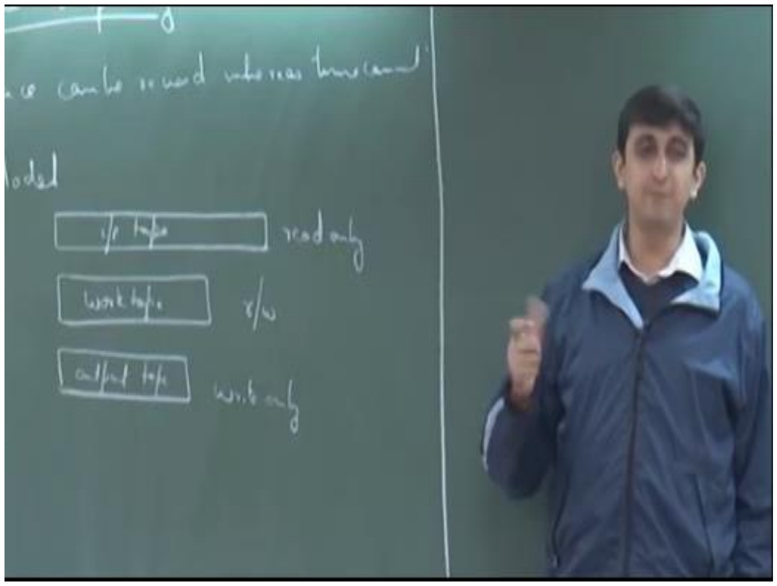
Computational Complexity Theory

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

Lecture -06
Introduction

So, today we will look at another resource bounded model. In particular we will consider space as our resource and will try to study what problems can be decided and not decided where space is our resource.

(Refer Slide Time: 00:39)



So what is the fundamental difference between time and space complexity? So far we have been discussing about time complexity, in particular we saw that so what is time complexity? It is basically the measure of the number of steps the turing machine takes. And then with respect to that we defined various classes for example polynomial time, exponential time and then again in the non deterministic setting also we looked at these functions.

So, again we can build up this whole theory with respect to space as well where instead of looking at the number of steps the turing machine is taking on a particular input, we count the total number of cells the turing machine is using to perform its computation. We do not bother about how many steps it takes; it is just the number of cells that turing machine is taking. And when we are looking at the number of cells, we restrict ourselves to the work day.

So I will just soon motivate why, we do that? But that is basically what the model is. So, what is the fundamental difference between these two resources? So again I mean the main difference is that time is a quantity which cannot be reused. I mean once you have taken certain number of steps you have gone past it and it cannot be used whereas space is a resource which can be reused. If you are able to do so in an appropriate manner it can be done.

So the first point is that space, if you consider it as a ‘resource’ can be reused, whereas time cannot. So, this is the main contrasting feature between these two resources. So what is our model? So our model of computation would be turing machines and in particular we will look at the following types of turing machines. So, our turing machine, will so our model would be as follows; so a turing machine would have an input tape and as always we would assume that the input tape is read only.

There would be a work tape, with both read and write capability so it is read and write and there will be a output tape. That is write only. So, when we are looking at decision problem that is given in an input we want to give a yes or no answer. Then the output tape is redundant in some sense I mean we can think it off as the turing machine just going into an accepting state or a rejecting state.

Or if you want to keep the output tape you can just think of it as its outputting a 0 or a 1 bit on the output tape, so that is the model. But the output tape becomes necessary, for example when we are looking at reductions space, bounded reductions, where we want to compute a certain function not just a boolean function but arbitrary function. So that is one place where output tape becomes necessary.

Also places where we are looking at instead of decision problems, let us say search problems. For example given n numbers I want to sort them or problems of similar nature. So for our general model we will just keep the output tape. So now when we say that the number of the space used by a turing machines is the number of cells that it uses, which cells do we count?, so that is an important question and that is something that needs to be fixed a priori.

Do we count the input tape work tape and output tape or some subset of them or whatever so clearly we do need to count the work tape because that is where we are performing our computations so that needs to be counted. And I will give the answer to that question and so when we are looking at space as our resource we do not count the input tape or the output tape. And the reason for that is that most often well I should not say most often but often you do come across problems.

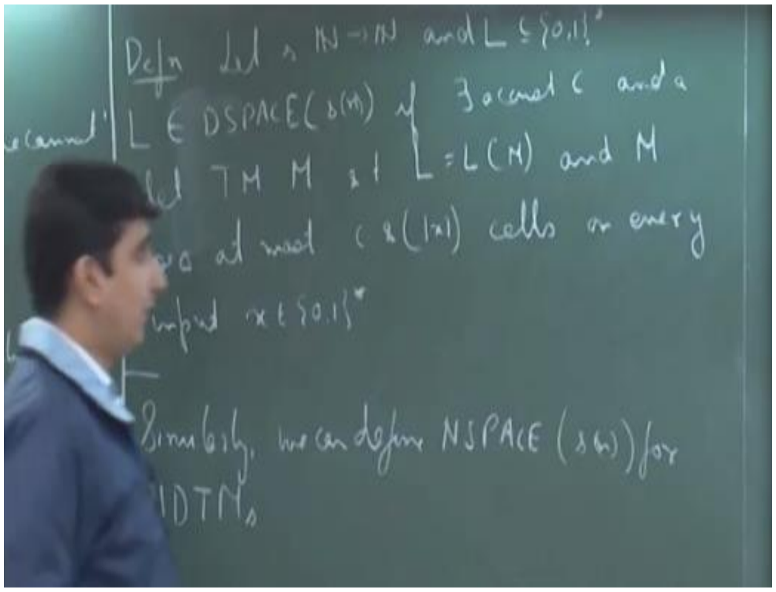
Where the amount of work tape cells that you are using is much less than what the size of your input is. So I will give you an example, for example you are trying to compute what is the maximum element in an array? So the array is given on your input tape and you want to compute the maximum so what do you need? Just think you need one cell or so cells can only hold 0 or 1. So they are you are only looking at binary alphabet it is about log. So basically I mean what is the algorithm?

The algorithm is that you keep a variable where you store the maximum and keep on updating it as you scan the array. And if you look at what is the size of that variable as a function of your input tape, it is about log of the length of the input tape, because if there are n elements you need about log n bits to represent one element. Well it, so that is a good question so it is about n log n you can think if you want to think of it that way the input tape has n log n cells.

So it is some polynomial in n, because it does not matter i mean whether it is n or n log n or n square the point is that the amount of work tape cells you are using is order of that many thing. So when you take log of that number it comes to just about log n. So that is the thing, so as we can see that we can do computations where the number of cells required is much less than the number of cells in the input tape.

So this is again a contrasting feature with time bounded computations, in time bounded computations we saw that, I mean the number of steps that a turing machine takes has to be at least n in order to in order for the turing machine to at least to read the entire input. So the space is basically the space used by the work tape. This is the space that we consider and that is another reason why we restrict our input tape to be only a read only tape because we do not want it to store additional bits and use that tape for computation. So that is our model.

(Refer Slide Time: 09:51)



Defn Let $s: \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$

$L \in DSPACE(s(n))$ if $\exists$ a const c and a det TM M s.t. $L = L(M)$ and M uses at most $c\, s(|x|)$ cells on every input $x \in \{0,1\}^*$

Similarly, we can define $NSPACE(s(n))$ for NDTMs

So now with this model let us define complexity classes with space as our resource. So let S, be a function from natural numbers to natural numbers and L be some language. So we say that L belongs to DSPACE s of n, if there exist a constant c and a deterministic turing machine m such that L is equal to L of M, and M uses at most c dot s of x cells on every input x belonging to 0, 1 star.

So similarly we can define the class n space as well, so I would not write down the entire definition. So similarly we can define NSPACE s of n for non deterministic turing machines. So now what can we say, about certain basic relations between deterministic and non deterministic turing machines.

(Refer Slide Time: 12:46)

Well one thing is quite obvious I mean if I want to look at with respect to a function let us say s of n, i just state it as a fact. DTIME s of n is contained in DSPACE s of n, why is that? Because if a turing machine is let us say running for s of n number of steps it cannot use more than s of n memory cells, more than that many memory cells. So that is an obvious upper bound, because in each step I mean at most you can visit one extra cell in your work tape.

And of course DSPACES of n is contained in NSPACES of n because every deterministic machine is also a non deterministic machine which is not using its non deterministic machine. So here let me mention one more property that we use. So we do not restrict our or we do not restrict a non deterministic turing machine to halt all possible inputs. So if you look at the definition I mean not only non-deterministic even the deterministic, so let me use the word non deterministic.

So if you look at the definition here we are not requiring that on a given input x the turing machine has to halt and then produce its output. All we are requiring is that it has to use this many memory cells for a given function x, s of x, or s of n and the reason why we do not impose this restriction is because we only look at so the reason is that we only look at the class of space constructible function.

So space constructible function is again defined in a way similar to time constructible functions, that is we say that a function is space constructible if there is some turing machine which given x would write would output s of x on its output tape using space at most order of s of x. So let me just define that so a function s from n to n is said to be space constructible, I am just replicating the definition from time constructability and only replacing time with space.

If there exist a turing machine M that given x outputs s of x on its output tape using order of s of x space. So what we require again in this entire theory of space complexity is that; all the functions that we are considering they are all space constructible functions. In other words given a function there is some turing machine which can do this. So now that we have this restriction on our functions let us say in this function s, I claim that it is not necessary to restrict a turing machine to always halt.

And the reason for that is, what we can do is just imagine a turing machine which has a counter attached to it. So the way to think of a counter is that just think of it as having two work tapes. In one in one work tape it is just performing its usual computation and in the other work tape it is a just keeping a counter. That is, whenever the turing machine takes a step that counter gets incremented by 1.

Now if you have a turing machine which uses let us say, some constant c times s n number of cells; the total number of different configurations that that machine can get into is 2 to the power c of s of n or some order of s of n number of configurations. So now what we can do is that so we can just fix that counter that, whenever the counter reaches that maximum value ok whatever that constant c is that 2 to the power c times s of n value, will just make that turing machine halt.

Because then what we can claim is that, if even within this many number of steps the turing machine has not halted; then it will again go back to some configuration which it already was in at an earlier step, because we know that the total space is bounded by s of n, and if it is going back to a configuration in which it was already in it; basically means that it is running into a loop.

So what is the point of having a turing machine which goes into a loop. So that is why we can just make it halt after that many steps and we are using this fact that the function is space constructable to know exactly what the value of s of x is given x. So that is the reason why we do not have a strict enforcement in our definition because without loss of generality we can always assume that it would always halt, so is that clear to everybody, any questions?

(Refer Slide Time: 21:04)

So let us move on to a very important concept related to space complexity that we would see again and again this is what we call a ‘configuration graph’ of a turing machine. Again this is something which is very intuitive and very useful in the study of space complex. So intuitively the configuration graph is something which captures the entire computation of a turing machine M on an input x.

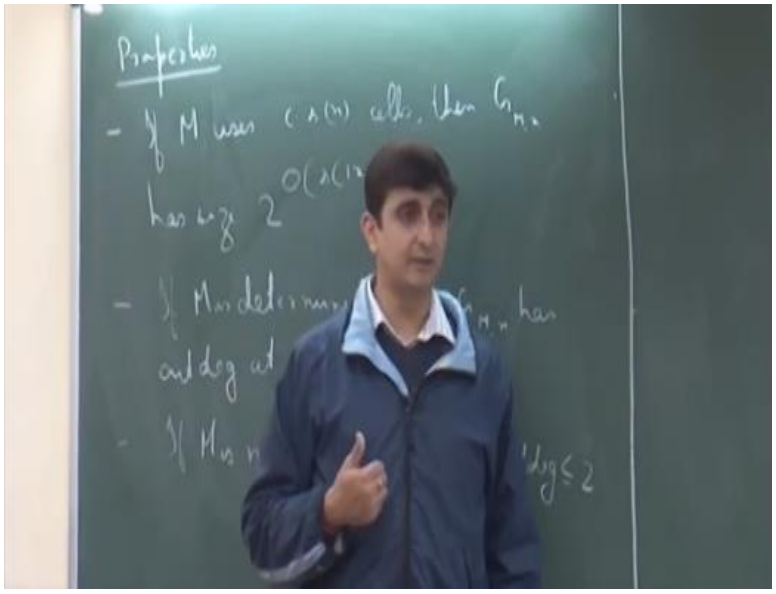
So it is the configuration graph is with respect to a given turing machine and a given input. And this is denoted by g subscript m, x. So let us define what the configuration graph is. So the vertex set of this graph is the vertex set is the set of all possible configurations of the machine, that is so what are the factors on which it depends? What are the factors on which configuration of a turing machine depends on?

So it depends on the current state. It depends on the content of the work tape, so we do not care about the content of the input tape because that is fixed. Because we are looking at the configuration graph with respect to a given input so the input head I mean the content of the input tape is always fixed. So we look at the contents of the work tape so that determines and we also need to know, what are the positions of the various heads of the turing machine?

For example where is the input head where is the work tape head and so on. So the position of M's heads and I think that is pretty much. So essentially all these things are constant so the number of states is constant I mean if you once you fix a machine M this is a constant but these need not be constant because they can they basically depend on what x is i mean how large x is. So we will come to what is the size of the vertex set in a moment.

But for now think of it as that every vertex in this configuration graph corresponds to a configuration of the machine. So how do we define the edges? So there is an edge from a configuration C to C prime, if M can go from configuration C to configuration C prime in one step according to the transition function. So basically all so basically the graph that we are looking at is a directed graph and we put a directed edge from a configuration C to C if C prime can be reached in one step from C.

(Refer Slide Time: 27:31)


Properties
- If M uses c s(n) cells, then $G_{M,x}$
has size $2^{O(s(n))}$
- If M is determ... $G_{M,x}$ has
outdeg at ...
- If M is n... outdeg ≤ 2

So now let us look at some properties of this graph. So firstly what is the size of this graph? So let us say that M is a turing machine with a space bound, s of n; what would be the size of this configuration graph? So if M is say if M uses some c dot s of n cells then G M of x has size 2 raised to the power order of s of we put it as x. Where the constant in this big O term is determined by various things for example what is the size of the state set and how many tapes does it.

I mean generally, we assume it has only one work tip but we can also extend it to some constant number of work tapes and that big O takes care of all those constants. What else, so note one thing that in this definition in the definition of a configuration graph we did not say whether M is deterministic or non-deterministic; in other words this entire definition also carries on for example if M is non-deterministic.

So suppose if M is deterministic what can we say about this graph? So if M is deterministic then G M of x has out degree 1 or at most one let us say because some configurations can be halting configurations from where you do not have any outgoing edges. Yes for all vertices it will have out degree at most one because it is a deterministic machine. So from a configuration there is only one other configuration that you can go to once you fix the input.

And what if M is a non deterministic so then G m of x has out degree at most two, because now you the machine can non deterministically make a choice from any given configuration between two different configurations. So from C it can go to let us say either a configuration C prime or a configuration C double prime. Well I mean that is just again a convention I mean you can think of it as being more than 2 also.

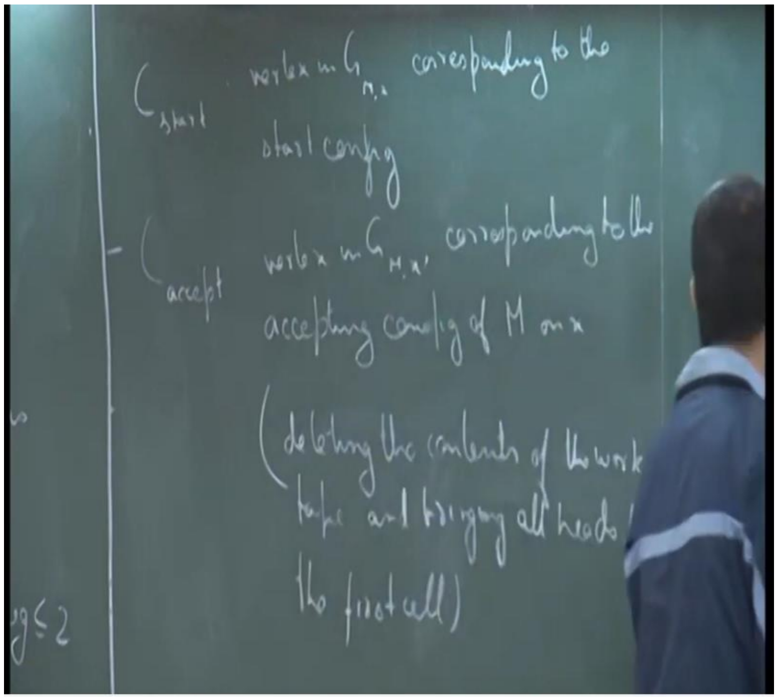
So one reason is because the way we define non deterministic machines. So one way to think of it is that the machine has two transition functions, two deterministic transition functions and at each step the machine chooses whether to apply transition function 1 or transition function 2. And whether it should proceed according to transition function 1 or transition function 2, so that is one way to think of non deterministic machines.

And now, so if you look at that model then the configuration graph will have out degree at most 2 depending on which transition function it is choosing. More than a property this can be thought of as something which can be assumed without loss of generality, so we can assume one more thing without loss of generality. Now so before I come to that, so now as i said earlier that this idea of configuration graph is very useful in studying complexity theory.

So why is that, I mean how does this graph help us in let us say determining whether a given input x belongs to the language of a machine or not. So, how can we determine? So let us say we are given that this graph G M x, that is for an input x we have constructed what this graphics. So can you look at this graph and determine whether x belongs to the language of M or not. So your idea is essentially correct but we have to be a little bit careful about fixing certain things.

So our start configuration is unique. So once you fix an x there is always a unique start configuration that is the machine is in the start state, all the input heads are at the first location and the well the work cell is completely empty.

(Refer Slide Time: 34:25)



C_{start} : vertex in $G_{M,x}$ corresponding to the start config

C_{accept} vertex in $G_{M,x}$, corresponding to the accepting config of M on x

(deleting the contents of the work tape and bringing all heads to the first cell)

So there is a unique configuration corresponding to the start configuration so let us call that vertex as C start. So, C start is the vertex in G M x corresponding to the start configuration. So again we can assume without loss of generality that the machine has only one accepting configure accepting state. But note that it can have many accepting configurations in other words for every possible string that can be present in the work tape.

And for every possible position in which the heads of the turing machine can be present at each of them will define a different configuration which are all accepting configurations. So, it is not clear whether checking for reachability between 2 fixed vertices determines whether x belongs to L of m or not. So is that clear because see what can happen is so as i said a configuration determines is dependent on all these factors.

So the state can be a an accepting state, but these values that is the content of the work tape and position of M’s heads I mean they can take various values. But again what we can do is that so let us say that when M reaches an accepting state what we can do is that after that we can just go ahead and delete let us say everything that is present in the work tape. So we fill the work tape with the all blank symbol.

And let us say we take all the heads and bring them to the first cell in every tape. So basically what I am trying to do is that without loss of generality I am just trying to reduce all the accepting configurations to just one accepting configuration. So what we can assume is that, there is 1 vertex in the configuration graph I will call it C accept so this corresponds to the accepting configuration of M on x.

And this configuration basically, we I mean we can assume that there is a unique such configuration by deleting the contents of the work tape and bringing all heads to the, let us say the first cell in each tape. So, now yes so now we can just do a reachability check that whether in this graph there is a path from C start to C accept or not and that would determine whether x is in the language or not.

See I am stating all these things that is deleting contents of work tape and many such things but you should always convince yourself that these things are possible. Given a arbitrary turing machine which is just halting on x, you can construct an another turing machine let us say which is doing the same thing but then at the very end it is doing this additional work. So you should always convince yourself that these things are possible.

And just not take my word something guaranteed and please stop me I mean if something is not clear, do not hesitate to ask me.

(Refer Slide Time: 39:52)

Fact 2 x ∈ L(M) ⟺ ∃ a path from C_start to C_accept in G_M,x
Fact 1a NSPACE(s(n)) ⊆ DTIME(2^O(s(n)))
L NTM M
G_M,x having 2^O(s(|x|))

And now I will just state this as a fact that an input x belongs to the language of a machine if and only if there exists a path from C start to C accept in G M of x. So this is the crucial fact because of which configuration graphs are studied because now an entire computation instead of thinking of it from a turing machine point of view. We have just reduced it to a graph theory problem. So I just erased that earlier relation that I had written between the various time and space classes with respect to a function s of n but actually we can now extend that.

So I will just call that fact 1 a, so what I wrote earlier was that DTIME s of n is contained in DSPACE s of n and DSPACE of n by definition is contained in NSPACE s of n right. So now I can extend that to the following result that, NSPACE s of n is contained in DTIME again of course not s of n but 2 to the power big O of s of n. So that is if you have a non deterministic machine which uses space at most s of n you can simulate it by a deterministic turing machine which runs for at most two to the power order s of n number of steps.

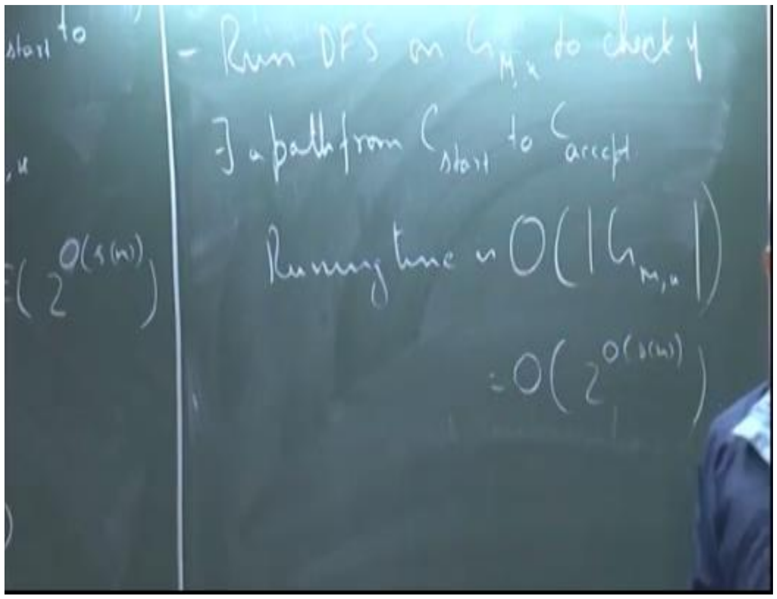
And how can you do that? Big O, this is big O. I mean I just cannot write it as 2 to the power s of n because there can be some constants associated over here so that is why. Why do you think this containment is true or how do you prove this? So think in terms of the graph, so suppose you have ok so suppose if you have you have a language in this class L, it means that there is some non-deterministic machine which is using space order s of n to decide L.

What would be the size of the configuration graph of that machine on any input? So there is a so assume that there is some L that is belonging here. And this happens via some non deterministic machine m, what would be the size of the configuration graph? Exactly right since the machine is restricted to use this much amount of space the total number of different configurations as we just argued little while earlier is at most 2 to the power order s of n.

So therefore for all inputs x, G M x has size 2 to the power order s of x. So now what? How do you come up with a algorithm that works in this much amount of time? What about graph traversal algorithms? So now we know that what we have all that we have to decide is that in this graph we have to check if there is a path from C start to C accept or not. So the entire problem of checking whether x belongs to L of M or not; is checking reachability.

So, for example you can use DFS or BFS so I hope that all of you here have seen DFS BFS. So, that is all, so what you can do is you just let us say run a DFS algorithm on this graph.

(Refer Slide Time: 44:58)



start to

, x

$(2^{O(s(n))})$

- Run DFS on $G_{M,x}$ to check if $\exists$ a path from C_{start} to C_{accept}

Running time is $O(|G_{M,x}|)$

$= O(2^{O(s(n))})$

So run DFS algorithm on this graph to check if there exists a path from C start to C accept. If it is then you accept x otherwise you do not accept. And what is the running time of DFS? Number of vertices plus edges and since our graph has a constant degree the number of edges is order of the number of vertices. So therefore, running time is big O of, so let me just write it as the size of this graph, which is nothing but order of two to the power s of n.

So that is why it is in this class, any questions? Which one? Well, we know that I mean there can be problems in this class, so this problem is not complete for this class. So there can be other problems in this class for which we do not know n spaces of an algorithm. We are just telling that this problem takes that many steps, but there can be many other problems in that class. So we will stop here.