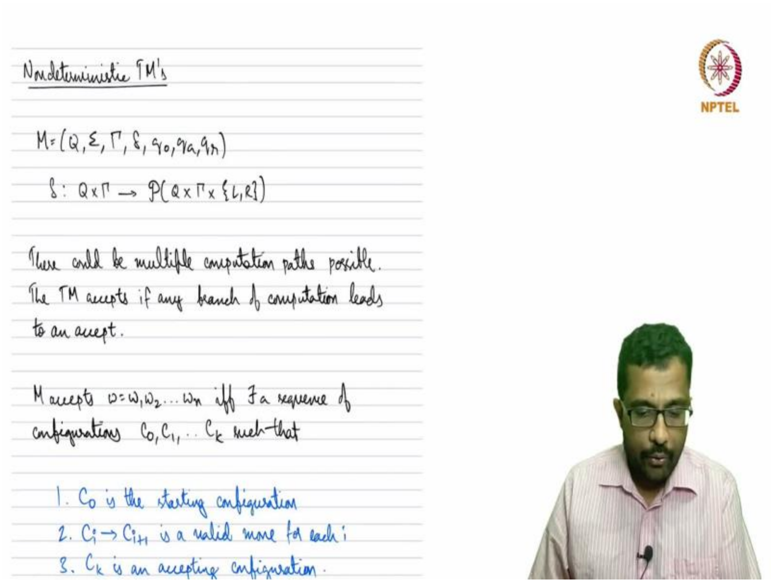
Theory of Computation

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

Non-Deterministic Turing Machines

(Refer Slide Time: 0:18)



Nondeterministic TM's

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$

There could be multiple computation paths possible. The TM accepts if any branch of computation leads to an accept.

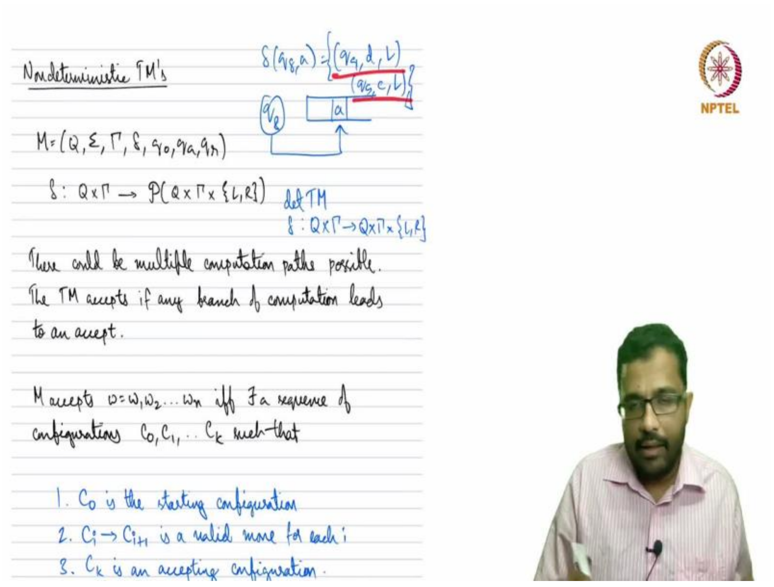
M accepts $w = w_1 w_2 \ldots w_n$ iff $\exists$ a sequence of configurations $C_0, C_1, \ldots C_k$ such that

1. C_0 is the starting configuration
2. $C_i \rightarrow C_{i+1}$ is a valid move for each i
3. C_k is an accepting configuration.

Hello and welcome to lecture number 30 of the course Theory of Computation. In the past couple of lectures, we saw Turing machines and we saw the standard model of Turing machines with single tape, then we saw multi tape Turing machines. In this lecture, we are going to see a different type of variant, which is non-deterministic Turing machines.

So, the Turing machines that we saw were deterministic, but now we are going to see non-deterministic Turing machines. So, this is similar to the non-determinism that we saw in 2 previous machine models. So, we saw deterministic finite automata and then we saw non-deterministic finite automata. Then we saw PDAs, pushdown automata, which also had non-determinism built into them. Similarly, we have non-deterministic Turing machines. So, the main difference between deterministic and non-deterministic Turing machines.

(Refer Slide Time: 1:28)



Nondeterministic TM's

$\delta(q_8, a) = \{(q_9, d, L)\ (q_5, c, L)\}$

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$

det TM

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

There could be multiple computation paths possible. The TM accepts if any branch of computation leads to an accept.

M accepts $w = w_1 w_2 \dots w_n$ iff $\exists$ a sequence of configurations $C_0, C_1, \dots C_k$ such that

1. C_0 is the starting configuration
2. $C_i \to C_{i+1}$ is a valid move for each i
3. C_k is an accepting configuration.

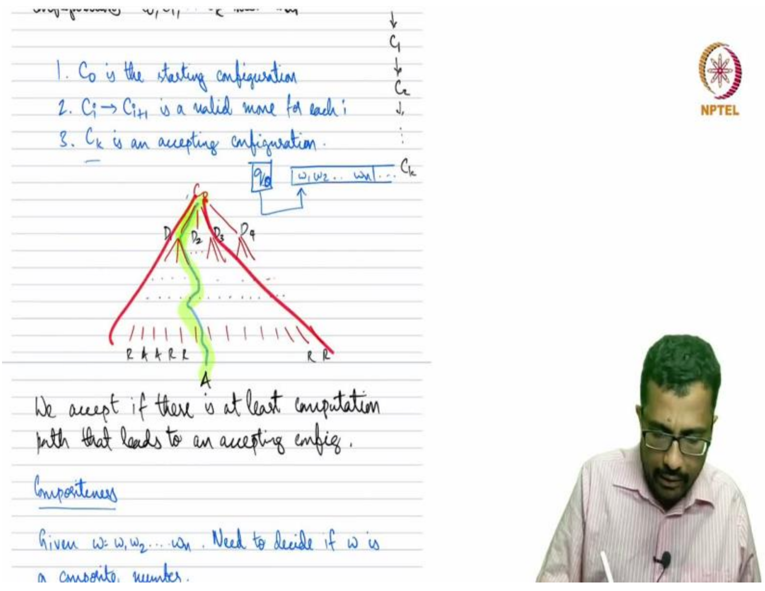
So, we assume just one tape, single tape. So, the point is this, that if you are at a certain state, let us say you are reading a symbol a and you are at a certain state, let us say q_8. In a deterministic Turing Machine $\delta(q_8, a)$ is defined to be something, let us say $(q_9, \mathrm{d}, \mathrm{L})$. But in a non-deterministic Turing machine, there could be multiple things that you could do.

So, perhaps $(q_9, \mathrm{d}, \mathrm{L})$ could be just one of the options. The other option could be $(q_5, \mathrm{c}, \mathrm{L})$. maybe something else $(q_4, \mathrm{c}, \mathrm{R})$. So, there could be multiple choices available, not just one option. The number of choices available could also be 0. If you recall, just like in NFA's for the current state and the symbol read you could have multiple choices, you could have just one choice or you could have 0 choices also.

Similarly, here you could have multiple, just one choice or 0. Formally speaking, in the case of deterministic Turing machines, we have the same 7 tuple, $M = (Q, \Sigma, \Gamma, \delta, \mathrm{q}_0, \mathrm{q}_\mathrm{a}, \mathrm{q}_\mathrm{r})$. Same thing here. The only difference is in the way δ is defined. So, in the case of deterministic Turing machine we had $\delta : \mathrm{Q} \times \Gamma \to \mathrm{Q} \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$.

Here instead of $\mathrm{Q} \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$, we mapped to the power set of $\mathrm{Q} \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$. This is because, in the case of non-deterministic Turing machines there could be multiple options. So, either you could go to q_9, write d, then left or you could do, if you just consider these as the 2 choices, $(q_9, \mathrm{d}, \mathrm{L})$ or $(q_5, \mathrm{c}, \mathrm{L})$. So, you could do that or this. There are 2 options listed here, but there could be more than 2 also, there could be 0 also, there could be 1 also.

(Refer Slide Time: 4:00)

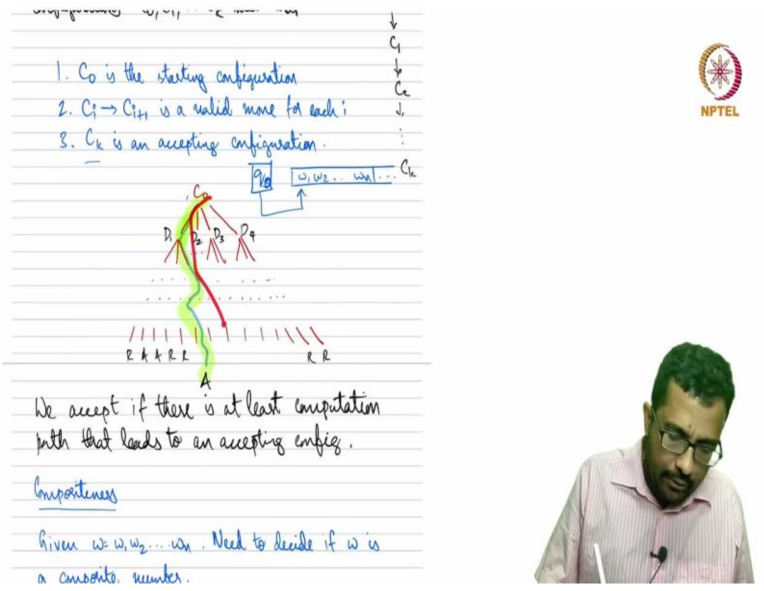


1. C_0 is the starting configuration
2. $C_i \to C_{i+1}$ is a valid move for each i
3. C_k is an accepting configuration.

We accept if there is at least computation path that leads to an accepting config.

Compositeness

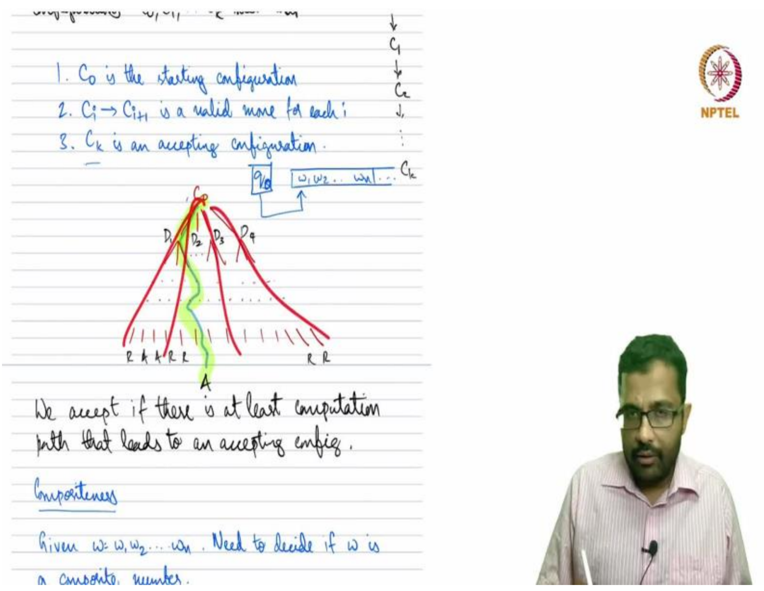
Given $w = w_1 w_2 \ldots w_n$. Need to decide if w is a composite number.


1. C_0 is the starting configuration
2. $C_i \to C_{i+1}$ is a valid move for each i
3. C_k is an accepting configuration.

We accept if there is at least computation path that leads to an accepting config.

Compositeness

Given $w = w_1 w_2 \ldots w_n$. Need to decide if w is a composite number.


1. C_0 is the starting configuration
2. $C_i \to C_{i+1}$ is a valid move for each i
3. C_k is an accepting configuration.

We accept if there is at least computation path that leads to an accepting config.

Compositeness

Given $w = w_1 w_2 \ldots w_n$. Need to decide if w is a composite number.

So, as a result, what happens is that... The starting configuration is fixed, right. It is the starting state followed by the input symbol. The starting configuration is input string, so, you have you have $w_1 w_2 \dots w_n$ and you are at the starting state let us say q_0, then the head pointing to the first location. So, this is the starting configuration which I am calling it C_0.

And in case of a deterministic Turing Machine these conditions were the same. When does this string get accepted? A string gets accepted if there is a sequence of configuration such that C_0 is a starting configuration each move is valid. $C_i to\ C_{i+1}$ is valid C_0 to C_1, C_1 to C_2, C_2 to C_3 and so on and finally C_k is an accepting configuration. What I mean by that is C_k contains the accept state.

Same thing here also. There should be a sequence of configurations such that C_0 is starting configuration, C_1 should be a valid successor of C_0, C_2 should be a valid successor of C_1 and so on. Finally, the C_k should be valid accepting configuration meaning the state should be the accepting state. So, what are the differences then between deterministic and non-deterministic Turing machine? The key difference is this in a deterministic Turing machine we had something like this, we had the starting configuration.

The starting configuration completely tells us which move to take meaning you know the state you know where the head is pointing. You know the next state, you know what to write next and where to move. So, the starting configuration tells us what is the successor, that successor tells us what is the next successor and so on. So, the computation goes in a kind of a linear path and finally, you reach some states which could be accepting or you never reach anything.

However, in a non-deterministic Turing machine, let us say C_0 could have multiple states as successors, it could have let us say, let me just write D_1, D_2, D_3, D_4 many states as succesors, each of the D_1, D_2, D_3, D_4 could also have further many states. So, now, when do we accept? We accept when we can find a computation path something like this. We accept when we find some computation path finally leading to an accept state. So, this should be an accepting state, or accepting configuration.

So, some computation path has to be there. So, there are multiple configurations that are valid successors of the starting configuration. For each of these successors, there could be multiple valid successors themselves and so, at the end there could be multiple possibilities. Some of them could be reject, some of them could be accept, some of them could be reject and so on.

All that you care about is, is there at least one start to, one path that leads to an accepting configuration.

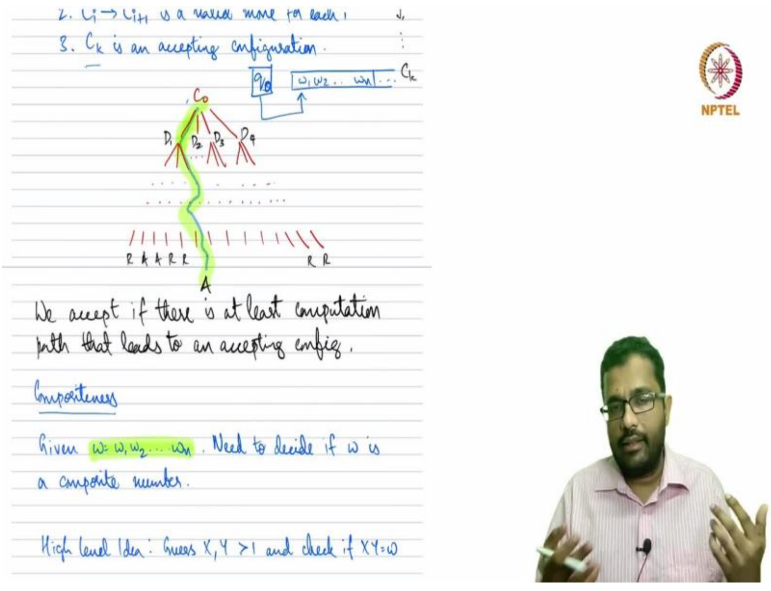
So, we accept if there is one, by one I mean at least one, if there is at least one computation path that leads to an accepting configuration. So, there could be multiple such accepting configurations, but as long as we find one, we are good. It could be that all the configurations are rejecting configurations except one. That is also good enough.

So, we say that non-deterministic Turing machine accepts even if there is one computation path that accepts. So, here we have just one computation path that accepts. Even then we are fine. Here there are other computation paths, that is okay. But we accept if there is at least one computation path that accepts. We reject if all the paths are rejected. If none of the paths accept, we say it is rejected. So, that is the thing. So, there is a certain asymmetry happening here, in between the accept and the reject. We accept if at least one path accepts and we reject if all the paths none of the paths are accepting.

So, there is this. So, it is not like one path rejects then you reject. For rejecting you want everything to reject. For accepting you just need one accepting computation path. So, non-determinism should be thought of like I have said before. If there is a non-deterministic Turing machine, it somehow magically finds the accepting path.

In non-deterministic Turing machine, you should not think of it as trying out all these paths. No, that is not how it happens. It somehow magically is able to do all these paths in parallel. So, if you see here there are multiple paths, one in the left, one on the right, one through the middle and so on. It will try out all these possibilities and magically find the correct accepting path if it is there. So, the time taken by the non-deterministic Turing machine is also the length of this accepting path. Not the sum of the lengths of all this.

(Refer Slide Time: 10:30)



2. $C_i \to C_{i+1}$ is a valid move for each i

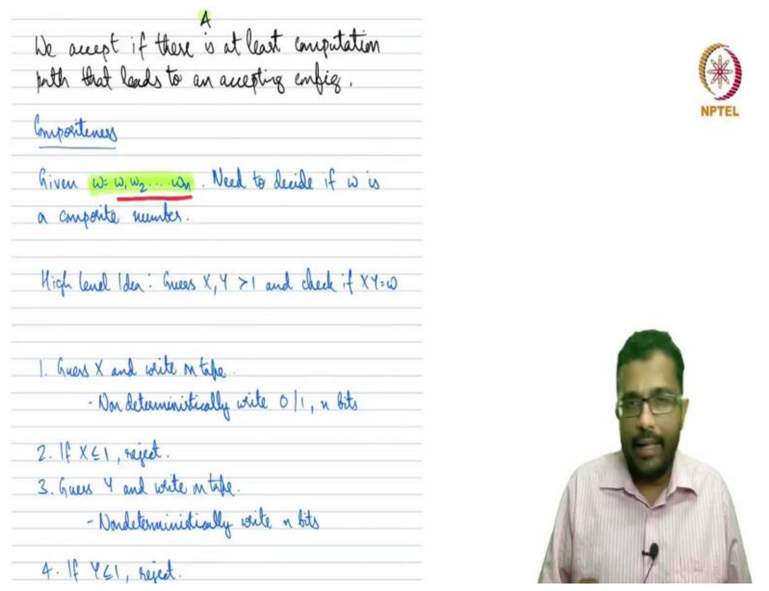
3. C_k is an accepting configuration.

We accept if there is at least computation path that leads to an accepting config.

Compositeness

Given $w = w_1 w_2 \ldots w_n$. Need to decide if w is a composite number.

High level Idea: Guess $X, Y > 1$ and check if $XY = w$



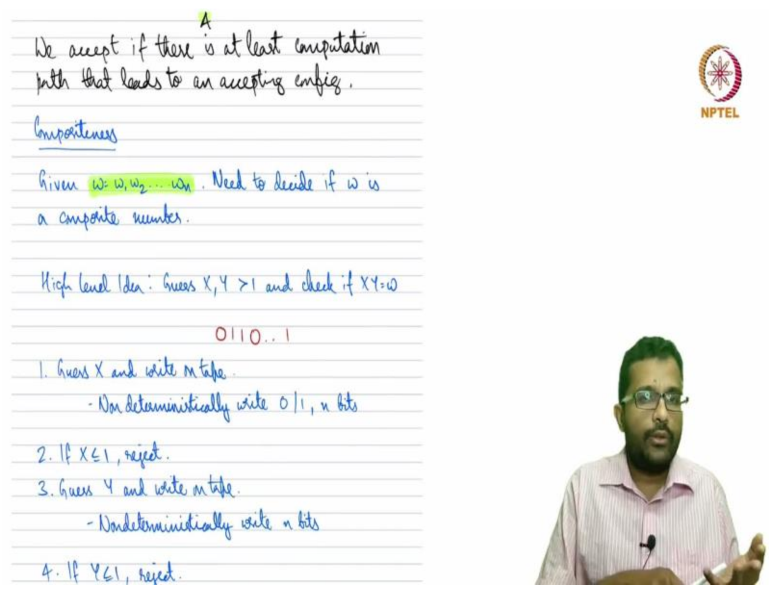
We accept if there is at least computation path that leads to an accepting config.

Compositeness

Given $w = w_1 w_2 \ldots w_n$. Need to decide if w is a composite number.

High level Idea: Guess $X, Y > 1$ and check if $XY = w$

1. Guess X and write on tape.
   - Non deterministically write 0/1, n bits
2. If $X \leq 1$, reject.
3. Guess Y and write on tape.
   - Nondeterministically write n bits
4. If $Y \leq 1$, reject.



We accept if there is at least computation path that leads to an accepting config.

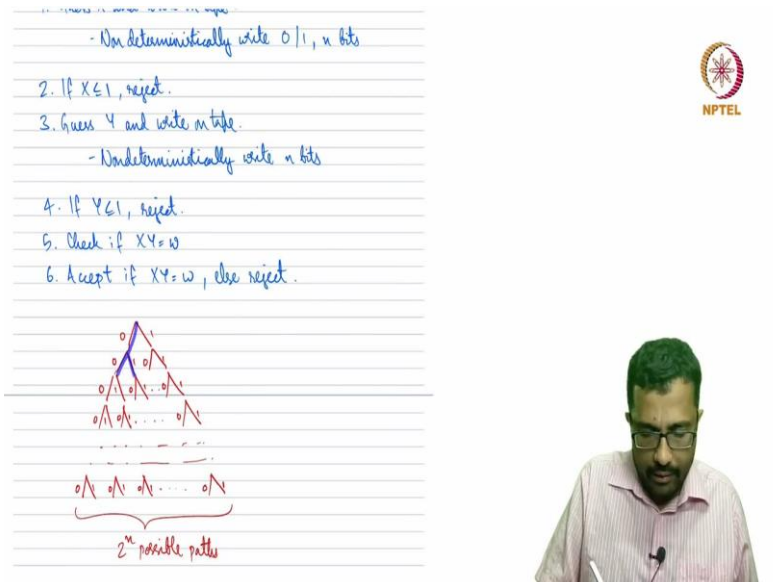
Compositeness

Given $w = w_1 w_2 \ldots w_n$. Need to decide if w is a composite number.

High level Idea: Guess $X, Y > 1$ and check if $XY = w$

0110..1

1. Guess X and write on tape.
   - Non deterministically write 0/1, n bits
2. If $X \leq 1$, reject.
3. Guess Y and write on tape.
   - Nondeterministically write n bits
4. If $Y \leq 1$, reject.



- Non deterministically write 0|1, n bits

2. If $X \leq 1$, reject.
3. Guess Y and write on tape.
  - Nondeterministically write n bits
4. If $Y \leq 1$, reject.
5. Check if $XY = w$
6. Accept if $XY = w$, else reject.

So, maybe some examples will make this clearer. Let us see one example. So, in this example, we are testing for compositeness, meaning you are given an integer, a number, we want to see whether it is composite. So, the given number is given number is w. And we want to see whether w is a composite number, composite meaning it is not prime.

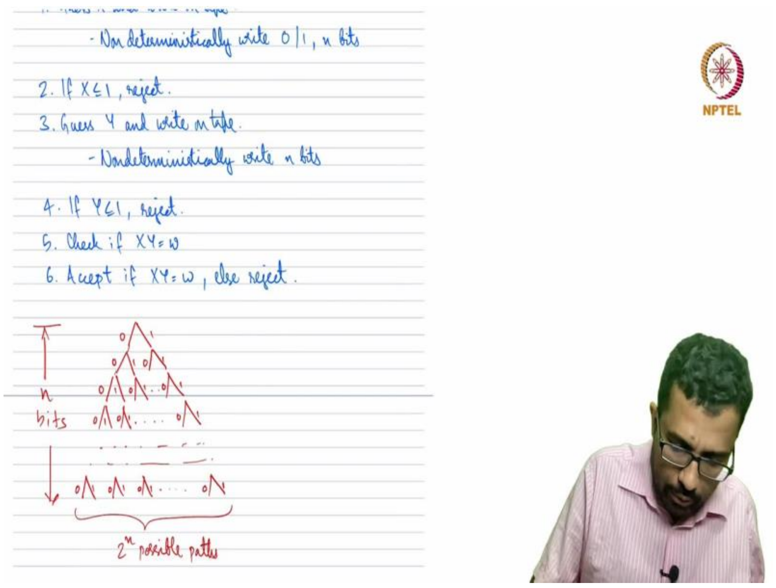
So, for instance, 20 is a composite number because it can be written as the product of two, 10×2 or 5×4. So, if it is a prime number, let us say 29, the only possible way you can multiply it, write it as a product is 29 times 1. Composite means not prime. So, the idea that we do here is this. So, you have your given w. You are just going to guess 2 numbers X and Y and check whether the product of X and Y is w.

So, if the product is w, if neither of X and Y is 1, then we know that w is a composite number because we are finding a non-trivial factorization of w. So, what we do is this, so how do we actually guess X and Y? So, suppose w, so the way w is given it has n bits. So w_1 to w_n. n bits. So first we are going to write X on tape.

So since the length of w is n, we are going to write an n bit number for X. So it could be anything. So you decide whether so you decide like this, you first write 0 or 1, let us say you write 0, then you write 1, then you write 1, then you write 0, then something like that, you write some n bits. So there are 2 possibilities, each bit you could write 0, or you could write 1. So, it is like this.

So, you are traversing through this computation path. So, it is like you first go left or right, 2 possibilities, and for each of these 2 possible there are 2 further possibilities. So, if you go left, then you could go left or right. If you go left, right, then you could again go left, right and so on. So there are multiple options.

(Refer Slide Time: 12:49)

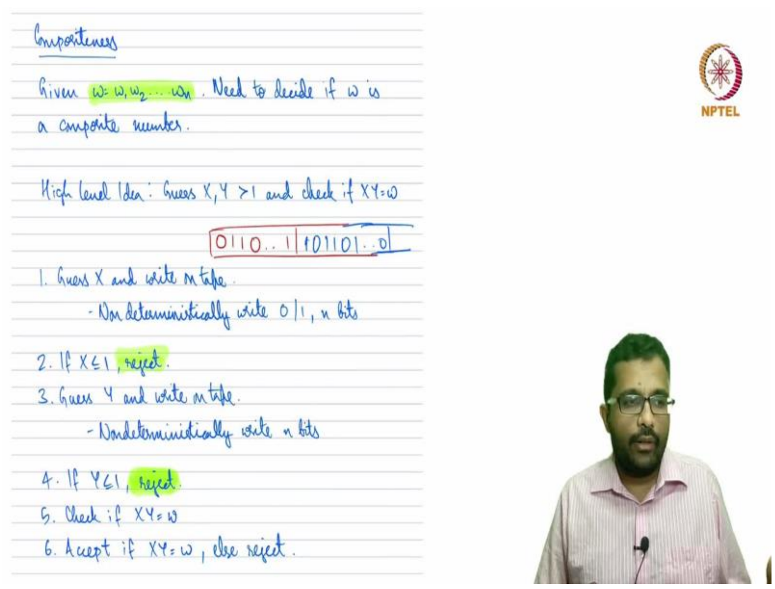


- Non deterministically write 0|1, n bits

2. If $X \leq 1$, reject.
3. Guess Y and write on tape.
   - Nondeterminidtically write n bits
4. If $Y \leq 1$, reject.
5. Check if $XY = w$
6. Accept if $XY = w$, else reject.

n bits

2^n possible paths



Compositeness

Given $w = w_1 w_2 \dots w_n$. Need to decide if w is a composite number.

High level Idea: Guess $X, Y > 1$ and check if $XY = w$

$\boxed{0110..1 \mid 101101..0}$

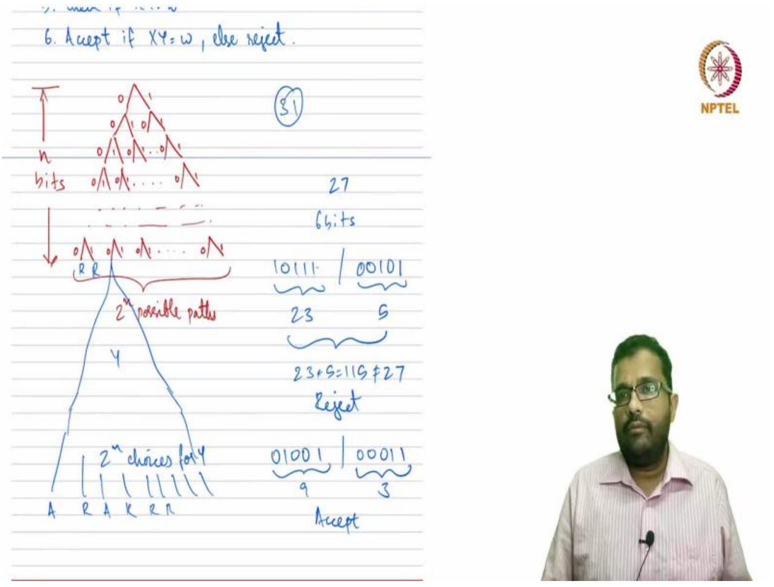
1. Guess X and write on tape.
   - Non deterministically write 0|1, n bits
2. If $X \leq 1$, reject.
3. Guess Y and write on tape.
   - Nondeterminidtically write n bits
4. If $Y \leq 1$, reject.
5. Check if $XY = w$
6. Accept if $XY = w$, else reject.

So, assuming this is n bits. This tree represents 2^n possibilities. So, you take the tape. So this is what you are doing. You take the tape and you first write X which is n bits, then check whether the number that you wrote is strictly greater than 1 or you check whether it is 1 or 0. If it is 1 or 0, then you are not going to get a non-trivial factorization.

So, then you reject. So, it is okay if you reject because all you need is one accepting computation and then you write a Y again you write some numbers, which is 1 0 1 1 1 something. And then again, you check whether that number is strictly greater than 1, if it is strictly greater than 1, you continue if it is not strictly greater than 1, meaning if it is 1 or less, you reject. So that is what I have seen here, if x is 1 or less you reject, if y is 1 or less you reject.

So, you write, you guess x you guess y. So guessing, by the word guess I mean non-deterministic choice. So, you could either write 01, then again write 01, and then again write 01, like that n times and then repeat it n times again for y. So, there are n choices made for x, n choices made for y. So basically, you are guessing all the possible 2^n paths of x. And all the possible 2^n paths for y.

(Refer Slide Time: 14:45)



So, what I meant earlier is that this path, the all 0 path that comes to the right side, this will be rejected because this gives us 0, this is not going to result in a valid computation. Even the one that goes 0 0 0 0 0 finally 1 that is also reject because that is also, we want something more than 1. But this is just x. Now for each of these choices of x, now, there will be another tree. So, let me just take one possibility and another tree. So, this is the choice of y and there will be 2^n possible choices for y. So, I am just writing the 2^n choices like this.

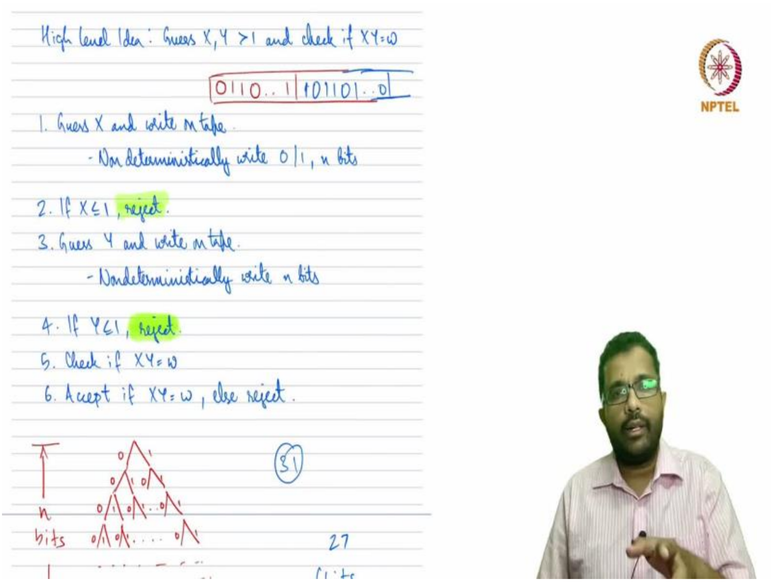
So, now finally, for each of these choices, you can check whether x multiplied by y is equal to the w so. Then if it is equal you accept, if it is not equal you reject. So, like that. So, just to give a concrete example, let us say you take the number 27. So, 27 we need 27 is less than 32. So, you need 6 bits to represent 27. 6 bits. So, then you guess two 6 bit numbers. Let us say, 1 0 1 1 0 1 1 0 1 something and then and then 0 0 0 1 0 1 or something.

So, this number if you check what is it let us see 16, 4 2 16 8 4 2 1 I am sorry, we do not need 6 bits, we only need 5 bits. So, let me just write this. So, now 16 8 4 2 1, 16 8 4 2 1. So, 16 plus 7, this is 23 and this is 101. This is 5 and so 23 multiplied by 5 is not 27, 23 multiplied by 5 is 115. So, this path will lead to 23 multiplied by 5 is 115, which is not 27. So, this will be rejected. Whereas, if you guess let us say 0 1 0 0 1 and it is 0 0 0 1 1 this is 9 and this is 3 and this will be accepted.

So, we will make many guesses which are wrong. But since 27 is a composite number, there will be at least 1 set of guesses, there could be more, 27 can be written as 3 times 9 or 9 times 3, two different ways. So, there will be at least one set of guesses which will certify that it is a composite number. And even if there is one set of guesses you will see that it is accepted. But however, if you take let us say 31. 31 there are no ways to factorize 31, other than the non-trivial factorization, you can only write it as 1 times 31.

So whatever path you take, whatever numbers you guess, you are not going to get 31, the only possibility you get is 1 times 31 which will anyway be rejected. So, this all paths will reject for 31. Whereas for 27, there will be at least 1 accepting path. So that is how you use non-determinism to decide whether a number is composite or not. Once again, it is not like the machine is serially trying to do all these different computation paths. You have to think of the machine just magically trying out everything and figuring the correct path if it exists, all it needs is one path that leads to an accepting computation, accepting state.

(Refer Slide Time: 19:30)

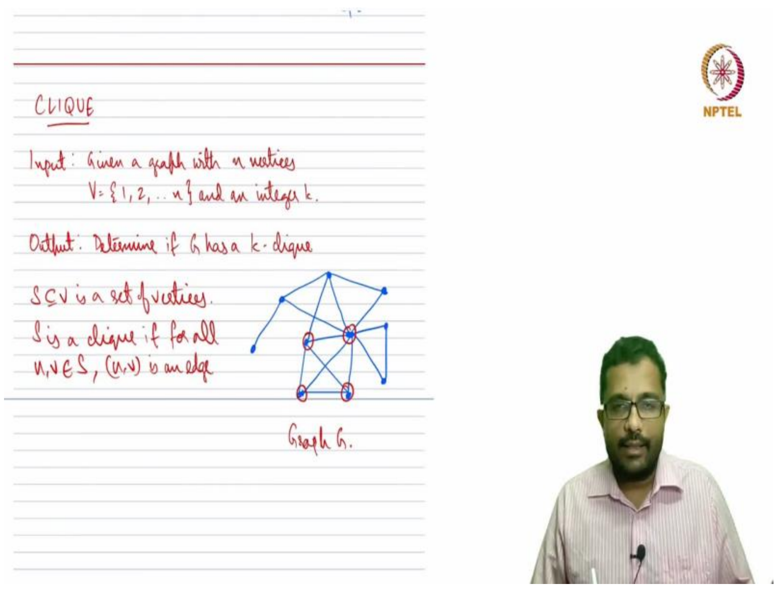

High level Idea: Guess X, Y > 1 and check if XY = w

0110..1 | 101101..0

1. Guess X and write on tape.
   - Non deterministically write 0|1, n bits
2. If X ≤ 1, reject.
3. Guess Y and write on tape.
   - Nondeterministically write n bits
4. If Y ≤ 1, reject.
5. Check if XY = w
6. Accept if XY = w, else reject.

Once again, it guesses X it guesses Y. If either one is 1 or less it rejects and then it tries to multiply. If the product is equal to w you accept otherwise you reject. So if w is indeed composite, there will at least one guess for X and Y which will for which we will get the w to be the product and then you will accept. So, that is how you would test whether a number is composite. This can be done for using deterministic computation also, but since the objective is non-deterministic computation, we are presenting it in a non-deterministic fashion.

(Refer Slide Time: 20:24)



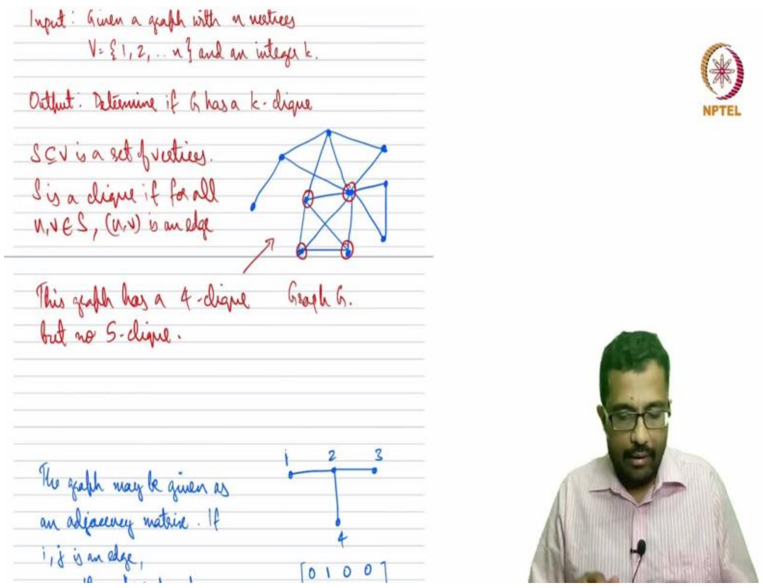
CLIQUE

Input: Given a graph with n vertices $V = \{1, 2, \ldots n\}$ and an integer k.

Output: Determine if G has a k-clique

$S \subseteq V$ is a set of vertices.
S is a clique if for all $u, v \in S$, (u,v) is an edge

Graph G.



Input: Given a graph with n vertices $V = \{1, 2, \ldots n\}$ and an integer k.

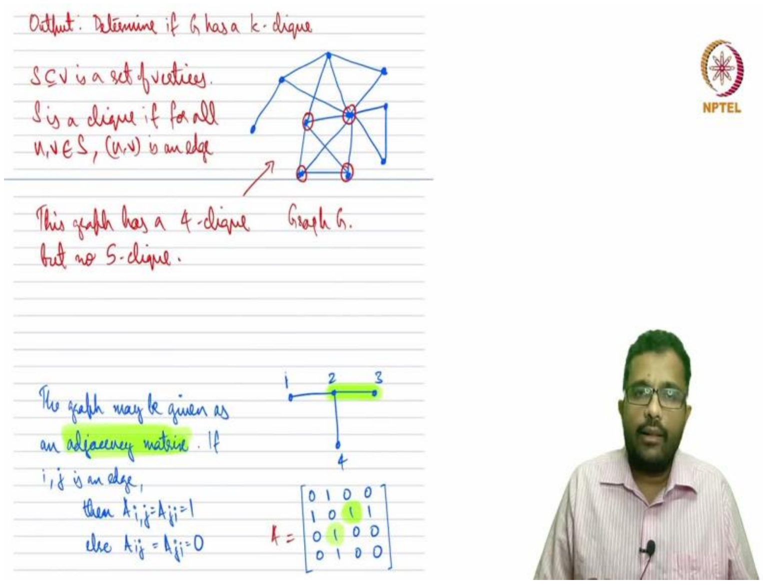
Output: Determine if G has a k-clique

$S \subseteq V$ is a set of vertices.
S is a clique if for all $u, v \in S$, (u,v) is an edge

This graph has a 4-clique but no 5-clique.

Graph G.

The graph may be given as an adjacency matrix. If i,j is an edge,



Output: Determine if G has a k-clique

$S \subseteq V$ is a set of vertices.
S is a clique if for all $u, v \in S$, (u,v) is an edge

This graph has a 4-clique but no 5-clique.

Graph G.

The graph may be given as an adjacency matrix. If i,j is an edge,
then $A_{ij} = A_{ji} = 1$
else $A_{ij} = A_{ji} = 0$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

So, the next thing is a clique. So what is the clique? So, we have a graph. So, this is a graph. You have these, the ones I have drawn here. You have vertices which are these dots and you have edges which are joining these 2 dots. So, this is an edge, this is an edge, this is an edge so, an edge connects 2 vertices and we say a set of vertices is a clique if all the vertices are adjacent meaning all the vertices there is an edge connecting them. So, S is a set of vertices S is a clique if for all $u, v \in S,\ (u, v)$ is an edge.

So, for instance I just draw one example, we will just draw one example. So, these 3 vertices form a clique because if you see this, there are 3 vertices and they form a triangle of sorts. Meaning, any two that you take, they are connected by an edge.

Another example is these 4 vertices. This also forms a clique because any 2 that you take, there is an edge between them. So, the goal is you are given a graph. Let us say this is the graph G which has n vertices and a number k. The question is does this graph have a k clique? So, in this if this is the graph, this has a 4 clique which is marked, but it does not have a 5 clique. At least I think it does not have it, but you can check it.

So, if you are given this graph and the number 4 you can answer that it does not have a or it does have a 4 clique, but if you are given the number 5 or number 6 or any higher number you can say there is no 5 clique or 6 clique. So, that is the goal here. So, this graph has a 4 clique meaning a clique of size 4 but no 5 clique if it does not have a 5 clique, there is no scope of having a 6 clique or a bigger clique.

So, this graph does not have a 5 clique. So, 4 is the largest size clique. So, of course, you can test for it deterministically meaning if there are n vertices, you can check in let us say case 4, you can check all possible subsets of size 4 which is roughly ${}^{n}C_4$ subsets and then test in that set of 4 vertices you can check whether all the edges are present. But this will take ${}^{n}C_4$ time which is kind of high. $O({}^{n}C_4)$ times which is kind of high.

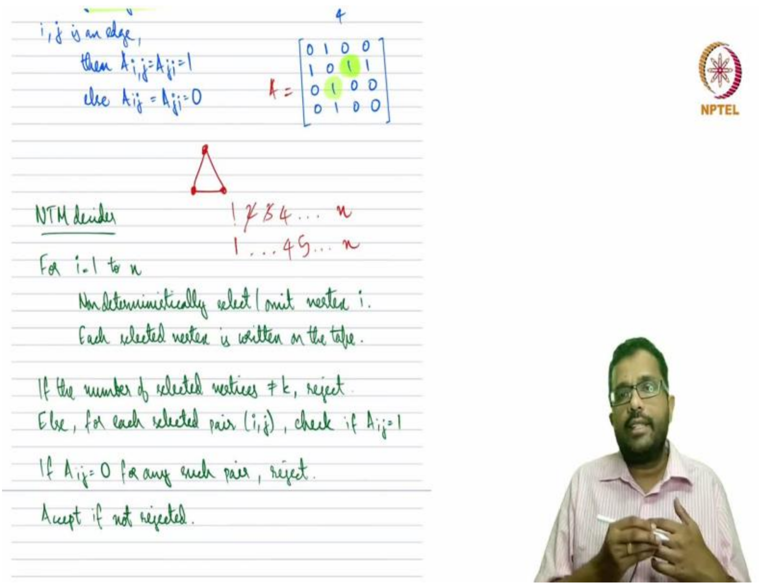
So, now, non-deterministically you can do it faster. Although it will also help to illustrate the notion of non-determinism. So, before that I just want to explain how a graph is represented. You cannot give a graph as a picture to a computer. So, if you have taken a data structures course, you might have seen these terminologies, one of the standard ways to represent a matrix represent a graph to a computer in a computer program is by an adjacency matrix.

So, if a graph has n vertices, you give an $n \times n$ matrix, the matrix has only 0,1 entries. For example, there is a graph here which has 4 vertices and the adjacency matrix A is written here.

Since the graph has 4 vertices, the adjacency matrix is of size 4 × 4. And basically, you put, let us say this entry, maybe I will just highlight something. So, this entry is one, because it is second row and third column. This is one because of this edge.

This is because of the edge connecting vertex 2 and vertex 3. And the same entry, the entry 3, 2 also, because the third vertex is connected to the second vertex. Because this is an undirected graph, which means an edge from 2 to 3 is the same as an edge from 3 to 2. And that is why this is also a symmetric matrix. So, if you see, if you try to take a transpose, it remains the same. This is how you represent a matrix. So, it has 4 vertices, so it is a 4 by 4 matrix, and you can verify that the other edges are encoded correctly. So how do you determine using a non-deterministic Turing machine if a given graph has a k clique?

(Refer Slide Time: 26:29)



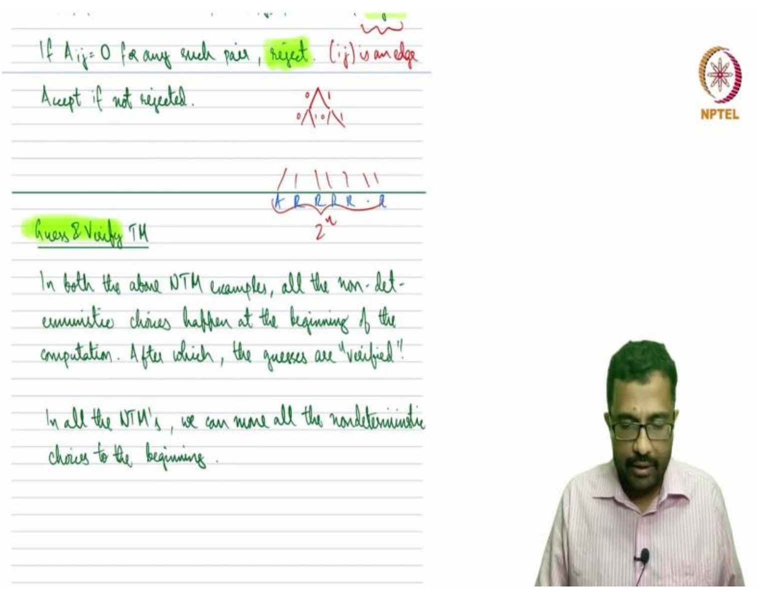
i, j is an edge,
    then $A_{i,j} = A_{ji} = 1$
    else $A_{ij} = A_{ji} = 0$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

NTM decider

```
For i = 1 to n
    Nondeterministically select / omit vertex i.
    Each selected vertex is written on the tape.

If the number of selected vertices ≠ k, reject.
Else, for each selected pair (i, j), check if A_ij = 1
If A_ij = 0 for any such pair, reject.
Accept if not rejected.
```



```
If A_ij = 0 for any such pair, reject.   (i, j) is an edge
Accept if not rejected.
```

Guess & Verify TM

In both the above NTM examples, all the non-deterministic choices happen at the beginning of the computation. After which, the guesses are "verified".

In all the NTM's, we can move all the nondeterministic choices to the beginning.

So, what we do is the following. We non-deterministically choose a subset. So how do we choose a subset? Let us say there are n vertices like 1 2 3 4, and so on. You decide non-deterministically to select 1 or not, maybe 1 is selected, maybe 2 is not selected, so you do not select it, maybe 3 is also not selected, maybe then 4 is selected, maybe then 5 is selected and so on, you select some vertices.

So, 1, 4, 5 something and then you check whether this set of vertices that you have selected is a subset of vertices, whether this forms a clique, this forms a k clique. So first, you non-deterministically, select a set a subset of vertices. So, you go 1, you include or exclude, 2, include or exclude, that is what I have written here. You would non-deterministically select or omit a vertex i and you write each of these vertices that are selected on the tape.

So first you so you want to k clique if the number of selected vertices is not k, you can automatically reject because you are looking for exactly k vertices. If it is equal to k meaning if you have if you have non-deterministically chosen a set of size k, then for that chosen set, whether you check whether it is a clique, meaning if there are 3 vertices they have selected, let us say k is 3, you check whether for all the possible pairs of vertices, there is an edge between them.

So, if there are 3 vertices, you want a triangle kind of thing. And that is it, so you non-deterministically select or omit each vertex. So again, first you are selecting or omitting vertex 1, so selecting maybe omitting is 0 selecting is 1. Vertex 2, again you are doing this, 01, and finally, you get to 2^n possibilities, because there are 2^n vertices, out of which many will not be subsets of size k, because 2^n means all possible subsets, but so then they will automatically be rejected. And for the subsets of size k you check whether it forms a k clique or not.

If it forms is k clique, it will be accepted. So, if it is a subset of size k, you check whether for all the pairs of vertices, if there is an edge, so $A_{ij} = 1$ means edge or rather (i, j) is an edge. If even one of the pairs (i, j) is not an edge, you reject. That is what I am saying here. So, if you do not reject you accept meaning, you accept if only 2 things are satisfied. First of all, if the number of all selected vertices is k and all the selected vertices, you take any pair of them there is an edge between them.

So, you first have these 2^n possibilities and then you check whether for each of these possibilities whether it is a k clique or not. So, some of them will be accept, some of them will be accept, some of them will be reject. If the graph has a k clique, then there will be at least one of these guesses, because the guesses you are trying together span all the possibilities, at least one of them will correctly guess the k clique that is there. But if none of the subsets of size k form a clique then all of them will be rejected.

So, if they are not of size k, they will be immediately rejected. If they are of size k they will be tested, but since none of the subsets of size k are going to be a clique, they will automatically be rejected. So, if there is a subset of size k, which is a clique, it will accept, which is what we want. If there is no such subset, they will be rejected, all paths will be rejected. So that is the… that is how we non-deterministically test whether the given graph is a clique.

Here all the non-deterministic choices are made in parallel. It is not even parallel and in fact, non-determinism is kind of an abstract concept where if there is a path that needs to accept you just are able to find it. So, it is not like we are searching this path and then this path and then this path, it just automatically you just select the correct path if it is there. So, time taken is to pick the make the choice and then verify it.

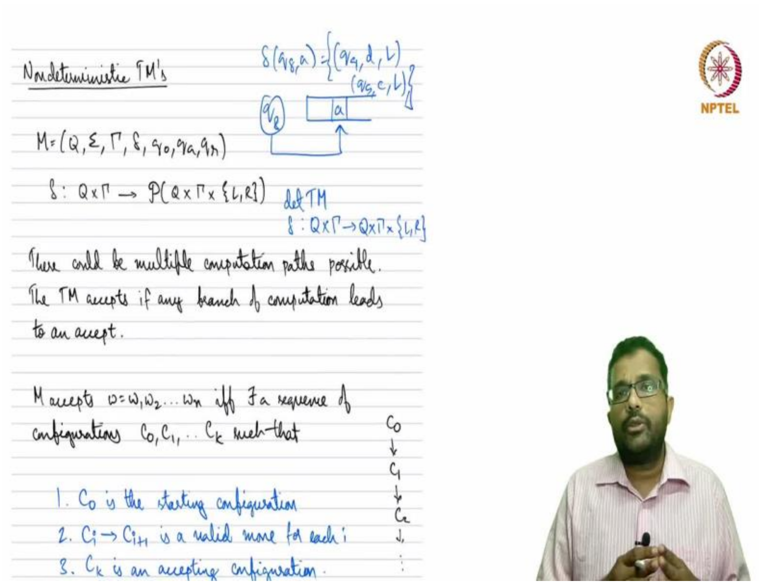
So, one point I want to quickly mention about the 2 non-deterministic Turing, non-deterministic choices or problems that we did. In the case of compositeness, we first get to non-deterministically choose X then non-deterministically choose Y followed by a verification or a checking whether X multiplied by Y is equal to the number and if X and Y are greater than 1. Even here, clique also is something similar. We non-deterministically choose a subset followed by a verification where we check whether the subset is indeed a k subset, k sized subset then checking whether it formed a clique.

So, in both the cases, we did some guessing, meaning non-deterministic choices, followed by checking whether these choices lead to the desired answer. The point is that this is kind of a model where you guess something or non-deterministically choose something then you verify. And the point is that for any non-deterministic computer, non-deterministic Turing machine we can do this. We can move all the non-deterministic choices right to the beginning. So, there will be a beginning part where we make a bunch of non-deterministic choices followed by a deterministic verification.

So, once I choose a subset in the clique problem, which is non-deterministically chosen, once that subset is there, then the rest of the verification is entirely deterministic. I need to check whether k vertices are selected, whether the number of selected vertices is equal to k which is a completely deterministic process. There is no choice or choosing happening. And once I am given a subset, I am just checking whether each pair of vertices form an edge between them. That is also a deterministic process.

So, in any non-deterministic Turing machine, we can reorganize the operation in such a way that all the non-determinism happens right at the beginning, such a non-deterministic Turing machine is also sometimes called a guess and verify Turing machine. So, this is something that I wanted to say. So, guess and verify means all the guesses at the beginning followed by the verification. So, again guesses are another word for a non-deterministic choice. And I think that completes this lecture, lecture number 30.

(Refer Slide Time: 34:23)



Nondeterministic TM's

$\delta(q_8, a) = \{(q_4, d, L), (q_5, c, L)\}$

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$

det TM

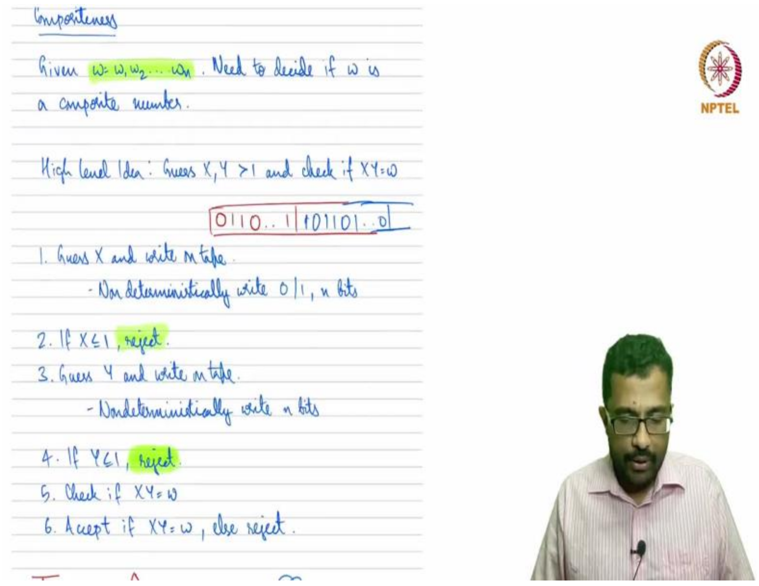
$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

There could be multiple computation paths possible. The TM accepts if any branch of computation leads to an accept.

M accepts $w = w_1 w_2 \dots w_n$ iff $\exists$ a sequence of configurations $C_0, C_1, \dots C_k$ such that

1. C_0 is the starting configuration
2. $C_i \rightarrow C_{i+1}$ is a valid move for each i
3. C_k is an accepting configuration.

$C_0 \downarrow C_1 \downarrow C_2 \downarrow \vdots$



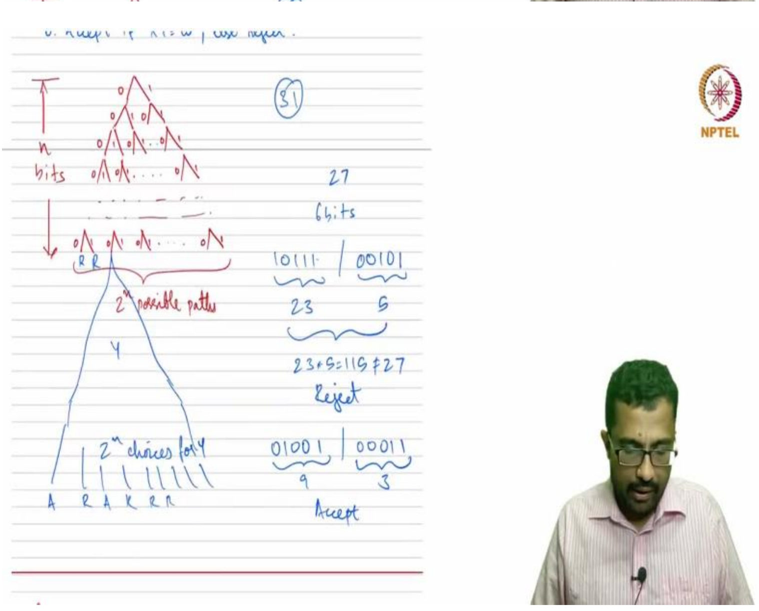
Compositeness

Given $w = w_1 w_2 \dots w_n$. Need to decide if w is a composite number.

High level Idea: Guess X, Y > 1 and check if XY = w

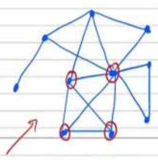
0110..1|101101..0

1. Guess X and write on tape.
   - Non deterministically write 0/1, n bits
2. If $X \leq 1$, reject.
3. Guess Y and write on tape.
   - Nondeterministically write n bits
4. If $Y \leq 1$, reject.
5. Check if XY = w
6. Accept if XY = w, else reject.



Output: Determine if G has a k-clique

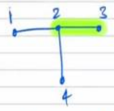
$S \subseteq V$ is a set of vertices.
S is a clique if for all $u,v \in S$, (u,v) is an edge



This graph has a 4-clique but no 5-clique.

Graph G.

The graph may be given as an adjacency matrix. If i, j is an edge,
then $A_{i,j} = A_{ji} = 1$
else $A_{ij} = A_{ji} = 0$



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

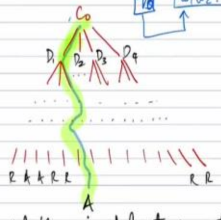
The TM accepts if any branch of computation leads to an accept.

M accepts $w = w_1 w_2 \dots w_n$ iff $\exists$ a sequence of configurations $C_0, C_1, \dots C_k$ such that

1. C_0 is the starting configuration
2. $C_i \rightarrow C_{i+1}$ is a valid move for each i
3. C_k is an accepting configuration.

$$\begin{tikzcd} C_0 \arrow[d] \\ C_1 \arrow[d] \\ C_2 \arrow[d] \\ \vdots \\ C_k \end{tikzcd}$$

q_0
$w_1 w_2 \dots w_n$

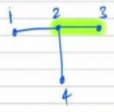


We accept if there is at least computation

This graph has a 4-clique but no 5-clique.

Graph G.

The graph may be given as an adjacency matrix. If i, j is an edge,
then $A_{i,j} = A_{ji} = 1$
else $A_{ij} = A_{ji} = 0$

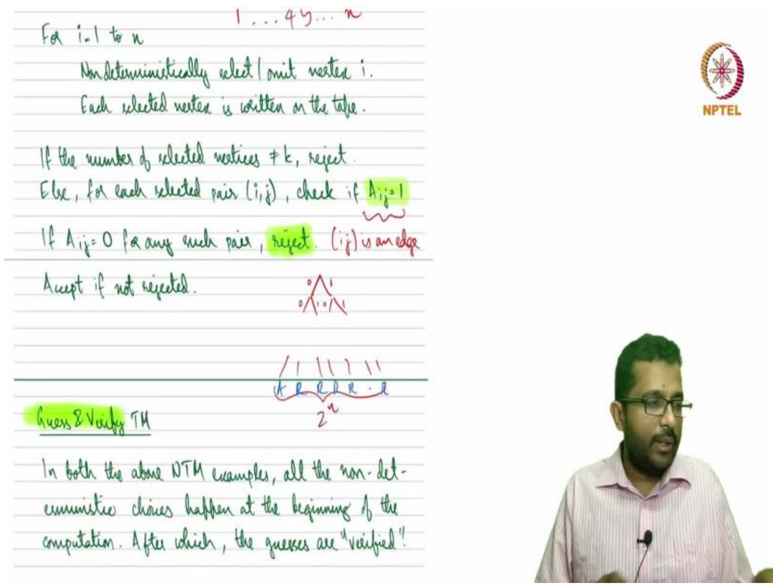


$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

NTM decider

1 ~~2~~ ~~3~~ 4 … n
1 … 4 5 … n

For i = 1 to n



For $i = 1$ to n $\quad 1 \ldots 4\,5 \ldots n$

Nondeterministically select / omit vertex i.

Each selected vertex is written on the tape.

If the number of selected vertices $\neq k$, reject.

Else, for each selected pair (i,j), check if $A_{ij} = 1$

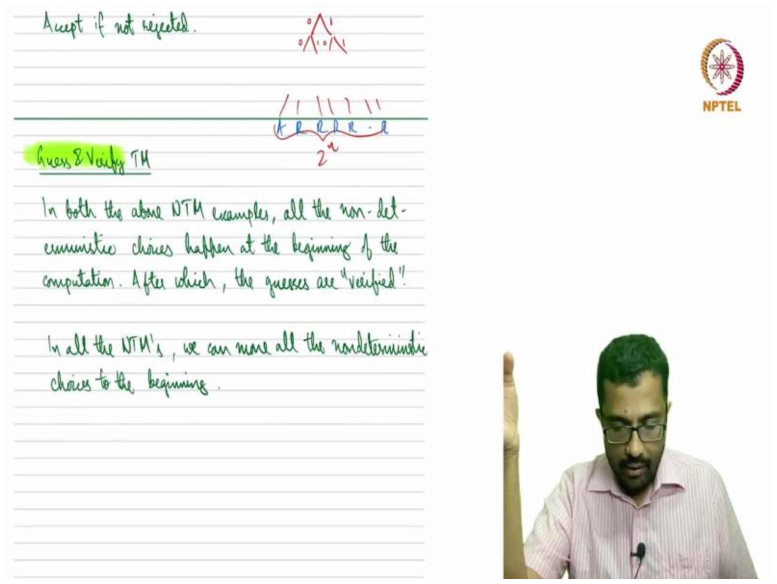
If $A_{ij} = 0$ for any such pair, reject. (i,j) is an edge

Accept if not rejected.

2^n

Guess & Verify TM

In both the above NTM examples, all the non-deterministic choices happen at the beginning of the computation. After which, the guesses are "verified".



Accept if not rejected.

2^n

Guess & Verify TM

In both the above NTM examples, all the non-deterministic choices happen at the beginning of the computation. After which, the guesses are "verified".

In all the NTM's, we can move all the nondeterministic choices to the beginning.

So, we discussed non-deterministic Turing machines, we saw examples of compositeness and k clique. We also explained what are acceptance conditions. So, you accept a string given as input if at least one of the many non-deterministic paths leads you to accept it. If there is at least one accepting computation path out of the many non-deterministic choices you accept. If all the paths are rejecting, then you reject.

So, this works for both k clique and compositeness. So, one may wonder because we saw non-deterministic finite automata which seemed to be more powerful because you could make multiple choices, multiple movements, but eventually it became equivalent in power to deterministic finite automata because the class of language captured by both of them were the regular languages.

Similarly, one can ask here. We saw deterministic Turing machine and non-deterministic Turing machine is clearly at least as powerful because we can do without making all these choices. We could always have exactly one choice. We could just have one possible option for all the state and symbol pairs, in which case it is the same as a deterministic Turing machine.

So, are non-deterministic Turing machines more powerful? Do they recognize some language that deterministic Turing machines do not recognize? Do they decide some language that deterministic Turing machines do not decide? So, this is something that is a natural question at this point. And the answer is no. In fact, whatever can be decided or recognized by a non-deterministic Turing machine can also be decided or recognized by a deterministic Turing machine. So, that particular thing we will see in the next lecture. As far as lecture 30 is concerned. That is it. And so, see you in the next lecture. Thanks.