maybe linear depth what kind of problems can this circuit solve or cannot solve. So these are the kind of very broadly, these are the kind of questions that we would like to address and we are going to learn how we will how I mean learn techniques that are needed or the techniques that are used usually use to address these kind of questions?

So I talked about these two models that is Turing machine and circuits and a very fundamental difference between these two models is what is called a uniform versus a non uniform model. So Turing Machine is what is called a uniform model of computation and a circuit is what is called a non uniform model of computation. Let me see a little bit about this. So a uniform model of computation is where you have a single machine or device whatever you want to call it for all input lens.

For example, I just forget about Turing Machine and circuits for the time being, when you write an algorithm to solve your problems if you write an algorithm to sort an array of numbers. You do not write separate algorithms that will take care of arrays of separate sizes. You write one algorithm and no matter what array is given to you, array of whatever size is given to you the same algorithm actually works on all array sizes.

That is what is called a uniform model, that is what is called I mean a uniform model to solve a problem. So Turing Machine is a uniform model, so there is only a single Turing Machine, if you have a Turing Machine that solves a particular problem, the same Turing Machine will actually work on all input length. Now circuits are actually not uniform. In other words, if you have a circuit, a circuit only has a fixed number of variables that it can take as input.

So once you define a circuit, it only has a fixed let us say an n number of variables, which it can take as input. Now if you have an input whose length is maybe n + 1 or n + 2, you cannot actually feed it to the same circuit. You need actually different circuit for that input. So what we do, the usual way actually to go about this problem is instead of having a single circuit we have what is called a circuit family for a problem, so a problem is solved by a circuit family.

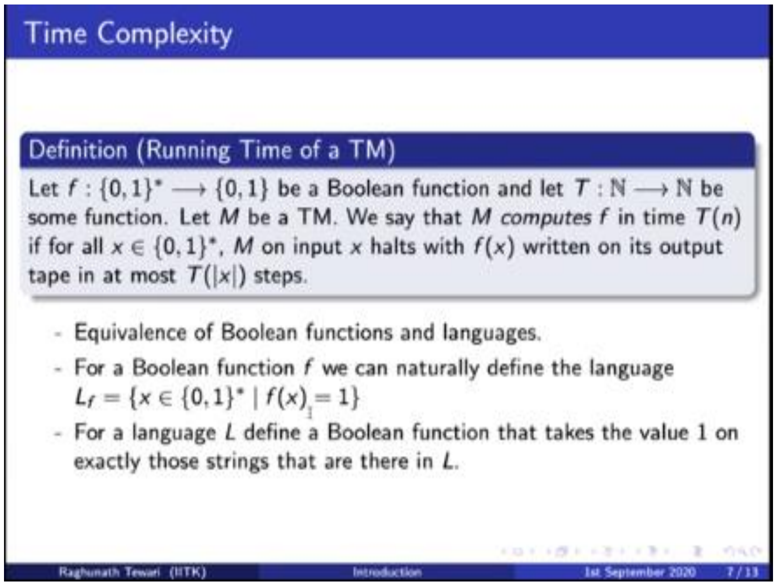
What is a circuit family? A circuit family is basically a collection of circuit where you have one circuit for each input length. That is if input length is 1, you have a circuit, if the input length is 2 you have another circuit if input length is 3 you have another circuit and so on. So you have a infinite family of circuits. Now for every input, for example if you have a circuit for input length 10, so all inputs which are of length 10, the circuit should correctly output the answer for all those inputs.

But the point is that it will not work for any input which has length 11. So that is what is called a non uniform model. So this is very important. Why do we study these two things? Now obvious question might comes to you that is why do we study circuits then? Why not just look at a Turing Machine? So the reason is sometimes it is easy to prove things in a non uniform model, sometimes it is easy to prove that you cannot do a certain task if you have a non uniform model like a circuit.

That is the reason why we study circuits and so of course Turing Machine will be our standard model of computation. Because it is what actually resembles physically I mean any kind of physical computational whether it is a laptop or computer or your cell phone or whatever. And the other thing is that circuits and Turing Machines are actually equivalent. So one can actually show or not exactly circuit but circuit family.

One can actually show that whatever you can do using a Turing Machine can actually be done by a circuit family. And whatever you can actually compute by a circuit family you can actually do by a Turing Machine. So when you actually study more about circuit we will formally define what a circuit family is and so on but I just wanted to give you a heads up before moving further. So next we move on to Turing Machines, so more about Turing Machines.

Refer Slide Time: 11:25)



Time Complexity

Definition (Running Time of a TM)

Let $f : \{0,1\}^* \longrightarrow \{0,1\}$ be a Boolean function and let $T : \mathbb{N} \longrightarrow \mathbb{N}$ be some function. Let M be a TM. We say that M *computes* f in time $T(n)$ if for all $x \in \{0,1\}^*$, M on input x halts with $f(x)$ written on its output tape in at most $T(|x|)$ steps.

- Equivalence of Boolean functions and languages.
- For a Boolean function f we can naturally define the language $L_f = \{x \in \{0,1\}^* \mid f(x) = 1\}$
- For a language L define a Boolean function that takes the value 1 on exactly those strings that are there in L.

Raghunath Tewari (IITK) Introduction 1st September 2020 7 / 13

The first thing is what is what do we mean by the running time of a Turing Machine? So suppose if we have a Boolean function f, so Boolean function is a function which takes a string as input and outputs either 0 or 1. So given a Boolean function f and function T from natural numbers to natural numbers. And let M be our Turing Machine. So M is a deterministic Turing Machine. We say that M computes the function f in time T of n, if for all x belonging to zero one star M on input x halts with f of x written on its output tape in utmost T of x number of steps.

In other words given a Turing Machine and a boolean function f that the Turing Machine is computing we say that M is computing f in time T of n, if for each and every string that is possible if M is given that string is input it must output fx on its output tape. So, whatever is that the value of the function on that input x and it can only take T of the length of x number of steps on that particular input not any more than that.

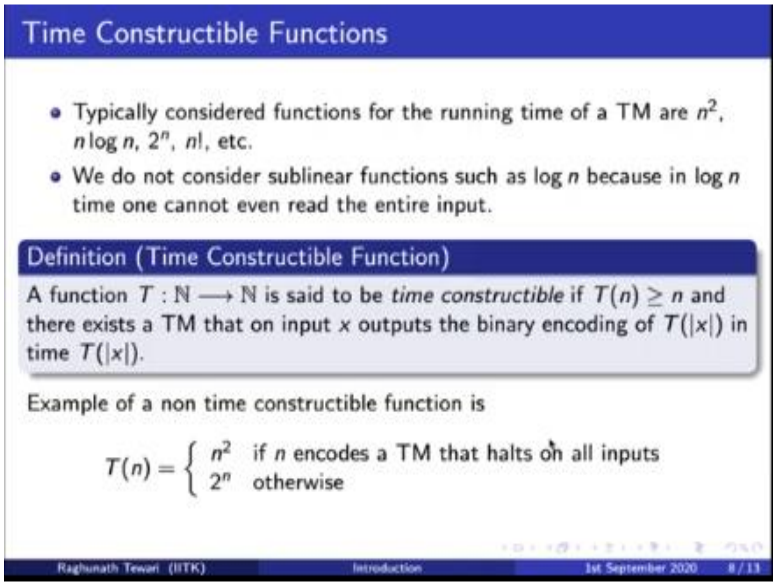
Now you might wonder that why I will be looking at a machine computing a function has opposed to a machine computing a language? So the point is that so they are actually equivalent. So, Boolean functions are languages are basically the same thing so that can easily be seen. So, suppose if you have a Boolean function f, we can naturally define a language out of that function.

So let us call that language L of f, so all the screens for which their function outputs 1. I put them inside that language and all the strings for which the function output 0. I put it outside the language. So the language consists of all strings which on which the function is outputting a 1. Now similarly given any language so what is the language? Language is just a given subset of 0 1 star.

So given any language I can correspondingly defined a Boolean function out of it. So all the strings that belong to the language I make the Boolean function map to one on those strings and all the strings that are outside the language, I make the Boolean functions map onto 0 on those strings. So I get a function which is from 0, 1 starts to 0, 1 which corresponds to this language L. The whole point is that we can actually interchangeably talk about Boolean functions and languages.

This is very important, because some at certain times we are going to talk about Boolean function and certain times we are going to talk about languages and when we say that the languages corresponding to the Boolean functions, this is what it means or when I say Boolean function corresponding to this language again this is what it means, you should be hear about this equivalence.

(Refer Slide Time: 14:43)



Time Constructible Functions

- Typically considered functions for the running time of a TM are n^2, $n \log n$, 2^n, $n!$, etc.
- We do not consider sublinear functions such as $\log n$ because in $\log n$ time one cannot even read the entire input.

Definition (Time Constructible Function)

A function $T : \mathbb{N} \longrightarrow \mathbb{N}$ is said to be *time constructible* if $T(n) \geq n$ and there exists a TM that on input x outputs the binary encoding of $T(|x|)$ in time $T(|x|)$.

Example of a non time constructible function is

$$T(n) = \begin{cases} n^2 & \text{if } n \text{ encodes a TM that halts on all inputs} \\ 2^n & \text{otherwise} \end{cases}$$

Raghunath Tewari (IITK) Introduction 1st September 2020 8 / 13

Now the next thing is when we talk about the running time of a Turing Machine, what kind of functions can the running time be or can it be any function of M, the answer is not it cannot be any function of M. There are typically considered functions for the running time of a Turing Machine are things like linear like n, n square, n log n, n to the power n, n factorial and many others.

So how we formally define, what kind of functions have allowed and what kind of functions are not allowed. So one type of function that is not allowed is if I have a deterministic Turing Machine is something like it is a log n, why is log n not allowed? The reason why log n is not allowed because, note that log n is actually less than n. If you have a running time of a Turing Machine which is log n, it just means that the Turing Machine is not even looking at its full input.

Because just to looking at the entire input it will take order n steps ok. if I just have to scan through the entire input, if it is looking at only a part of the input it just means that the Turing Machine will be let us say I mean for two different strings. I mean cannot even distinguish between two different strings, beyond log n many cells. So it can happen, you can, of course if a Turing Machine, which may be does nothing given an input it just accept or just rejects or something like that.

But the point is that it is not a nontrivial operation it is a very trivial operation and we do not on to allow such things. So this brings us to what I known as time constructible functions the type of functions that we allow the running time to be. So a function T is said to be time constructible if first of all as I said that T is greater than or equal to n, at least linear and secondly there exist a Turing Machine that on input x.

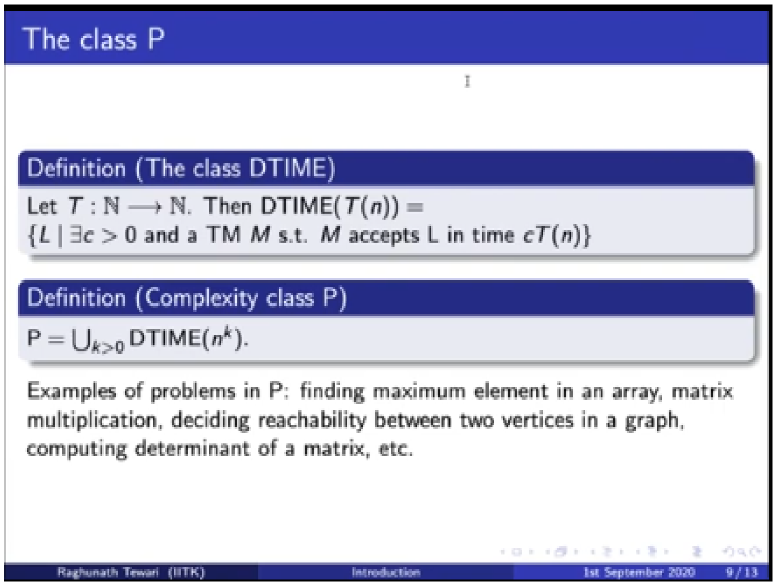
Output the binary encoding of t of x in time t of x. This is very important so this function T, that I have there should be some Turing Machine which should be able to output the binary encoding of T of x in time let us say at most T of x. Now functions like n log n or two to the power n or n factorial this is actually time constructible function. If you want you can actually try to move it by yourself that how can you design a Turing Machine which fills output.

Let us say given an input x of length n. So you can actually design a Turing Machine that when given an input x of length n can output a stream or output the binary encoding of 2 to the power n in time at most 2 to the power or any other function that is given in this list of examples. It is the time constructible function. Now you might wonder that what kind of functions are not time constructed?

So here is an example, so if I define a function T which is let us say equal to n square if n encodes the Turing Machine that also puts and it is equal to 2 to the power n otherwise this is a example of a function which is greater than equal to n, but it is not time constructible. The reason is I mean, given a number n, I mean how to I check whether n encodes the Turing Machine that halts the input or not because it is an undesirable problem.

So you cannot actually solve it using any Turing machine forget about doing it in time any or n square or 2 to the power n. So, this is an example of a non-time constructible function.

(Refer Slide Time: 19:17)



The class P

Definition (The class DTIME)

Let $T : \mathbb{N} \longrightarrow \mathbb{N}$. Then $\mathrm{DTIME}(T(n)) =$ $\{L \mid \exists c > 0$ and a TM M s.t. M accepts L in time $cT(n)\}$

Definition (Complexity class P)

$\mathrm{P} = \bigcup_{k>0} \mathrm{DTIME}(n^k)$.

Examples of problems in P: finding maximum element in an array, matrix multiplication, deciding reachability between two vertices in a graph, computing determinant of a matrix, etc.

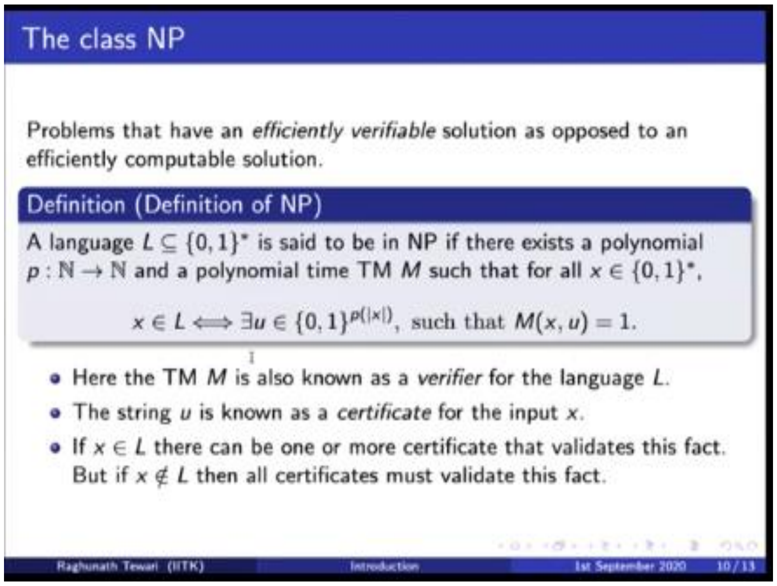
Raghunath Tewari (IITK) Introduction 1st September 2020 9 / 13

Now, we come to one of the major one of the most important classes that are studied in complexity theory the class P of the class corresponding to all problems that can be solved in deterministic polynomial time. So before that, I define this class D time. So given a function P from natural numbers to natural numbers the class D time T of n is the class of all languages such that there is a constant C and a Turing Machine M and the machine M accepts L in time c of T of n.

So essentially D time P of n is the class of all languages for which there is a machine which can actually decide that language in time, order T of n time, so c of T of n time and using this definition now I can define the class P as basically the union of all k greater than 0 DTIME n to the power k. So; all classes DTIME n to the power k greater than even I can say greater than or equal to 0.

So most of the problems that you have seen so far in your life the most of the problems that you have actually written algorithms for are actually problems that are in P like finding maximum element in a array, matrix multiplication deciding reachability between these between any two vertices in a graph DFS BFS computing determinant of a matrix and many more. There are probably very few examples of problems that you have seen which are not known to many people. Most of the algorithms that you write are usually polynomials time algorithms.

(Refer Slide Time: 21:12)



The class NP

Problems that have an *efficiently verifiable* solution as opposed to an efficiently computable solution.

Definition (Definition of NP)

A language $L \subseteq \{0,1\}^*$ is said to be in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time TM M such that for all $x \in \{0,1\}^*$,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)}, \text{ such that } M(x,u) = 1.$$

- Here the TM M is also known as a *verifier* for the language L.
- The string u is known as a *certificate* for the input x.
- If $x \in L$ there can be one or more certificate that validates this fact. But if $x \notin L$ then all certificates must validate this fact.

Raghunath Tewari (IITK) Introduction 1st September 2020 10 / 13

Now, we move to the next class that is the non-deterministic analogue of P and this is equally if not more important than the class P. So what was P? P is the class of those problems for which there is a polynomial time solution. On the other hand NP is the class of those problems for which we might not be able to construct a solution in polynomial time, but given a solution we can verify that solution in polynomial time.

So how do we define that formally? We can define the class NP as follows. So the class NP consists of all languages L such that there is a polynomial p and a deterministic polynomial time Turing Machine M. So again, please note that here M is a deterministic polynomial time machine such that for all x in 0, 1 star for all inputs x, I would accept x. So in other words the string x belongs to the language L if and only if there exist is string u whose length is at most polynomial in the length of x which is not that this p was a polynomial.

So this is a string u of length, p of x such that when the machine is given both x and u the machine accepts. So essentially what we are saying is that L language belongs to NP if there is actually another string u of polynomial L such that, so this language L belongs to NP if there is a deterministic time M, such that a string x belongs to the language if and only if there is some polynomial string u and if the machine is given both the string x and the string u then the machine can actually say yes.

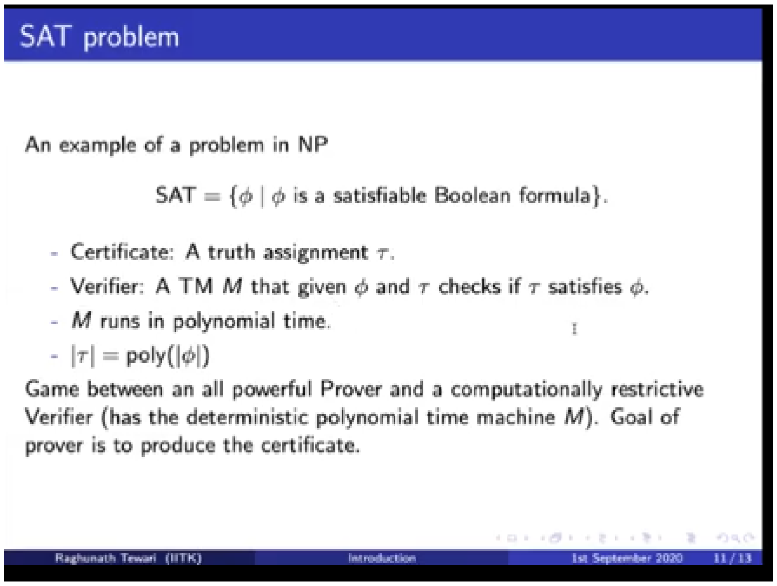
On the other hand if let us say if you look at the reverse of this statement what it says is that if x does not belong L then no matter what you pick no matter for each and every u the machine will always reach it automation will always output 0. Now some terminology this machine aim that we have which is taking the string x and the string u. So this is typically called the verifier for the language L, the deterministic verifier.

And the string u that we are giving which helps the machine M make a decision is what is known as a certificate. It acts like a certificate. So if x belongs to L, u kind of certifies the fact that yes indeed x belongs to L and it helps the machine if the right decision. On the other hand if x does not belong to L then no matter what certificate you give the machine will always reject. So no certificate actually exists.

So if x is in L there is actually at least one certificate that can actually be more than one certificate as well which validates the fact that x is in L and if x is not in L then all certificates that you give which have this length polynomial length will validate the fact that x does not belong to it. In other words, M will reject x, u. So this is what is known this is what we mean by efficient verification.

So we are not able to, so there is not deterministic polynomial time machine M, which when given x can say with whether to accept or not, but if we are given a certificate along with the machine then we can easily say whether to accept or not. So now let us look at a example.

(Refer Slide Time: 25:22)



SAT problem

An example of a problem in NP

$$\text{SAT} = \{\phi \mid \phi \text{ is a satisfiable Boolean formula}\}.$$

- Certificate: A truth assignment τ.
- Verifier: A TM M that given ϕ and τ checks if τ satisfies ϕ.
- M runs in polynomial time.
- $|\tau| = \text{poly}(|\phi|)$

Game between an all powerful Prover and a computationally restrictive Verifier (has the deterministic polynomial time machine M). Goal of prover is to produce the certificate.

Raghunath Tewari (IITK) Introduction 1st September 2020 11 / 13

We will look at this problem called the satisfiability problem. So what is the satisfiability problem? So a satisfiability problem consists of Boolean formula. So I am assuming all of you know what a Boolean formula is. So Boolean formula is a formula consisting of variables and gates such as AND, OR and NOT gates and the language set consists of those Boolean formulas which are satisfiable.

So what is the satisfiable Boolean formula? Satisfiable Boolean formula is a Boolean formula such that there is some assignment of values to the various to the variables of that Boolean formula, which makes the formula accept and on the other hand unsatisfiable Boolean formula will be one such that no matter what value. I mean, no matter what assignment of values you give to the variables of that formula it will never exit will always reject it will always output 0.

Now we say that formula is satisfiable when there is exists 1 satisfiable assignment. So this satisfiable assignment is this assignment is also what is known as a truth assignment. Now, how do we show that the problem is in NP? So we show that this problem sat is an NP very trivially. So my certificate here will be a truth assignment tau. So it is tau is nothing but an assignment of values to all the variables if this formula has n variables then tau is basically assigning either 0 or 1 to each of the n variables.

So it is just a string of length n. It is a Boolean stream of length. What is the verifier? The verifier is a Turing Machine that when given phi and tau it just checks whether tau satisfies phi. If I just plug in the values of tau to each and every variable in phi whether it evaluates to 1 or it evaluates to 0. Now this M can be designed in polynomial time which takes a Boolean formula and it takes a assignment of values to all the variables of variables of the Boolean formula.

And then says whether it output 0 or 1. So this can be designed in polynomial time, and of course the length of tau is polynomial not only is it polynomial it is only linear. So this proves that the problems SAT is belongs to NP. Now sorry, another way of trying to understand this or trying to see this certificate verifier model is to think of this as a game between and all power full prover and a computational restrictive verifier.

So there are these two people now the prover is all powerful the prover actually given an input it can immediately answer whether that input belongs to a language or not. For example the prover given a formula 5 it knows whether it is satisfiable or not but the verifier is computationally restricted if the verifier only has deterministic polynomial amount of power. You can think of it like that.

I mean it cannot for example given a formula phi if you have only given phi how do you check whether it is satisfiable or not? One way to check whether a formula is satisfiable or not; is to cycle over each and every truth assignment. Now how many possible truth assignments are there? There are 2 to the power n proof assignment. Now this formula can actually be satisfiable on any one of those truth assignments that you do not know which one it which ones satisfies that formula and which one does it?

So and if you actually have to cycle through all of them it is much more than polynomial and as it turns out we cannot do actually much better than 2 to the power n. So this is an algorithm this is a deterministic algorithm which takes to the power n time, but unfortunately even the best known algorithms do not solve this problem in time that is any many slightly better than 2 to the power n but not much better certainly nowhere close to being in polynomial.

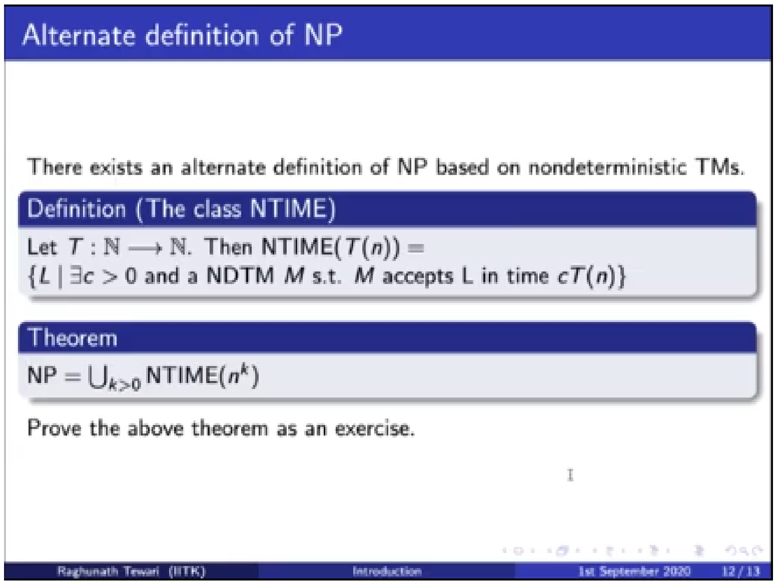
So the point is the verifier cannot actually solve the problem directly. So the goal of the prover here is to produce the certificate so the prover actually wants to convince the verifier somehow that to accept the input. Now what the prover can do is because the prover is all powerful given any input let say given a Boolean formula 5 the prover actually can produce the certificate and then pass on so the input is accessible to both the prover and the verifier.

Now the prover actually gives the verifier the certificate. Now if the input was indeed in the language then we know by the definition that there exists some certificate. Now the; verifier when given the input and the certificate can actually check in polynomial time, whether it is in the language or not. On the other hand suppose if you have an input which is not in the language then no matter what certificate the prover produces the verifier always ends up rejecting.

Because we know that by definition if a language is not in NP then for each and every certificate of polynomial length the, machine M, when given x and that certificate does always reject. So this is broadly how we show that a problem belongs to NP. So we produce a certificate we say what the certificate should be we give the verifier algorithm and then just argue why that algorithm runs in polynomial time and why the length of that certificate is also polynomial in the length of the input.

So usually the third and the fourth steps are quite easy it is only producing the certificate and the verifier’s algorithm which is little bit non-trivial at times but again usually that also follows from the definition itself, the definition of the problem itself.

(Refer Slide Time: 31:37)



Alternate definition of NP

There exists an alternate definition of NP based on nondeterministic TMs.

Definition (The class NTIME)

Let $T : \mathbb{N} \longrightarrow \mathbb{N}$. Then $\text{NTIME}(T(n)) =$
$\{L \mid \exists c > 0$ and a NDTM M s.t. M accepts L in time $cT(n)\}$

Theorem

$\text{NP} = \bigcup_{k>0} \text{NTIME}(n^k)$

Prove the above theorem as an exercise.

Raghunath Tewari (IITK) Introduction 1st September 2020 12 / 13

Now so if you recall so we started off by talking about the class polynomial time. And we define polynomial time in terms of deterministic Turing machine, but when my talked about NP we defined NP in terms of the certificate verifier model. So is there any other definition of M? And in fact the name NP itself stands for non-deterministic polynomial type. So there actually exists another alternate definition for NP based on non-deterministic Turing Machines.

So, what is that definition? So similar to DTIME we can define the class N time Tn as class of all languages, such that there is a constant C and a non-deterministic Turing Machine M and the machine M accepts L in time c times T of n and now I can define NP as union over k greater than 0 of n time n to the k. Now, why are these two definitions the same? So here is that a one definition of NP using n time using this class NTIME.

There was another definition of using NP using the certificate verifier method. So I will just leave this as an exercise today for you to prove that if you have any languages which see you have these two definitions of NP. So you take any language in NP using let us say the previous definition. So they can only be one definition for an entity. So if you take a language with belongs in NP, you can prove that language also belongs to union of k greater than 0 n time into the power k.

Similarly if you take a language with belongs to the right hand side that is union over k greater than 0 n time n to the power k then it also belongs to NP by the certificate verified definition. So please try to prove this by yourself. You do not have to submit these exercises that I given class unless I explicitly tell you that this is something that you have to submit. This is only for you practice. So, I will end here today.