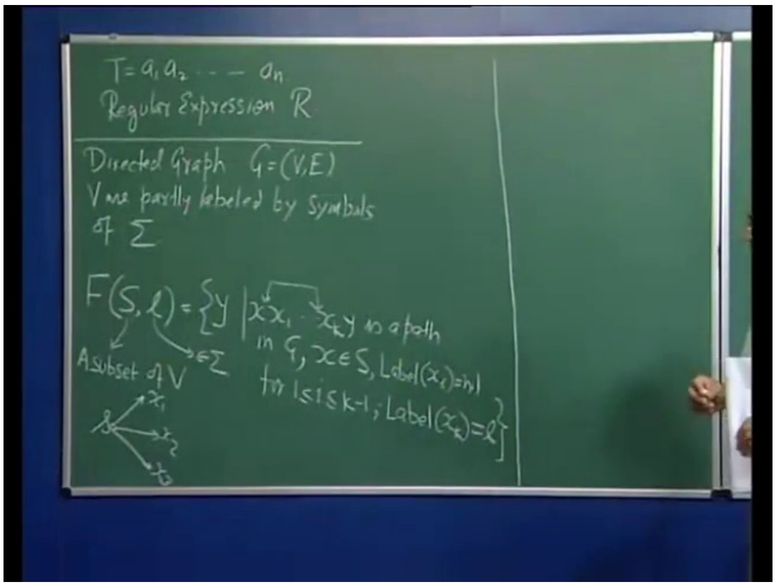
Computer Algorithms - 2

Prof. Dr. Shashank K. Metha
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 16
NFA Simulation

(Refer Slide Time: 00:25)



$T = a_1 a_2 \cdots a_n$
Regular Expression R

Directed Graph $G = (V, E)$
V are partly labeled by symbols of Σ

$F(S, \ell) = \{ y \mid x x_1 \cdots x_k y$ is a path in G, $x \in S$, Label$(x_i) = a_i$ for $1 \le i \le k-1$; Label$(x_k) = \ell \}$

A subset of V
$\in \Sigma$

$$\begin{tikzcd} & x_1 \\ S \arrow[ur] \arrow[r] \arrow[dr] & x_2 \\ & x_3 \end{tikzcd}$$

Hello, so today we are going to take a little bit abstract view of our string matching problem, recalled that we used to have text string a 1, a 2, a n. Now, in place of a pattern string I am going have a regular expression R. Now, you must be familiar with a regular expression, this is equivalent to regular languages, every regular language has a corresponding regular expression and vice versa. Our problem now is to determine whether there exists some index i, such that starting from a i there exists a string, a substring of t starting at a i which belongs to the language of R.

Now, we know normally that there is automaton we can build, from a regular expression and then they can input the string starting from location i and see where is the final state? That would be a simple answer to this, but we also know that in general, the number of states may be exponential in the size of R. If we want to design a deterministic auto meter and that would not be very efficient.

So, what our role is that we design a non-deterministic finite state automaton a suitable way, such that searching for a member starting at a i in the language can be done in polynomial time. So, what I am going to do first is device a algorithm for a specific task in the context of graph and then we move on to design an automaton and solve the problem.

So, right now my problem moves to a issue in a graph, we have directed graph G, G and the vertices of V are either unlabeled or they are labelled from a set of labels sigma. So, V are partly labelled by symbols of sigma. Now, our task is to compute the following function let us denote that by F S and a label l. So, S is a subset a subset of V, set of sum of the vertices, l is a specific label belongs to sigma. I would like to compute some of this state sum of the states, sum of the vertices of the graph which are given by the following.

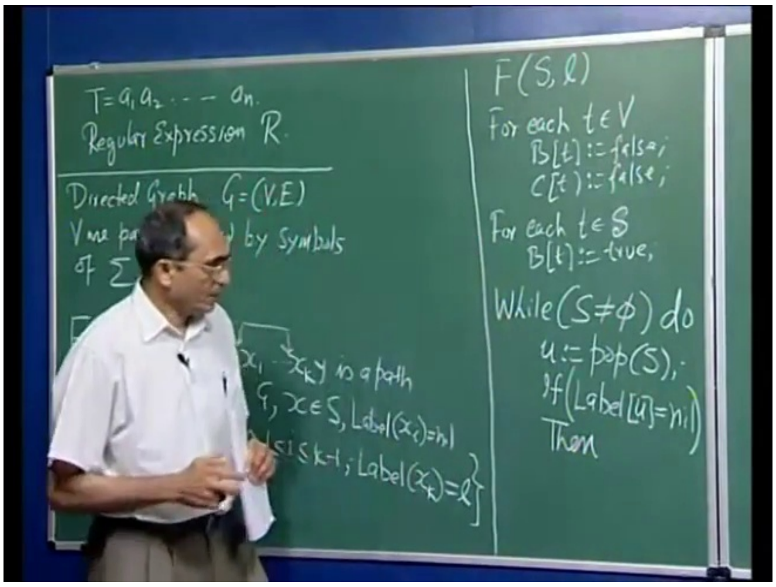
So, I would like to compute a vertex the set of vertices would be y, where there is a path from x, x 1, x k y is a path in g x belongs to S. So, this is a path in G x belongs to S. So, this is a path that begins at some vertex in S, further we want that all of these vertices x 1 through x k, they have no label. So, label of x i is nil, for i between 1 and k minus 1 up to this one, further the label of x k is l.

So, we begin at some vertex of S, we walk along unlabeled vertices of course, this is a directed path until we hit a vertex which has label l. Then the following that there is some vertex to which it can go from x to y such vertices are what we want to compute. Now, to compute such vertices we effectively can initiate a search process starting with this set of vertices. Now, recall that in a search process such as that the first subset, what we did was? We began with a vertex, say s small s and we went along the edges wherever it took us and then those vertices. So, we had an s here and it went to say sum of these vertices say x 1, x2, x 3.

What we did at that time is, we said these vertices in a data structure, in one case it was this stack in another it was q. Now, it is not important what data structure here you, but we save them and then we pick a vertex from that data structure. Then proceed the expansion, resume the expansion from these points. So, we will continue to do that in the mean time that data structure has several vertices, those are the vertices from where we have yet to perform the expansion.

So, we have essentially done a similar kind of search. What we will do is, we will continue to expand as long as the vertex has no label, because that is what this is effective. Until we hit a vertex with label l, if we hit a vertex with another label. We will terminate that part, if we hit upon a vertex with label l, then the following vertices is everything that we can reach will be all valid members of this set. So, let us just capture that in the in this algorithm, so let us say we have F to which we are given a subset of vertices S and a label l. I am going to use two data structures one in fact both of them are arrays of Boolean value for each t in V, we will set B t to be false and C t also will be false.

(Refer Slide Time: 08:30)


$T = a_1 a_2 \cdots a_n$.
Regular Expression R.
Directed Graph $G = (V, E)$
V are pa[illegible] by Symbols
of Σ
$x_1 \cdots x_k$ y is a path
G, $x \in S$, $Label(x_i) = nil$
$\leq i \leq k-1$; $Label(x_k) = l$ }

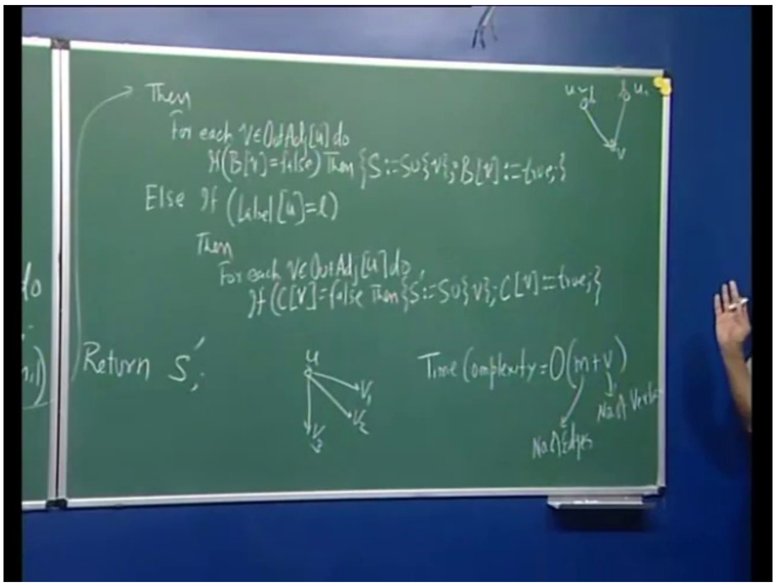
```
F(S, l)
For each t ∈ V
    B[t] := false;
    C[t] := false;
For each t ∈ S
    B[t] := true;
While (S ≠ φ) do
    u := pop(S);
    If (Label[u] = nil)
    Then
```

What this indicates is that this vertex has not yet been incorporated into the data structure, from where we are going to take vertices and expand and C t says that t has not yet been output into this collection is not been. We do not want to duplicate these efforts that is why we are using such Boolean arrays. Now, so far we have initialize 2 n t, but we also know that to begin with we will put all these in fact, we use the same set s as the data structure into which vertices will go for expansion. So, initially the value of S, this S itself, so I am not going to use a new data structure, but say for each t in V, in S, in S, we will set the to be true.

So, all of these have been accepted for expansion, because we are going to start, we are supposed to start at any vertex in this, so all of them are put into S. Next, we are going to continue the expansion until S becomes empty. So, we will say while S is not empty do. We will keep a vertex 1 S, we will look at its label, if the label is nil. Then we will look at its neighbour and decide whether they should be put back into S for further expansion and if the label is l. Then we will decide the adjacent vertices should be put into the data structure as prime the final output set I will call this set as prime, so I will decide to put that into s prime, so let us just do that.

So, let us say we have u pop from S by this I mean I am extracting one vertex, any vertex from S and removing it from S and then check if label of u is nil. So, all right then so I am going to resume from here in that case. Then we are going to look at all the neighbours for each V in the out adjacency, which, this is a directed graph every vertex to which there is an arrow from u.

(Refer Slide Time: 12:21)



```
Then
   For each v ∈ OutAdj[u] do
      If (B[v] = false) Then {S := S ∪ {v}; B[v] := true;}
Else If (Label[u] = l)
   Then
      For each v ∈ OutAdj[u] do
         If (C[v] = false Then {S' := S ∪ {v}; C[v] := true;}
Return S';
```

So, I will say out adjacency of u, if V has already been incorporated in S and has been either expanded or waiting for being expanded, we do not have to do anything. So, we will say if B of V is false, that is to say it has not yet been appraised for expansion. Now, we have found it, we are going to then put this vertex in S. We will set the value of this vertex to be true, so future expansion will be possible from this.

Now, this was if the label was the nil, now else if the label of u. Now, the next thing we are going to check is if the label is l, in that case. I should have initialized this set S prime to be empty, because this is where I am going to put my output. So, in that case we will again look at all the neighbours for each V in the out adjacent t of u, we do the similar thing, but this time I am going to output that vertex.

So, if C of V, C of V is false which means V is not yet been output, but please note that we have found a path starting from some vertex of original S, where all the subsequent vertices has label nil, until we hit u which had label l and then we have neighbour of u V, which should be put into that set. So, we are setting them S prime is, S prime union V and we record this fact in the array V by setting this to true and that is about all. So, this is where the process will terminate and finally, we will output we will return S prime.

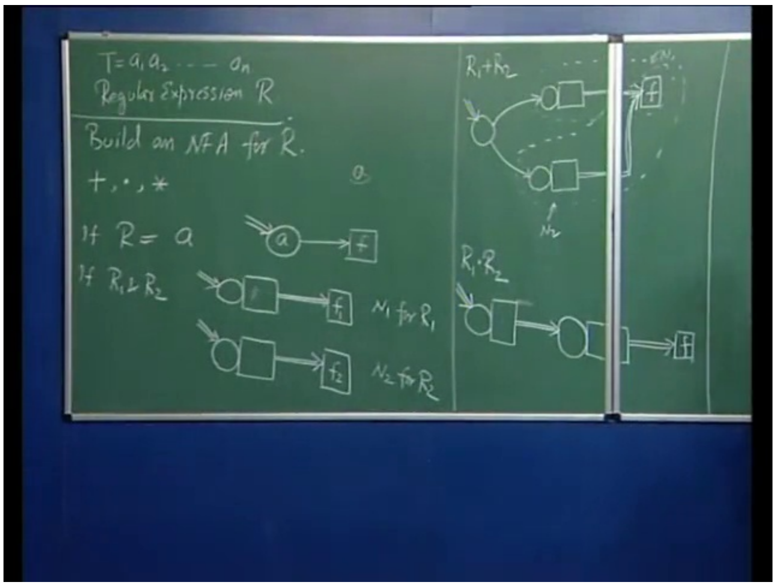
Now, once you see that this is exactly what this algorithm computes and you notice that this is really the same process of says, that we have seen in both depth first and depth personal. The cost can be very easily determined, we never expand this same vertex second time, once a vertex enters S it is eligible for expansion. Once it is expanded, we in fact the moment we have taken it into S, we have set this to two this Boolean value to true.

So, second time we have checked whether it is false, so that will no longer be true the second time. Hence, you will never expand the same vertex again and the starting from a vertex u, we are looking at its neighbours V 1, V 2, V 3 and so on. Since, we are not going to expand again, we are not going to visit this edge again. So, each edge can be visited at most once and we may not have any edges and we may have all the vertices in S. So, we have to actually initiate the process from every vertex in S, hence the total cost, the time complexity will be order m plus v, where m is the number of edges number of edges and this is member of vertices.

We also ensure that no vertex is put into output set S prime more than once by using this array C V. So, we are not going to duplicate any effort. So, suppose if we have a vertex V, which can be appraised from u 1 as well as u 2. When we were expanding u 1, where is there and it was eligible. So, it was this head label l, so at this one, this was eligible to be put in a string we did it.

Second time, when we come here this cost goes to this edge, but then they are not going to, we will find the label is false. The C label is false, hence we are not going to put this again into that array as planned, so no duplication. Hence, this is very easy to see the total cost is linear in the vertices and the edges of the graph. So, now let us come back to our original problem of matching a string with the language given by a regular expression. Now, what we will do is, we will specify a certain way of designing a non deterministic automaton for R. Then we will relate the cost of search in that with the problem that we have just solved.

(Refer Slide Time: 20:17)



So, let us say build an NFA for R, we notice that there are three basic operations so the operations are the plus (( )) and twin star, these are the three operations you have. So, we will do, we will describe this building process in a recursive fashion. So, if R is simply a single symbol a, then the automaton we are going to design, we will have one step a and then we will keep a separate state, which will be our final state. Of course, this arrow indicates this start state is this. So, this will take of this regular expression, there is only a string containing a, one single string that this recognizes.

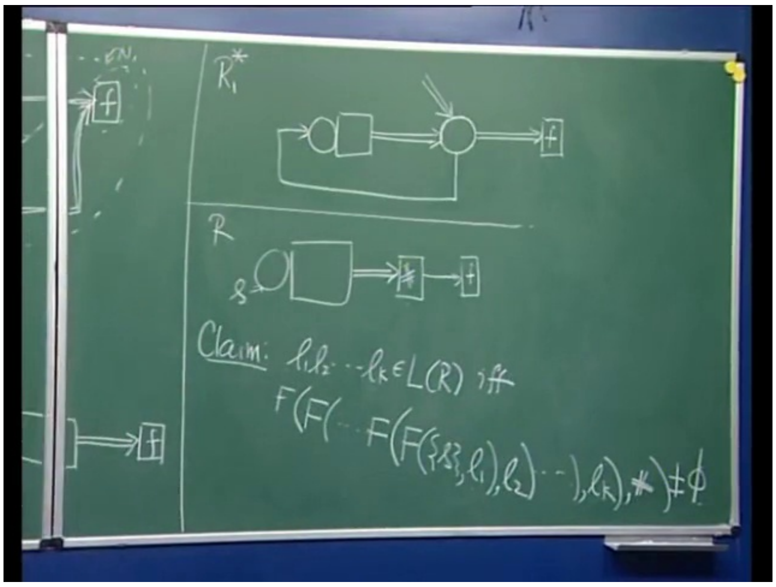
Now, suppose we have two regular expression R 1 and R 2. Let us symbolize the corresponding auto meter as well. I will just indicate that this is my automaton N1. This is it is start state and this is indicating that there may several edges going from this part of the structure to this state, which is the final state of N 1. Similarly, I have this is N 1is for R 1. Similarly, you have automaton here, we have N 2 which accepts the words of R 2 given these two and then to… Now, put together an automaton for R 1 plus, plus R 2 note that this is the union of the language of R 1 and R 2. So, we must be able to accept any string that is acceptable either by either of the two. Okay?

So, I am going to have a state new state to indicate the start state and then we will have an arrow. So, this is actually original N 1 and then I have, since it is a non determininistic automaton, you can have edge another edge. So, we have this would have been N 2, what we have done is merged the final states into a single state. We have created a new state for a start state, you can start from here and either way you can move. Once you go there without spending any symbol, you can move here and then you go there. Notice that our requirement is that when you hit a labelled state at that time, you must spend one symbol of this label to move forward. So, if my string is just a, then even if there were some other states without label I can proceed. But the moment I hit here I can only move forward, if I have a in the input this is our requirement.

So, suppose this is was very simple, simply a single state with a label a and this was a single state with label B, then I would have moved from here and if I had a string containing just a, then it would have allowed me to move one to the final state. If I had simply B, then this will not allow, but this will show this makes the union of the two languages. How about concatenation of the two languages? This is a set of two words where you can each word is castellation of word from l R 1 and a word from R 2. So, this will essentially be the same R 1, R 2 maton earlier there was a final state, we will replace that and directly put the automaton for R 2. Then we will let this go to state F.

So, if you notice this is exactly the automaton of R 2. If you treat this as final state then this would have been the automaton for R 1, so we just put them together, so its again easy to see. Okay? Here is the start state should indicate that if there was initial substring in my input belonging the language of R 1, then there was a way to reach here at the end of that and here I can begin the second part, here where I had a string of R 2 and then I would have reached the final state.

(Refer Slide Time: 27:52)


R_1^*
R
Claim: $l_1 l_2 \cdots l_k \in L(R)$ iff
$F(F(\cdots F(F(\{s\}, l_1), l_2)\cdots), l_k), *) \neq \phi$

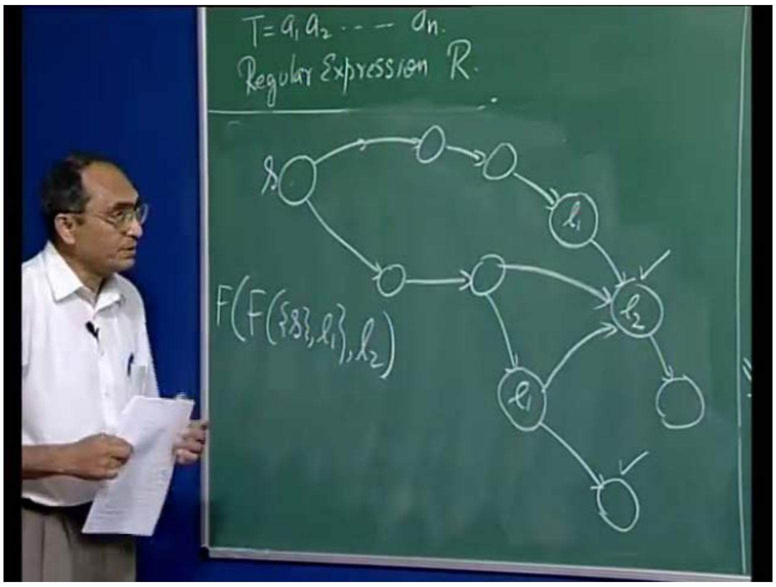
The last one is a clean closure and to compute automatons for R 1 star, we are going to take the automaton for R 1. Now, earlier I had the final state for R 1 instead of that instead of that I am going to introduce a new state. I will introduce a new state and then let it go to the final state. Then we will have this as out start state, but this will also have one way to go to the to the original state of R 1. We have noticed that if I start I can immediately move into this with empty string, because empty string is in the closure also I will just move here and after one string of R 1. I will reach back to this, I can either go there or resume the search.

So, you notice that these are the steps to construct non-deterministic finite state automaton for any regular expression R all right. So, once we have defined all the constituent steps, we can now build the automaton for any regular expression. Now, at the end once we have built and automaton for R symbolically state is looks like this. We will make just one more addition to this and that will be… We will invent a new symbol a special symbol and associate that with the so called final state. Then attach to this the state which will be actual final state and now I am going to claim.

The claim is that any string l 1, l 2, l k belongs to the language of R, if and only if starting with the start symbol called this state this. So, I start with S search for all the states which pass through l 1 and then proceed with this set of states, I go beyond l 2 and go on and finally, I will have l k. Now, that will allow me to go past the last state and then ultimately I must be seeking this symbol, because this is the symbol of final state.

So, at last I will do this, if I do this and I still have some state left that means I have manage to reach the final state. So, this should be non-empty that is the complete characterisation of the membership of this string into the language of R. Now, to justify this I will not give any formal argument, but just to justify this. Let us just take a look at, let us say a small example.

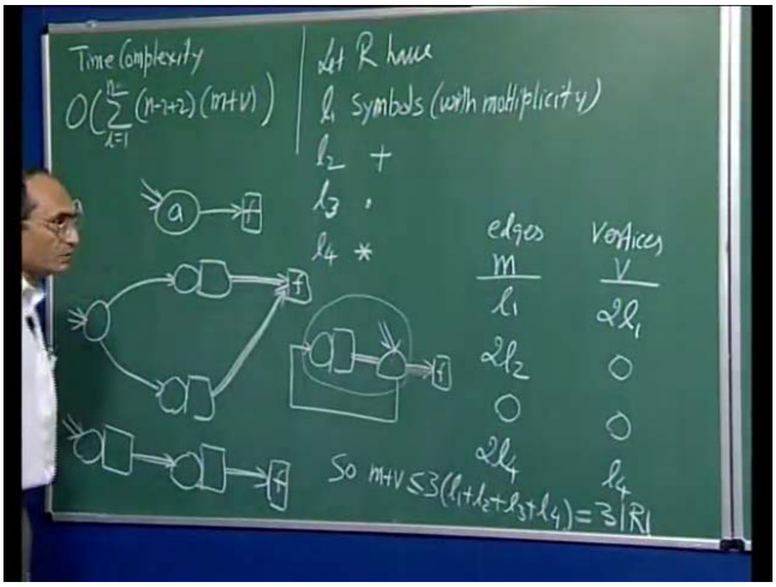
(Refer Slide Time: 32:01)



Suppose, I have a start state and I have a state with label l 1 maybe then I have state with label l 2 of state. Here maybe similarly, then the application of F on state s with l 1 will allow me to reach this state? Because we have just crossed l, it will allow will go this way this way will terminate, because we have just find l 2 directly this path will also find this, here maybe have another state here. So, we will reach this state and this state.

If I apply F on this set of states which is this and this with l 2, then I will find that l 2 matches right from the beginning, we have a state matching with the desired symbol. I will jump to this state. Let us say this is not labelled with l 2. So, this will continue if there is a further path will resume search. So, what will happen? Let us come back to this automate, that ultimately I will reach a state, if indeed this was in the language. I will reach a state in this automaton from where there is a absalom path, unlabeled path of arbitrary length to this state, because this was the original final state. The last application with respect to the sharp symbol will go reach here and jump finally, output this state as the value of this expression. That means that if this is non-empty means I have reached here, that would mean there was indeed a path through these labels and empty symbols in between which took me to the final state. Hence, if this is true then this symbol this word is indeed in the language of the automaton and converse it. So, the search cost for any given string of length k will be k plus 1 applications of, so that is the routine that we had written will be run k plus 1 times on this automaton.

(Refer Slide Time: 35:28)



Time Complexity
$O(\sum_{i=1}^{n} (n-i+2)(m+V))$
Let R have
l_1 symbols (with multiplicity)
l_2 +
l_3 ·
l_4 *
a
f
edges
Vertices
m
V
l_1
$2l_1$
$2l_2$
0
0
0
$2l_4$
l_4
So $m+V \leq 3(l_1+l_2+l_3+l_4) = 3|R|$

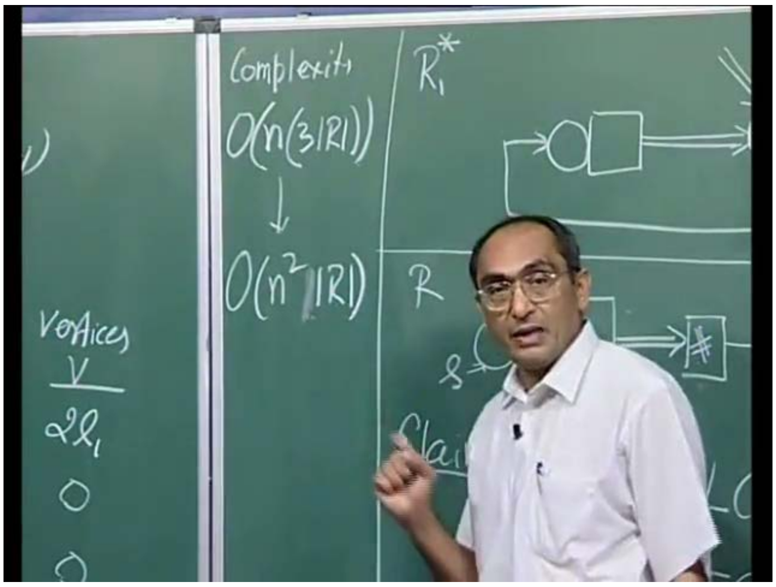
Now, the time complexity will be, if I start the search starting from each x i, then will be i equal to 1 to n that will be n minus i plus 2, m plus v. So, let us now try to decide the value of m and v, for this we have to go back and see how we have built our automaton? So, for a single symbol the automaton looked like, for each symbol we will generate two states and one edge. So, let me now we assume, let R have l 1 symbols by symbols I mean total with multiplicity the total occurrence is of the symbols. So, if there are two occurrences of a it will be contributing to l 1, l 1 symbols with multiplicity. We have l 2 or symbols l 3 concatenation and l 4 closure symbols.

So, we will estimate our edges and vertices which is V. We get to l contribution l 1 contribution and l 1 contribution due to the symbols themselves, then we have our operation in that what we did is, we made a new state here and we had a. So, whatever these automata already had that we do not have to account for. But we have created a new state here and we have merged two states here. So, we have not created any additional state in this case.

So, the number of states will be nil and number of edges. We have created will be one and two more, so 2 l 2 edges due to the R operation or plus operation the catenation. So, whatever the numbers of edges, where we have a final state, here this was our final here. In this case, once again the final state of the first automaton has been removed and we have used this as the replaced that final state by the state.

The final state of the second automaton is here, so no states here. We do not have any new edges, either there are no new edges in this construction. So, we have 0 edges and closure was built in the following fashion, we had this was the original automaton. So, we have added one more state, we have added one more edge here and one edge here, so two edges that will mean 2 l 4 and 1 l 4 here. So, m plus v, m plus v, is less than equal to 3 l 1, plus l 2, plus l 3, plus l 4. Notice that the symbol of either the expression for R its complexity is l 1 plus l 2 plus l 3 plus l 4. This is the total number of symbols in it. So, we can write this as 3, the complexity of R.

(Refer Slide Time: 41:35)



So, the time complexity becomes complexity will be order n 3 R, searching from any one symbol, if you are going to search from each i. Then this will be order n square bar R, this will be the total cost for searching for any string of the language of R, any whether any contiguous string belongs to the language of R. The, well this is actually polynomial, you notice that we used to have boot force algorithm for simple pattern and take string was n times same. If we think of this as the length of the pattern string and then this is n square m. But notice that here we are able to search the entire language generated by R. So, this is where we complete the discussion of pattern matching and will begin with a new topic in the next class.