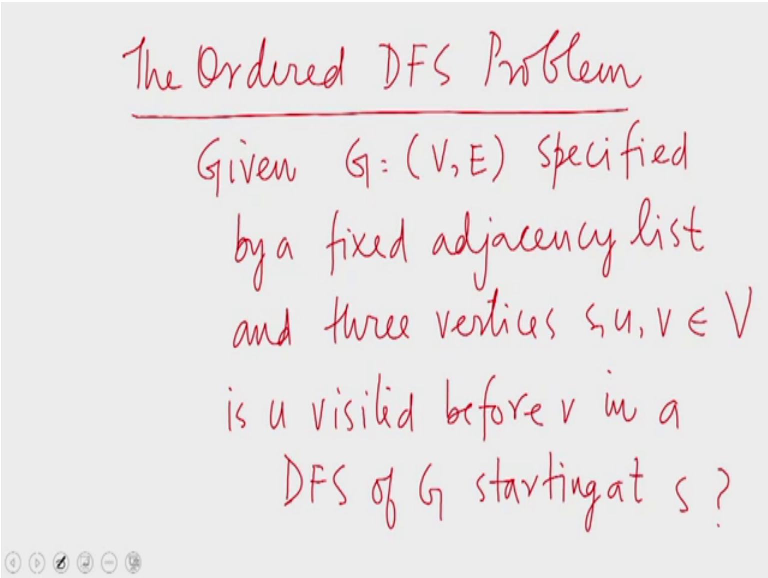
Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 36
Ordered DFS is P-complete for NC-reductions

Welcome to the 36th lecture of the MOOC on Parallel Algorithms. In the previous lecture, we found that the circuit value problem is P - complete for NC-reductions. We did this by showing that an arbitrary language of the class P can be NC reduced to the circuit value problem. Today we shall see that another problem, namely the order DFS problem is P-complete for NC-reductions.

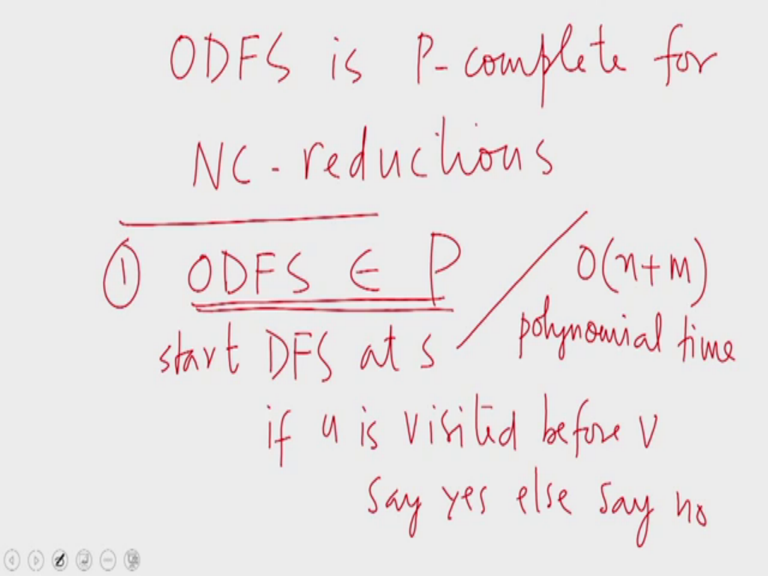
(Refer Slide Time: 00:55)



First let us define the problem, the ordered DFS problem is a decision problem. It is defined as given a graph G equal to V, E; where V is a vertex set and E is the edge set, specified by a fixed adjacency list and three particular vertices s, u and v of the graph is u visited before v in a DFS of G starting at s? This is the question that we have to answer. We are given a graph; the graph is specified using a particular adjacency list, we know that the choice of the adjacency list representation we will decide the order in which the vertices are visited in DFS.

So, given this adjacency list representation and three particular vertices s, u, v; you have to decide whether u is visited before v in a DFS, which begins at s. So, this is the decision problem at hand.

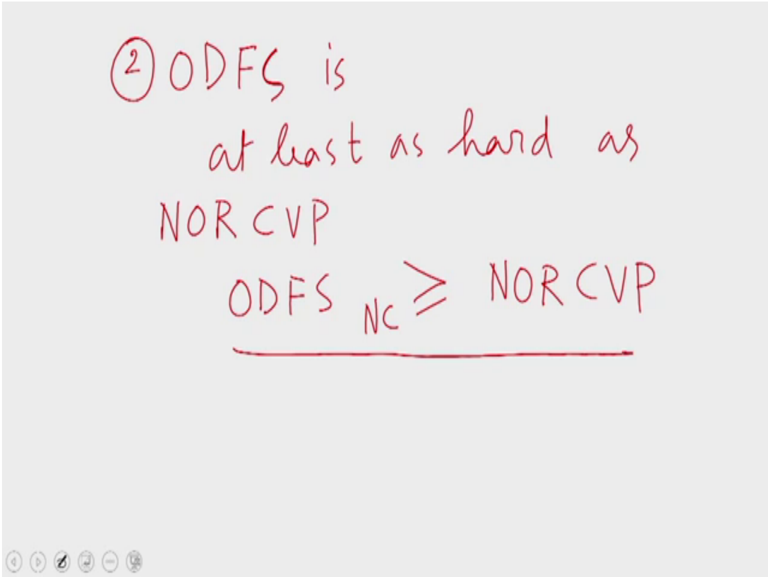
(Refer Slide Time: 02:24)



We want to show that the order DFS problem is P-complete for NC- reductions, proving this as we know involves two steps. First of all we have to show that the order DFS problem belongs to P, but this is easy to show you are familiar with the DFS algorithm. You start DFS at s if u is visited before v, say yes else say no.

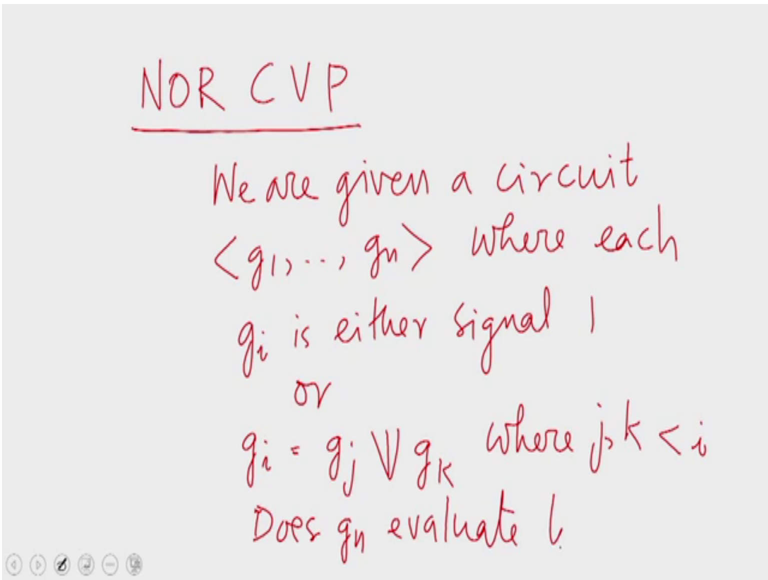
We know that DFS on a graph of n vertices and m edges will run in order of n plus m time, the number of edges in the graph is at most order of n squared. So, this is a polynomial time algorithm. Therefore, ODFS belongs to P. So, this is the first part of the proof.

(Refer Slide Time: 03:30)



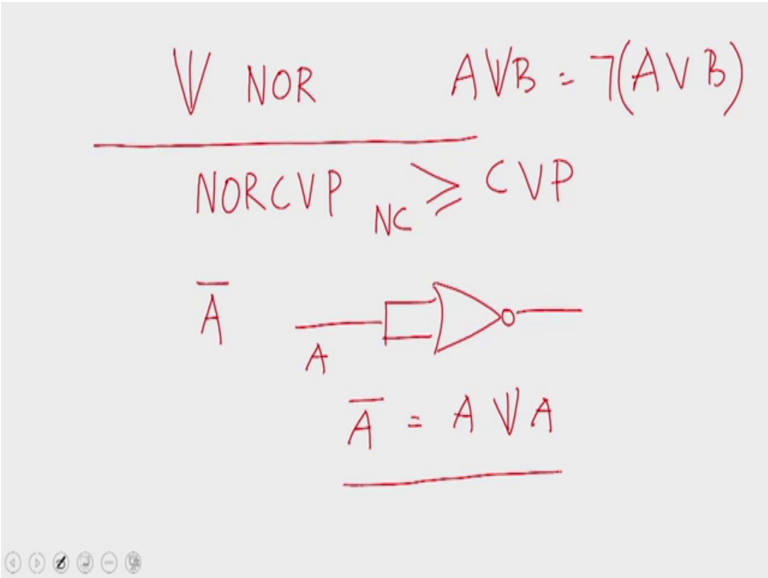
The second part of the proof involves saying that ODFS is at least as hard as CVP. In fact, we shall consider a variant of CVP we shall consider what is called NOR CVP that is we shall show that ODFS is at least as hard from the perspective of NC-reductions as NOR CVP or we shall show that NOR CVP is NC reducible to ODFS. This will be the second part of the proof; combining the two proofs, we would have established that ODFS is P-complete for NC-reductions.

(Refer Slide Time: 04:21)



Now, what is NOR CVP? NOR CVP is a variant of CVP. So, in this case we are given a circuit as in CVP, the circuit consists of a sequence of gates, where each g i is either signal 1, mind you g i is not allowed to have a value of 0, g i is either signal 1 or g i is g j NOR g k, where j and k are less than i.

(Refer Slide Time: 05:01)

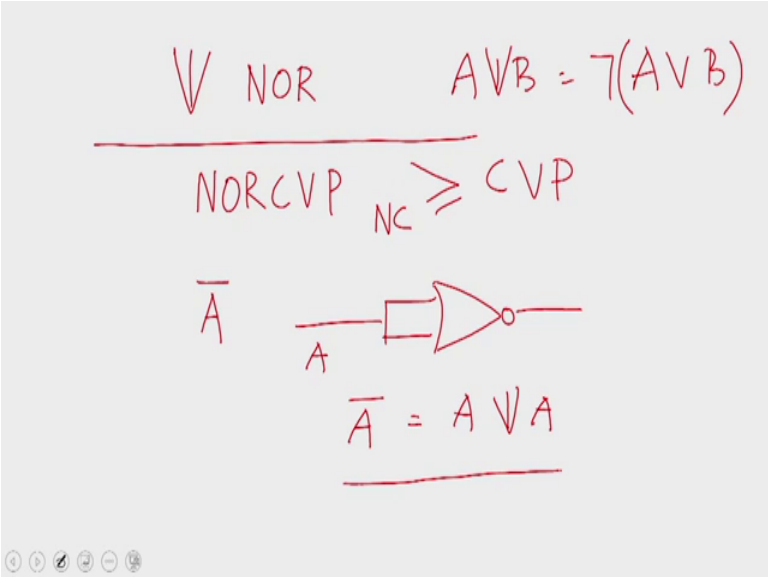


So, if you are not familiar with this notation this stands for NOR, so that A NOR B is the same as A or B negation. So, NOR CVP is an instance of this form we are given a sequence of gates and we have to say, if g n evaluates to 1 or not? This is the question we have to answer. Now, it can be easily shown that NOR CVP is harder than CVP or CVP is NC reducible to NOR CVP that is because when we are given an instance of CVP, we can convert into an instance of NOR CVP.

Mind you, CVP is a general Boolean circuit and in NOR CVP we are supposed to contain a Boolean circuit with only NOR gates. So, an instance of CVP can be converted into an instance of NOR CVP by replacing every single gate of CVP with an equivalent circuit using only NOR gates, this substitution can be done in the following manner. The negation of a signal can be achieved like this, this is a NOR gate, this is because as you can verify the NOR of A and A is a compliment.

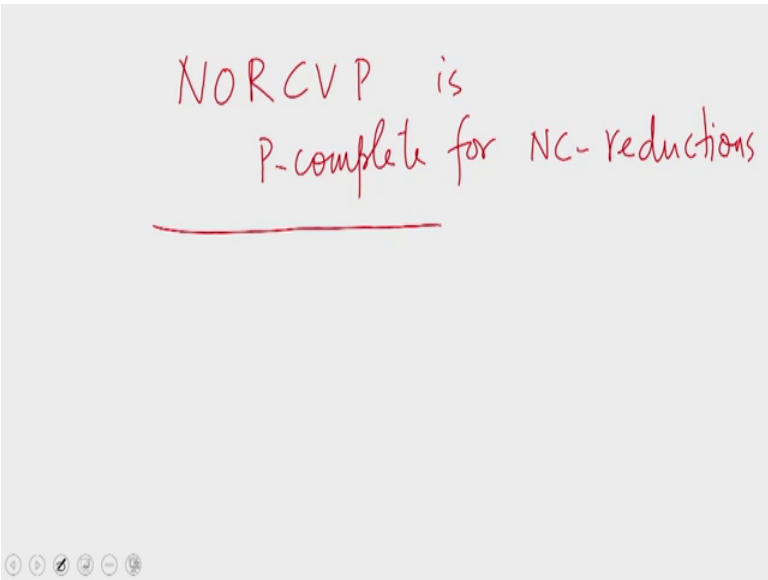
So, even if the circuit has only input values 1, we can generate the 0’s that we want by negating the 1’s. So, 0’s can be generated in this manner and also negations can be generated in this manner using NOR gates.

(Refer Slide Time: 06:41)



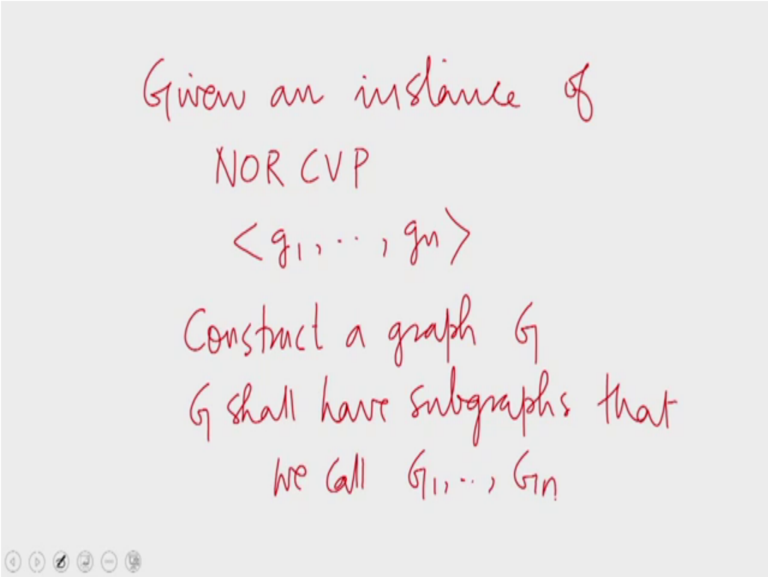
A or B is the same as A NOR B negation, but negation is already known to us, we know how to create negation using a NOR gate. So, A NOR B NOR A NOR B is A or B. On the other hand A and B by De-Morgan's law is the negation of A or the negation of B complemented, which is A negation NOR B negation; but A negation is A NOR A and B negation is B NOR B. So, an AND gate also can be synthesized using NOR gates. Therefore, given an instance of the circuit value problem, we can construct an equivalent instance of the NOR circuit value problem.

(Refer Slide Time: 07:36)



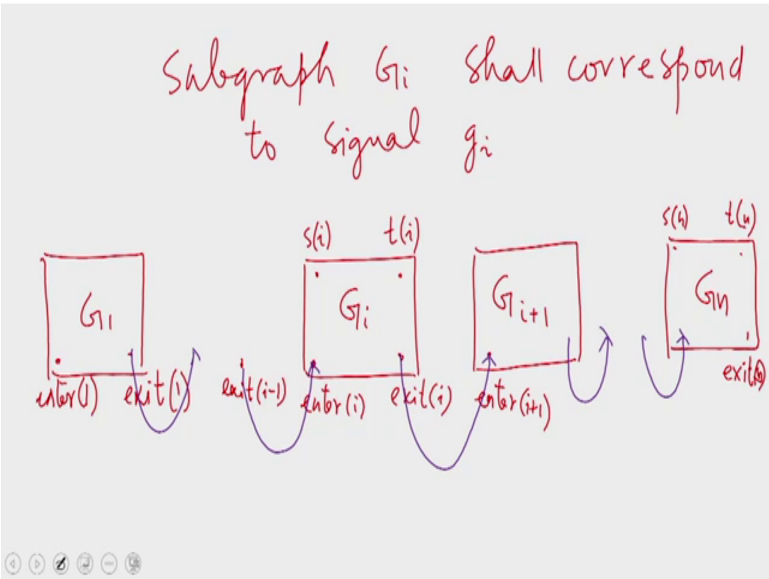
So, that establishes that NOR CVP is P-complete for NC-reductions. So, this is the result we are going to use we shall show that ODFS is NC reducible to NOR CVP.

(Refer Slide Time: 07:53)



So, let us say we are given an instance of NOR CVP. So, let us say g 1 through g n is that instance, where each g i is either signal 1 or g j NOR g k for j and k less than i. So, the reduction is affected in this manner we construct a graph G i G, the G that we are going to construct shall have sub graphs that we call G 1 through G n.

(Refer Slide Time: 08:29)



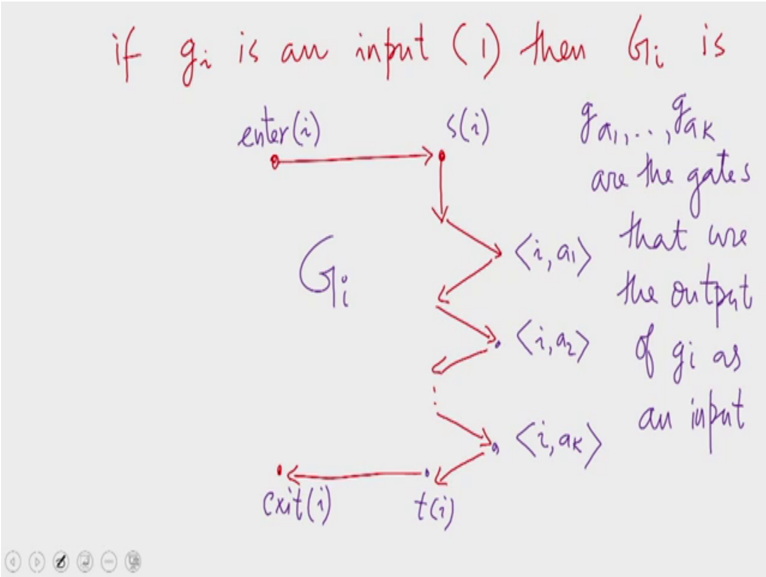
So, sub graph G i capital G i shall correspond to signal g i or the output of gauge small g i. And these units in particular will have these vertices, so if this is capital G i, it will have a vertex named enter i and a vertex named exit i, it should also have a vertex named s of i and a vertex named t of i. How these nodes are interconnected with in capital G i we shall see in a moment, but let us first see the overall structure of the graph.

If this is G i plus 1; G i plus 1 will have enter i plus 1. We shall have a directed connection from exit of i to enter of i plus 1, there shall be a connection of this form. Similarly, enter i will have an edge coming into it from enter i minus 1, exit i minus 1. Exit i minus 1 will belong to the sub graph G i minus 1. Of course, the first sub graph which is G 1 shall correspond to input G 1, it will have a vertex called exit 1 which connects to enter 2. It will have also have a vertex called enter 1, but there will be no edge coming in to enter 1 that is because there is no unit before this.

Similarly, the last unit will be a sub graph called G n, this will have a vertex called a exit n with no out degree, exit n will not be connected to any other vertex that is because G n is a last unit. Now, here there will be two particular vertices; s of n and t of n. Exit 1 will be connected to exit 2, enter i exit i plus 1 will be connected to enter i plus 2, similarly, enter n will be receiving an edge from exit n minus 1 and so on.

So, apart from these connections that are shown in blue, there will also be some edges some vertices that are common to multiple components. So, these components would interact even in some other ways all that will become clear, once I show how these components are designed.

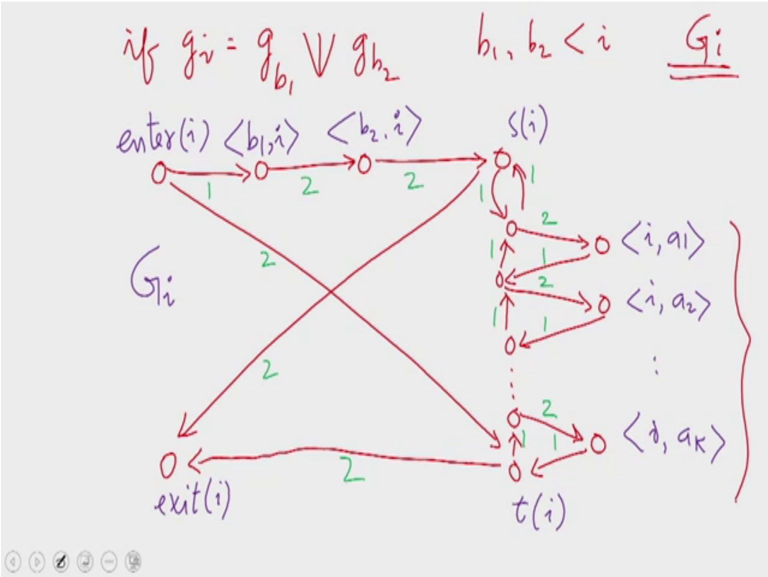
(Refer Slide Time: 11:02)



Now, let us see how these components are designed. If g i is an input of course signal 1 every input is of value 1, then G i is going to be a sub graph of this form. This is the vertex called enter i, this vertex is named s of i and this vertex is named t of i, and this is vertex named exit of i. Now, we have some vertices here which shall be named i a 1, i a 2 and so on.

Here we assume that a 1 through a k are the gates or strictly speaking I should say g a one through g a k. So, a 1 to a k are the indices of these gates, g a 1 through g a k are the gates that use the output of g i as an input. If g i is in this case g i is an input itself therefore, these are the gates that will use g i as an input. So, this is how graph G i will look like if small g i is an input in particular g 1 is an input and therefore, sub graph g 1 will look exactly like this with i replaced by 1.

(Refer Slide Time: 12:40)



Then, if g i happens to be the only other possibility for g i is that it is a NOR of g j and g k or let me use different subscript g b 1 NOR g b 2 for b 1, b 2 less than i. If g i happens to be a NOR of g b 1 and g b 2, then G i will be designed in this manner. Here we have a vertex, which will be enter i and then we have 3 vertices of this form and we setup edges of this sort. So, exactly as in the input case we have a connection to the exit vertex like this, but we shall also have edges of this form.

So, now let me label the vertices, this is vertex enter i, this corner as before is s i and this corner as before is t i and here we have exit of i. This is as before i a 1; this is i, a 2 and so on and this is i, a k. These vertices are numbered exactly as in the previous case, this is the vertex b 1, i; this is the vertex b 2, i.

So, as you can see b 1 of i belongs to G i as well as g b 1. So, similarly b 2, i belongs to graph sub graph g b 2 as well as sub graph G i. So, as I said some vertices can belong to multiple sub graphs. So, the sub graphs will interact in this manner as well some vertices could be common to various sub graphs.

Now, there is one more piece of data to be added, which is the label for the edges. So, first let us consider this graph and label the edges in this manner. I will put a label of 1 here, 1 here and 1 here when the label of an edge is the ordinal value of the edge within the adjacency list of the originating vertex. So, at vertex i, a 1 what we say is that this edge which is marked is going to be the first edge in the adjacency list of i, a 1.

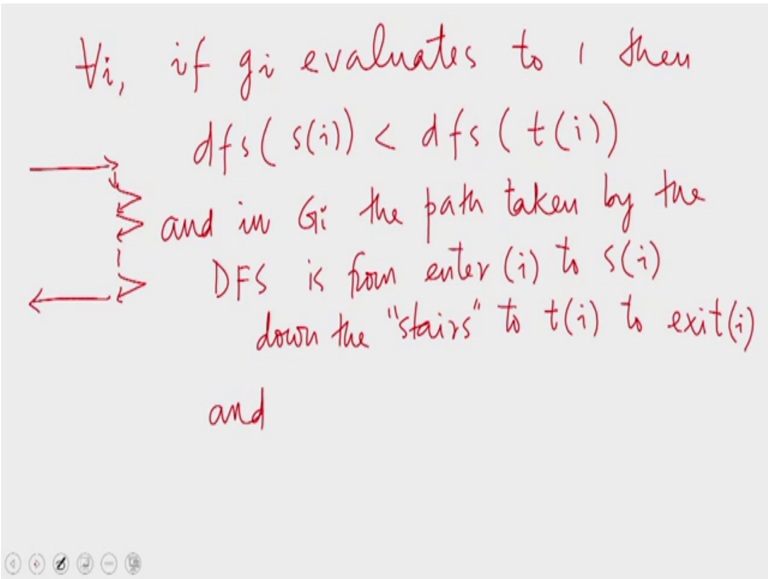
If an edge is unlabeled, it means its source vertex has only one edge and therefore, this edge is labeled 1. So, every unlabeled edge by default has a label 1 and that is the only edge belonging to the adjacency list of the source vertex. For example, enter i has a only one edge, when g i is an input and therefore, the edge going out of enter i is left unlabeled. Vertices i a 1, i a 2, etcetera are going to have multiple edges incident to them. Therefore, we have to label the edge going out of it, this particular edges labeled 1.

Now, for the case where g i happens to be the NOR of 2 gates, the labels would be given thus. Here we have a label of 1, here we have a label of 2 and here also we have a label of 2, these two cross edges have labels of 2 each, these edges are label 1, this edge is label 2, this is label 1, this is label 1, this is 1, this is 2, this is 1 and so on.

So, you can imagine that this would be 2, this would be 1 and this would be 1, this edge is labeled 2. So, I think here we have labeled every single edge belonging to this graph. The purpose of labeling the edges is that we are prioritizing certain edges over the others, a an edge which is labeled 1 will be taken before an edge label 2 that is when this vertex is visited. The first edge that is taken by the DFS is going to be the edge, which is label 1 out of this vertex.

Only after all the searches that are accessible from that vertex are finished will the DFS come back to this node and then trace the edge which is label 2, if the destination vertex has not been visited yet, so that is the purpose of labeling the edges in the fashion.

(Refer Slide Time: 17:31)

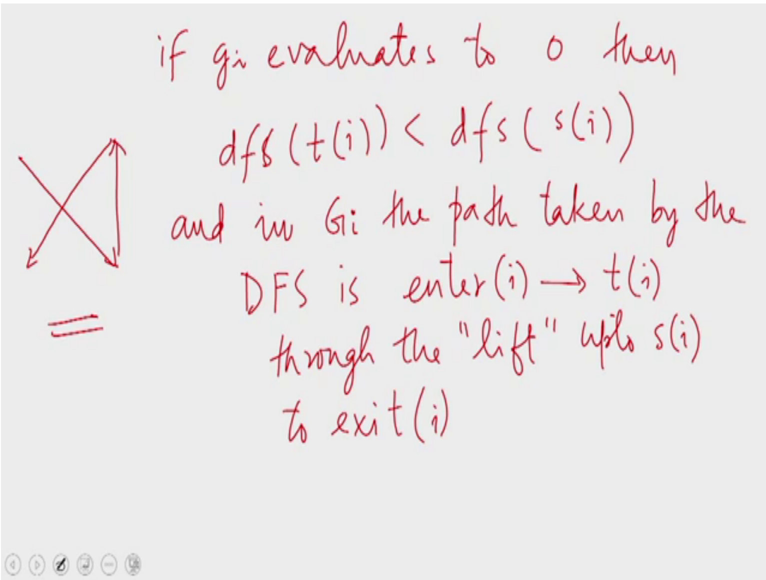


Now, how do we achieve the reduction now? This reduction is going to be claimed using a theorem, we want to claim that for all i, if g i evaluates to 1, then DFS of s of i is less than DFS of t of i and in G i the path taken by the DFS is from enter of i to s of i down the stairs to t of i to exit of i.

In this picture, the path that we are talking about will be starting at enter of i traveling through b 1 of i, b 2 of i and then coming to s of i. And then going down the path labeled 1 and then through the edge label 2 to i of a 1 and then out through the edge labeled 1, and then to i of a 2 and so on; i of a 1, i of a 2, etcetera will be visited in turn. So, this will create the impression of taking the stairs downwards, until we reach the vertex t i and then we exit through exit of i.

In other words, we will be traveling in this manner the path taken would be like this, if g of i evaluates to 1 and the statement is not complete.

(Refer Slide Time: 19:39)

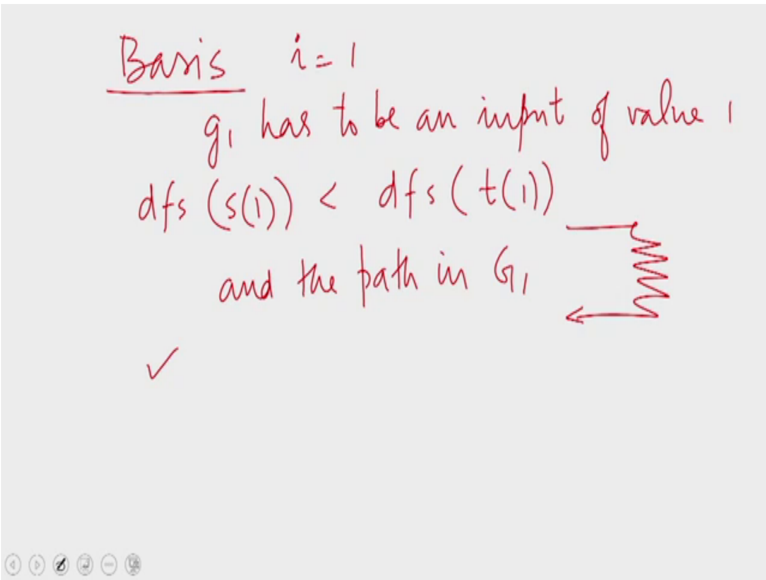


And if g of i evaluates to 0, which is the other possibility then dfs of t of i will turn out to be less than dfs of s of i, which means t of i will be visited before s of i. And as supposed to the previous case, in g i the path taken by the DFS will be different now. It will start from enter of i within g of i go to t of i and then through the lift up to s of i and then to exit of i.

Going back to the figure once again, starting at enter of i we would take the edge label 2 to come to t of i. From t of i we take the edge which is labeled 1 to go up from there again we will take the edge which is labeled 1 to go up and so on, going all the way up to s of i. This going up is what I termed taking the lift upwards. So, from t of i to s of i we take the lift upwards, traveling through the straight edges until we reaches of i, after that we take the edge which is labeled 2 to come to exit of i and then get out.

So, in this case the path trace would be like this. So, once again the claim that we make is this, for every i if g i evaluates to 1, then s i will be visited before t of i and if g i evaluates to 0, then t of i will be visited before s of i by the dfs. And moreover, if g of i evaluates to 1, the path taken by the dfs in G i would look like this one. On the other hand, if g i evaluates to 0, the path taken within G i by the dfs would look like this. So, this is our claim.

(Refer Slide Time: 21:38)

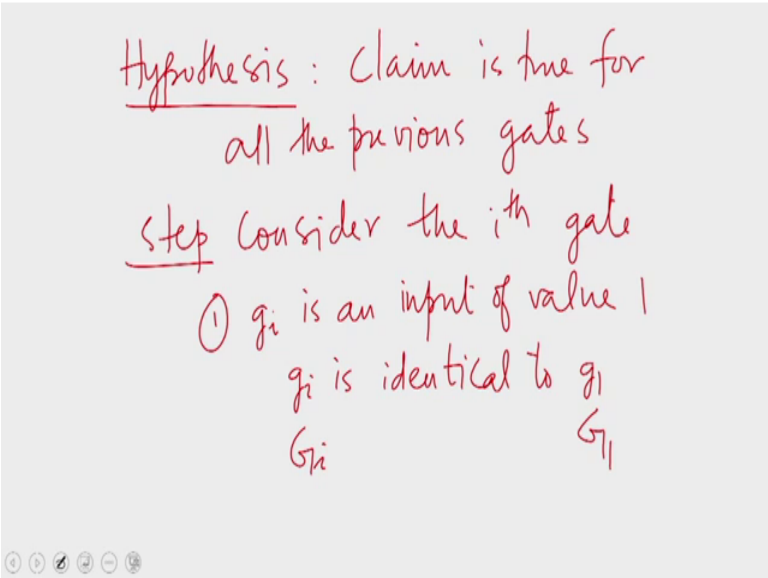


So, we shall prove this by induction. For basis let us consider i equal to 1, g 1 has to be an input of value 1. Therefore, according to our claim for our claim to be true, we should have a path of this sort traced within g 1. So, let us see what would how would the dfs go within g 1. So, g 1 is an instance of this i replace by 1.

So, the dfs will begin with enter 1, there is only one way for the dfs to go which is to s of 1 and from there, there is only one way to go from there it will go to i of a 1. At i of a 1, the dfs will naturally take the edge which is labeled 1, so it climbs down. And then again there is only one way to go which is to i of a 2, from there again it will take the edge which is labeled 1 and so on it takes the stairs downwards, until it reaches t of i.

There is no other way for the dfs to explore; the dfs must necessarily travel through this, because all these edges with labels have labels 1. So, from t of i, it will go to exit of i and then come out of g 1. So, all the vertices of g 1 would be visited in this manner, therefore the statement holds good. In other words, s of 1 is visited before t of 1 and the path in G 1 is of this form. So, the statement holds good for i equal to 1.

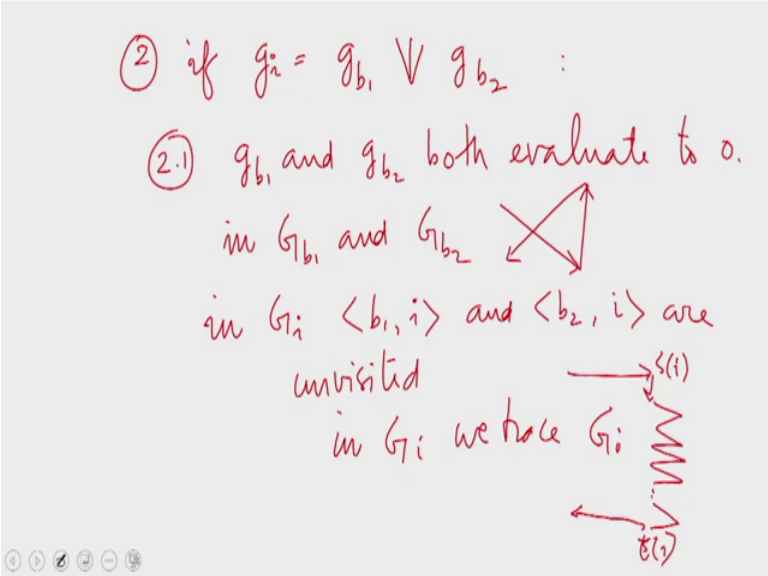
(Refer Slide Time: 23:09)



Now, as an induction hypothesis let me assume that the claim is true for all the previous gates. And then in the induction step, I am considering the i th gate. Now, for the i th gate there are two possibilities; one is the g i is an input, an input of value 1.

In this case, g i is identical to g 1. So, is capital G i to capital G 1 therefore, dfs will proceed exactly as it did in G 1. And therefore, path that we obtain is of this sort and s of i will be visited before t of i and in particular G i evaluates to 1 therefore, the claim holds good.

(Refer Slide Time: 24:07)



On the other hand, if g of i happens to be the NOR of 2 gates g b 1 and g b 2, then let us see. Then there are various possibilities, the first of the possibilities is that g b 1 and g b 2 both evaluate to 0. Therefore by induction hypothesis in G b 1 and G b 2, the paths taken were of this form, just we had a path of this form in both G b 1 and G b 2.

Now, imagine the graph g b 1 in g b 1; b 1 i was a vertex on the right hand side like this. In g b 1 this is g i instead if you imagine g b 1 here, i is a vertex that is to come later therefore, I mean i is a gate which will use the output of g b 1. Therefore, the vertex b 1, i would appear on the right side. Now, we know that g b 1 evaluated to 0 therefore, in g b 1 the path taken was from enter 1 to t of i and then up through the lift out through exit i, which means all these vertices on the right hand side had not been visited.

Which means in particular, all the edges which are labeled one out of these vertices on the right hand side have been taken, but vertices b 1 i has not been visited in particular and this is indeed the case with g b 2 as well. Therefore, we find that in G i b 1, i and b 2, i are unvisited, these are unvisited vertices.

Once again go back going back to the figure, now let us see how the dfs would proceed within G of i, when the dfs begins at enter of i, it tends to take the edge which is labeled 1; it explore along the edge labeled 1, it comes to b 1 i; b 1 i has not been visited before. Therefore, it will take the edge, which is labeled 2 out of this. So, when you come to b 1, i it would explore the edge which is labeled 1 out of b 1 i; but this takes us to a vertex which has already been visited in g b 1. Therefore, we backtrack come back to b 1 i and take the edge which is label 2.

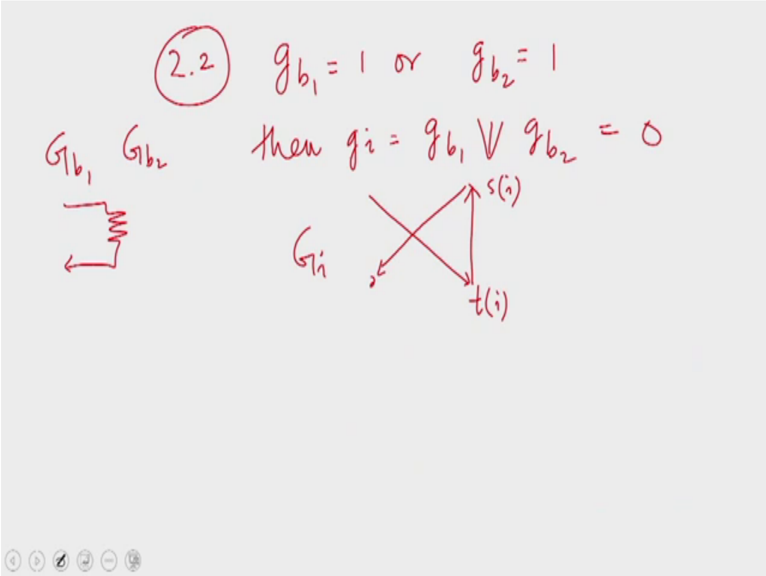
So, we have now reached b 2 of i; b 2, i, but b 2, i have not been visited before therefore, we visit b 2, i there is no backtracking. And then we consider the edge which is labeled 1 out of b 2, i; which again takes us to a vertex that is visited before in g b 2 therefore, we backtrack come to b 2, i and we proceed to s of i.

Once we are at s i we take the edge labeled 1 to go down the line from there the edge which is labeled 1 is going backwards that vertex is already been visited that is vertex s i has already been visited. Therefore, we would be going to i, a 1 from there we would take the edge which is labeled 1 to come to the vertex on the right side, which from there again vertex 1, edge labeled 1 is taking us back to the vertex which is already been visited.

So, we would be taking the edge which is labeled 2 to come to i, a 2. And we continue in this fashion visiting i, a 1, i, a 2, i, a 3; etcetera in turn until we come to i, a k. Once we are at i, a k again we take the edge which is labeled 1 we come to t i, once we are at t i the edge which is labeled 1 is taking us backwards to a vertex which is already visited. So, we do not take this edge we backtrack to t i and take the edge which is label 2 that will take us to exit of i. So, what we find is that the path that the DFS traces is from enter i to b 1, i; to b 2, i; to s of i and then through i, a 1; i, a 2 through i, a k then t i and then exit i.

This is exactly what the claim was since b 1, i and b 1, i were unvisited we were allowed to move along the path and therefore, in G i we trace a path of this form which is exactly according to the claim, this is the sort of a path that we trace within G i. And then obviously, s of i is visited before t of i, DFS of s of i is less than the DFS of t of i. So, in this case the claim holds good.

(Refer Slide Time: 28:41)



Now, let us consider the second case. In this case, let us say g b 1 equal to 1 or g b 2 equal to 1, at least one of them is 1 then g i evaluates to 0, g i happens to be the NOR of these two g b 1 and g b 2, g i evaluates to 0. What this means is that, when we visited one of the two graphs g b 1 and g b 2, we had taken this path; where s of b 1 came before t of b 1 that is of g b 1 had a value of 1. On the other hand, if g b 2 had a value of 1, then we would have done the same in g b 2.

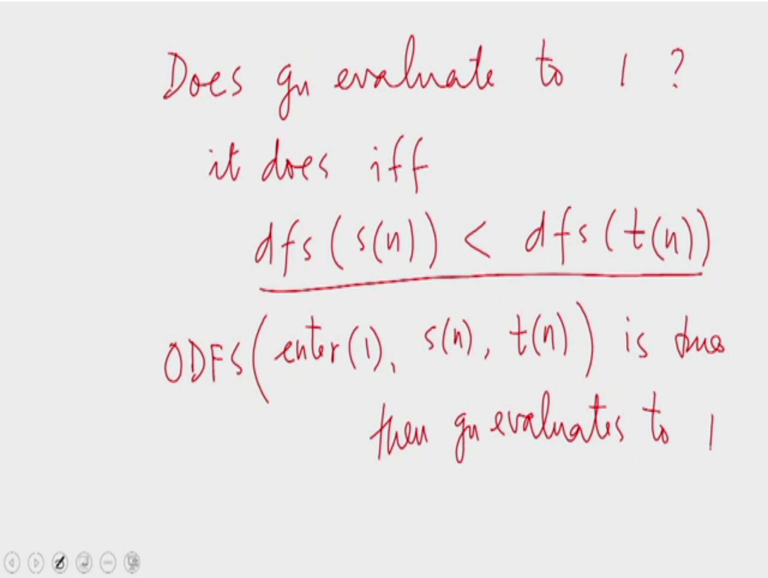
So, in at least one of the two graphs we had taken this path. Now, what does it entail? It means that the rightmost vertices had all been visited among them would have been either b1 of i; b 1, i or b 2, i. So, at least one of these two vertices b 1, i and b 2, i have been visited before. Therefore, when dfs starts at enter i here now, what happens is this it proceeds to b 1, i; if b 1, i had not been visited before, then we would proceed to b 2, i; but then we know that at least one of them has been visited. So, at one of these vertices we will backtrack, we will not be allowed to proceed further.

So, we backtrack all the way to enter i, once we are back at enter i, we attempt a search through the next available edge which is the edge that is label 2 that will take us to t i; t i has not been visited before, but once we are at t i we trace the edge which is labeled 1 that will take us up. So, we are now taking the lift upwards at each of these vertices we take the edge that is labeled 1. So, these edges will take us up to s i through the lift.

Once we are at s i, the edge which is labeled 1 is taking us back to a vertex that is already visited therefore, we should make an attempt through the edge which is labeled 2 that will take us to exit i. Now, we go out of graph g i. So, what we find is that the path traced happens to be in G i we take a path of this sort this is enter of i and this is exit of i. But in this case this vertex t of i is visited before s of i that is precisely what the claim was that t of i is visited before s of i or dfs t of i is less than dfs s of i.

So, once again the claim holds good, so that completes the induction therefore the claim is true. Once the claim is true, we can apply the claim to g n therefore, we find that in g n vertex s n is visited before vertex t n, precisely when gate g n evaluates to 1. Now, that is precisely our problem was with NOR CVP, we were given the circuit and we wanted to check.

(Refer Slide Time: 32:03)

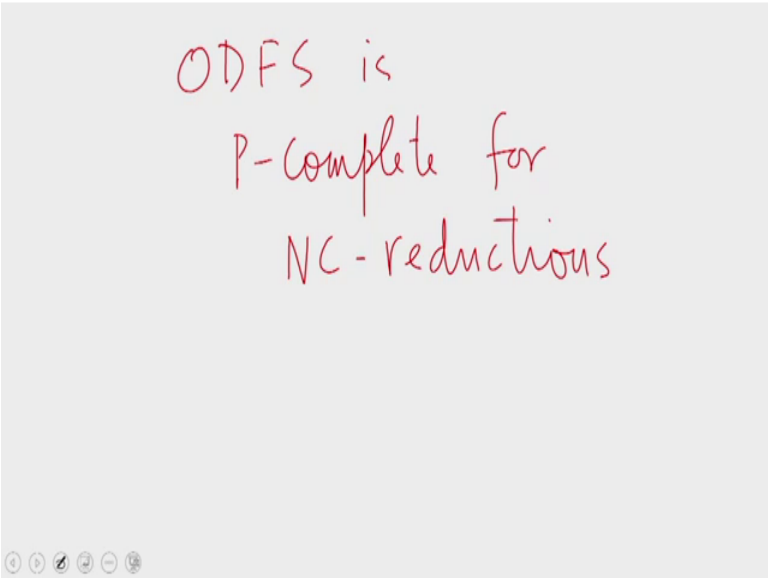


Whether g n evaluates to 2 or not, does g 1, g n evaluate to 1? This was our question, what we find is that this happens if and only if dfs of s of n is less than dfs of t of n, but this relation can be checked by invoking ODFS on the graph with three particular vertices. These are enter 1, the dfs must begin with enter 1 and then s of n and t of n. If this is affirmative, then g n evaluates to 1.

So, the NC - reduction that we are talking about involves this, given an instance of the NOR CVP circuit, we construct the graph g in this manner. And then invoke ODFS on this graph, but then how do you construct the graph? Constructing of the graph requires establishing sequences of certain sort. We have to establish sequences of vertices of the sort, but then you know all those can be done in poly logarithmic time using a polynomial number of processors.

Therefore, setting above the graph is indeed an NC work, so that establishes that the problem of NOR CVP can be reduced to ODFS that completes the second part of the proof.

(Refer Slide Time: 33:56)



Therefore, putting the two parts we know that ODFS is a P-complete for NC-reductions which is a bit of a setback, because dfs is one of the most elementary sequential algorithms and forms a subroutine to many algorithms, but here we find that DFS is not easily parallelizable. So, in the parallel setting DFS is not an easy problem. In the next class we shall show that the max flow problem is P-complete for NC-reductions as well therefore, we cannot expect to have efficient parallelization for that problem as well that is it from this lecture, hope to see you in the next.

Thank you.