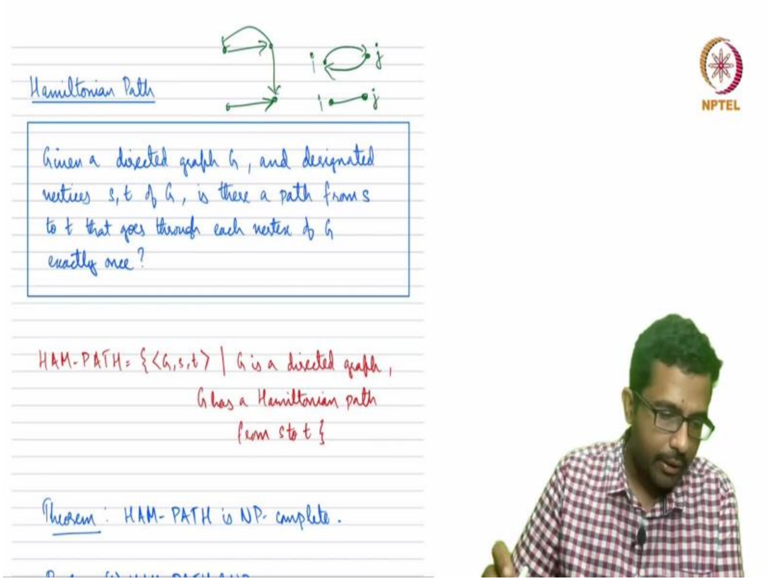
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 53
HAM – PATH is NP - Complete

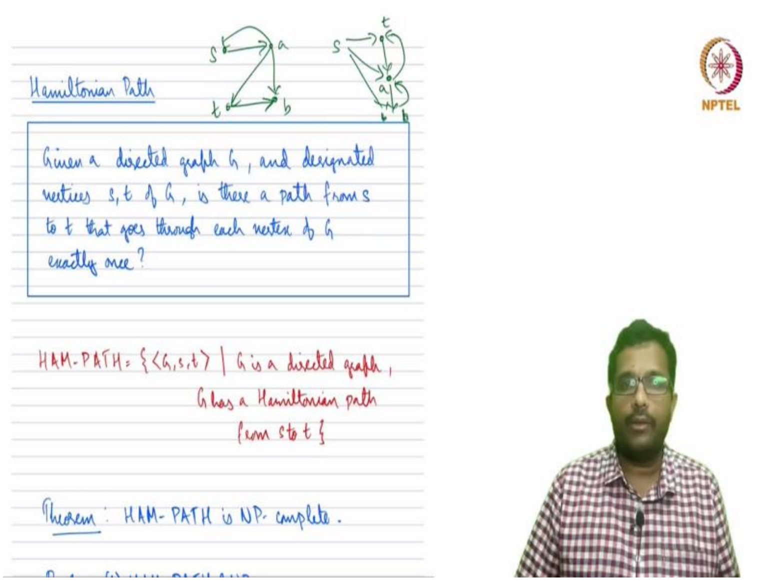
(Refer Slide Time: 00:16)



Hello and welcome to lecture 53 of the course Theory of Computation. In the previous lecture, we discussed two NP-complete languages, clique and vertex cover, which are both based on graphs. In this lecture, we will cover another NP-complete problem called Hamiltonian path. This lecture focuses on directed graphs.

A directed graph is a graph where the edges have directions. Previously, our edges were undirected, meaning the edge from i to j was the same as the edge from j to i. In directed graphs, edges are like arrows with directions, so an edge from i to j is not the same as an edge from j to i. In an undirected graph, give two vertices i and j, there could be no edge between them, one edge from i to j, one edge from j to i, or both edges could exist. These are the four possibilities: no edge, an edge from i to j, an edge from j to i, or both edges existing. This explains directed graphs.

(Refer Slide Time: 01:39)



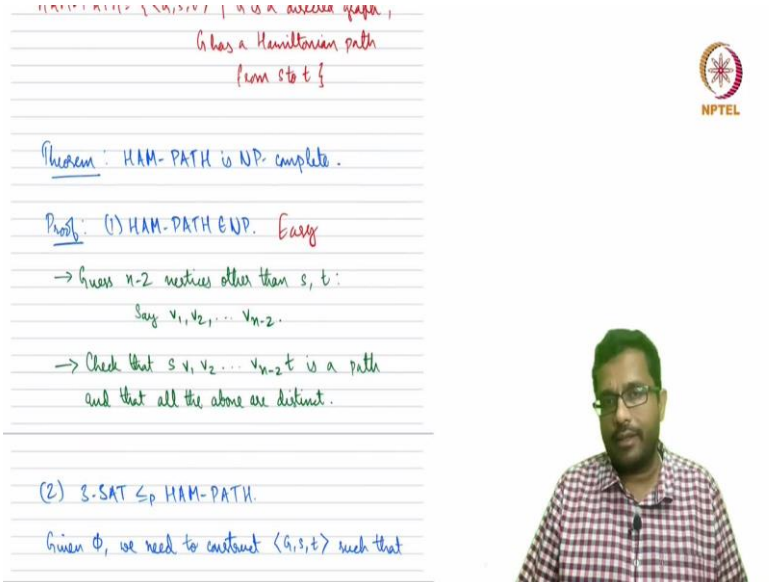
What is the Hamiltonian path? A Hamiltonian path is a path in a graph that goes through all the vertices exactly once. In this lecture, we will explore this concept with two designated vertices, s and t. The question is whether there is a path from s to t that visits every vertex in the graph.

In the given example, there is no such path. There is a path from s to t via a, but there is no path from s to t that includes both a and b. From s, you can go to a, but then you must go to either b or t. If you go to b, there is no way to reach t. Thus, this graph does not have a Hamiltonian path.

Consider another example with vertices s, t, a, and b. In this case, there is a Hamiltonian path. You can go from s to b, then to a, and finally to t. This path covers all vertices exactly once. However, if you go from s to a, then a to b, you cannot reach t. The correct Hamiltonian path is s to b to a to t.

A Hamiltonian path is a path from s to t that covers all vertices in the graph exactly once, without repeating any vertex. As a problem, it is given as a 3-tuple (G, s, t), where G is a graph, and s and t are vertices in the graph. It is a "yes" instance if there is a Hamiltonian path from s to t and a "no" instance if there is no such path. This problem is NP-complete.

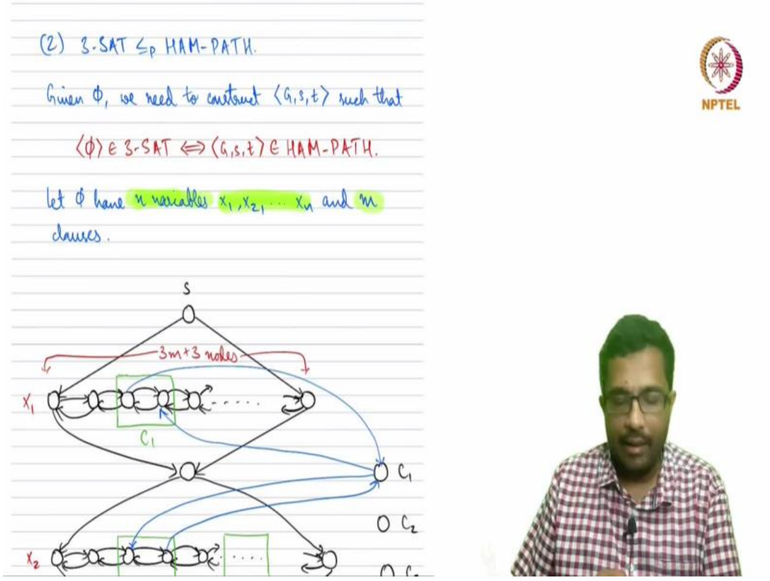
(Refer Slide Time: 04:28)



Like in the previous lecture, we will do two things. First, we will show that it is in NP. Second, we will reduce a known NP-complete problem to the Hamiltonian path. Showing that it is in NP is easy. We simply guess a sequence of (n-2) vertices. Suppose there are n vertices in the graph. We guess a sequence of (n-2) vertices, say v_1, v_2, ..., v_{n-2}. Then we check whether starting from s, followed by v_1, v_2, ..., v_{n-2}, and ending at t, forms a Hamiltonian path.

We verify whether this sequence forms a path by checking if there is an edge from s to v_1, from v_1 to v_2, and so on. We also ensure all vertices are distinct. This verification takes polynomial time, so we guess and verify the sequence, demonstrating it is in NP. We will not spend too much time on this part.

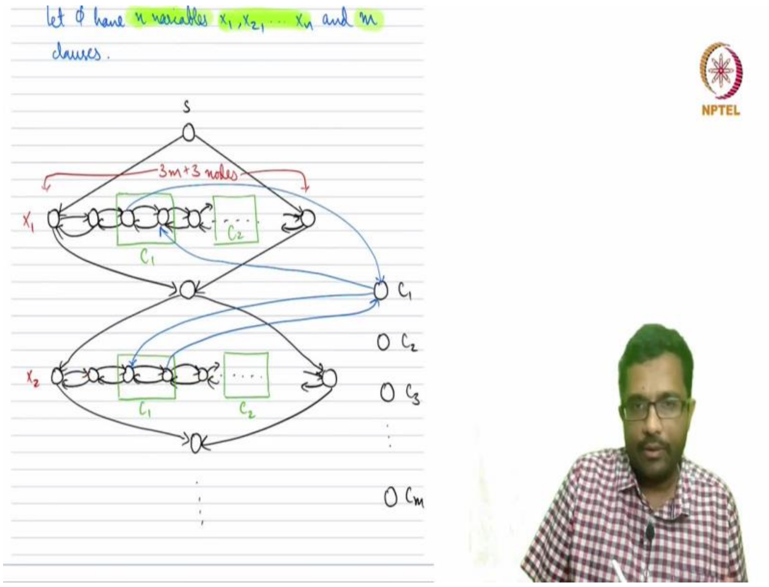
(Refer Slide Time: 05:37)



The more interesting part is showing that 3-SAT is reducible to Hamiltonian path. The NP-complete language that we picked is 3-SAT. Given a 3-SAT formula, we have to construct G, s, t, which is an instance of Hamiltonian Path such that this is a "yes" instance of 3-SAT if and only if it is a "yes" instance of Hamiltonian Path. In other words, the formula phi is satisfiable if and only if the graph G has an s to t Hamiltonian path.

So, let us see how to construct this. The construction is a bit more involved than vertex cover, so I will mostly be explaining through pictures rather than writing. Let phi be a formula that has n variables and m clauses. Let the n variables be X_1, X_2, ..., X_n and m clauses. Now, we are going to see how to build the Hamiltonian Path instance.

(Refer Slide Time: 06:45)

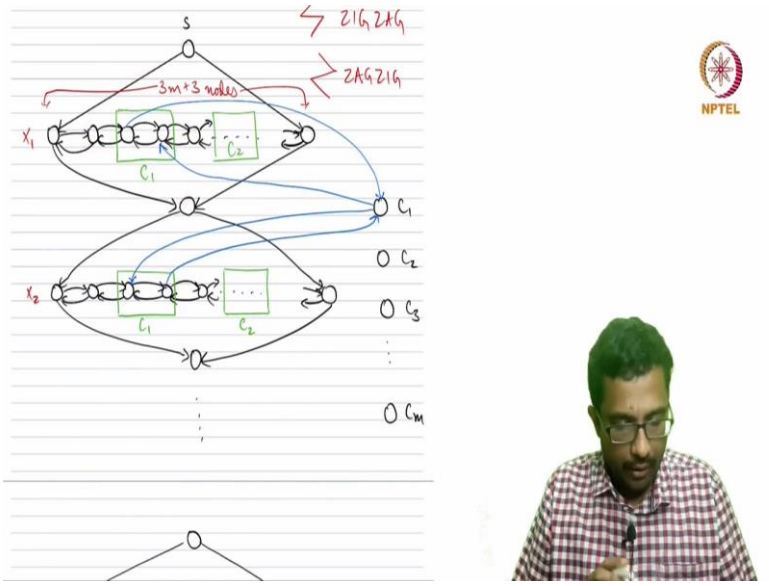


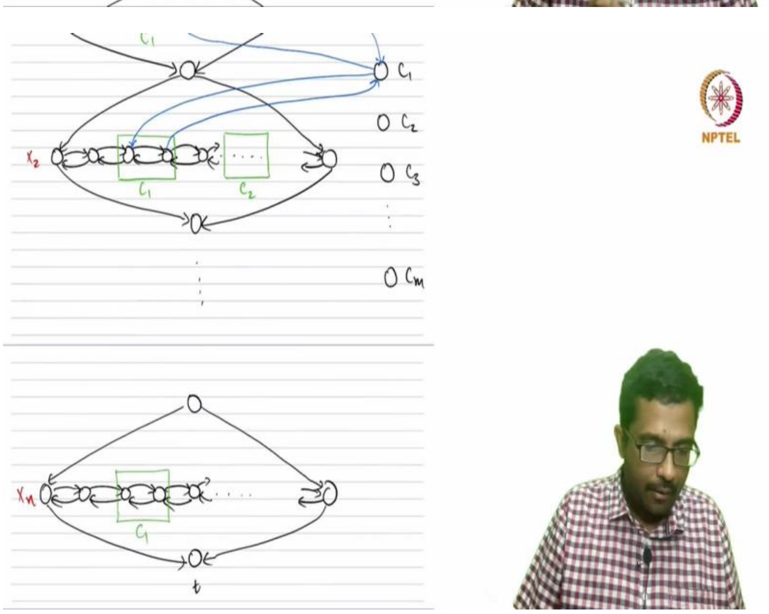
So, what we do is, we have this diamond-shaped structure for each variable. We start with X_1, so the top diamond corresponds to X_1 as marked here. It starts with s at the top. This vertex is s. I have this diamond structure where all arrows are pointing downward. The intermediate vertices, or nodes, have edges in both directions (left to right and right to left). There are 3m plus 3 vertices here, including the leftmost and rightmost ones. There is a path from the right to the left and a path from the left to the right in both directions.

Between each of these edges and vertices, there is an arrow from left to right and an arrow from right to left. I will explain why there are 3m plus 3 vertices. After the first two endpoints, we skip one vertex, and the next two vertices correspond to clause C_1. Then, we skip another vertex, and the next two vertices correspond to clause C_2, and so on, up to m clauses. Finally, we skip another vertex, and the last vertex is the rightmost one. This gives us 3m plus 3 nodes. There are 2m for the clauses and the rest will be m plus 3, so we get 3m plus 3 nodes.

Once again, we have a diamond starting from s at the top, going down to the left and right, and then converging to another vertex. There is a series of 3m plus 3 nodes with a path from left to right as well as right to left. We have identified some correspondence between two vertices and a clause, then skip another one, and the next two correspond to C_2, and so on, for each m clauses.

The same structure repeats in the next diamond for X_2. The top of the X_2 diamond is the bottom of the X_1 diamond with the same vertex. The same pattern continues, with a path from left to right and right to left, and identified boxes for C_1, C_2, etc. (Refer Slide Time: 10:22)





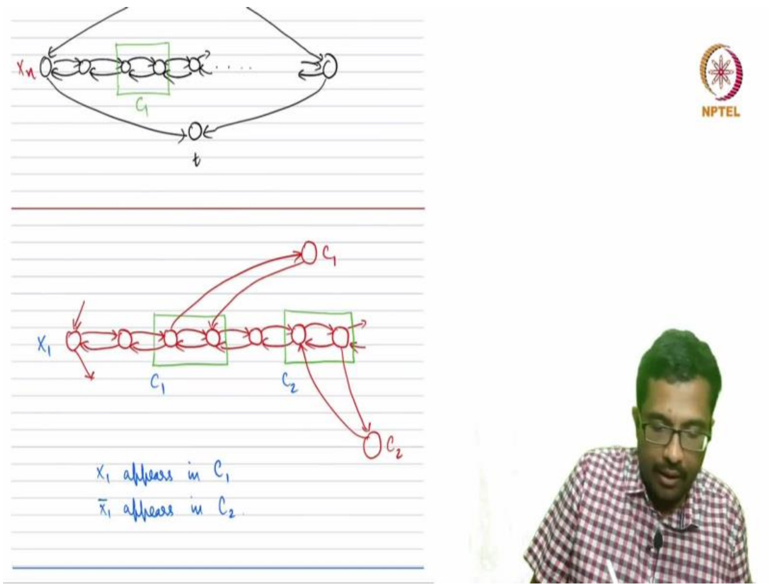
And finally, I have the nth diamond corresponding to the variable X_n. Similarly, I have these boxes corresponding to C_1, C_2, and so on. The bottom vertex of the nth diamond is t. The top vertex of the first diamond is s, and the bottom vertex of the nth diamond is t. There are more vertices that will come in a bit.

Ignoring the vertices on the right side for a moment, let's consider the diamonds and their vertices. How can we start from s, cover all these vertices, and reach t? One way is to start at s, go left, cover all these intermediate vertices going right, and then come down. Another way is to go right, cover all these vertices going left, and then come down. These are the two ways to cover the top diamond: one is a zigzag motion, and the other is a zagzig motion. Zigzag corresponds to the variable being set to true, and zagzig corresponds to the variable being set to false.

Starting from s, zigzag or zagzig determines the value of the variable. The same can be done for X_2. We can choose to zigzag on X_1 and then zagzig on X_2. We have two choices in each diamond, and there are n such diamonds. Therefore, there are 2^n ways to start at s and reach t, ignoring the vertices on the right side. So, there are 2^n Hamiltonian paths from s to t.

Now, let's consider how the additional vertices influence the paths. If we just add these vertices without a way to access them, there won't be any Hamiltonian path. We need to ensure that these vertices are reachable to maintain the Hamiltonian path from s to t.

(Refer Slide Time: 14:15)



So what we do is the following. Consider this figure: suppose a variable x1, let us say this is the diamond of x1. I have zoomed in a bit, focusing only on the diamond of x1. Suppose x1 features in clause 1. In the middle row of vertices of x1, there is a box that corresponds to c1, as we marked earlier.

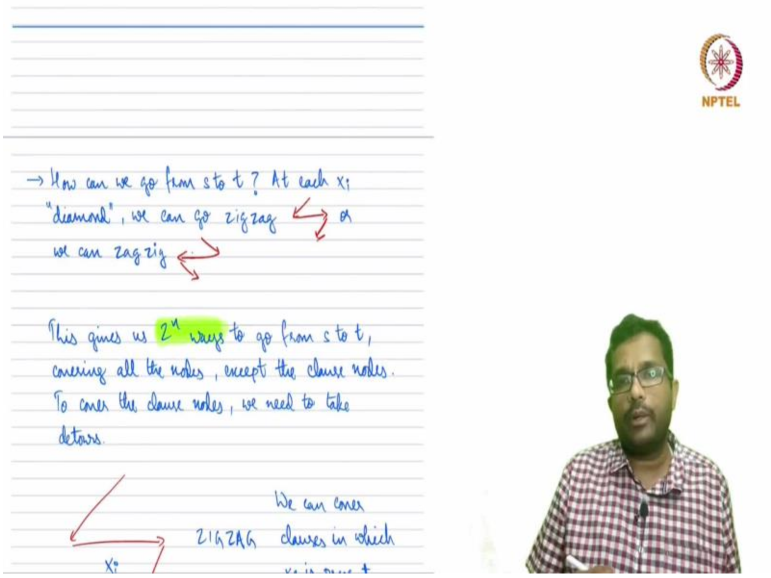
From the left vertex of that box, we put a directed edge to the c1 vertex. Before that, let me explain that there are going to be m vertices on the right side, corresponding to the m clauses. You have n diamonds, one for each variable, and m vertices, one for each clause. Let's take c1 and see how we connect c1 to this structure.

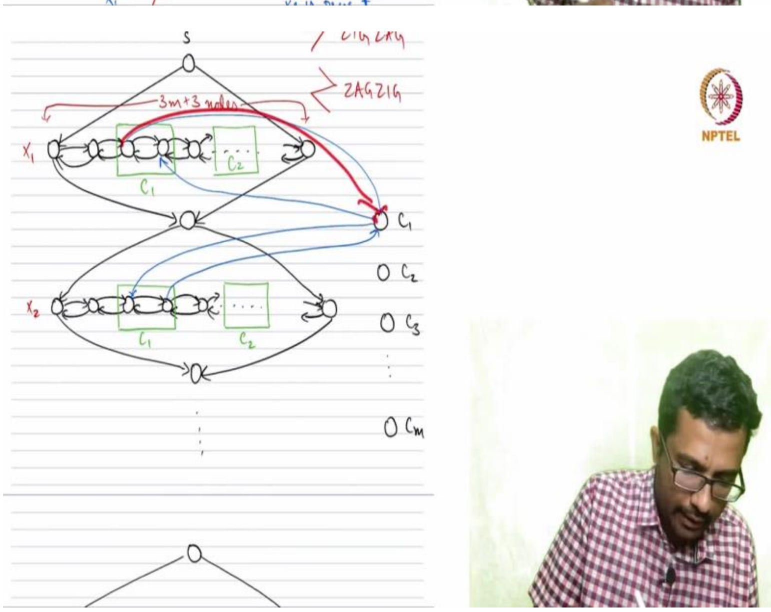
If x1 appears in c1, we look for the box marked c1 in x1. From the first vertex on the left, we put a directed edge to c1, and from c1, we put a directed edge to the second vertex in the box. If x1 appears in c2, we do the same thing. However, if x1 complement appears in c2, the order is reversed. From c2, we put a directed edge to the first vertex in the box, and from the second vertex, we put a directed edge to c2.

So, if the literal is positive, from the first vertex, we put a directed edge to the clause, and the second vertex receives the directed edge from the clause. If the literal is negative, from the clause, we put a directed edge to the first vertex, and from the second vertex, there is a directed edge to the clause. This is the setup for when x1 appears in c1 and when x1 complement appears in c2.

From the box marked c1, we only connect to c1 and nothing else. Similarly, from the box marked c2, we only connect to c2 and nothing else. That is why we have separate earmarked boxes for each clause.

(Refer Slide Time: 17:29)





And this is something I already mentioned: we can go from s to t if we ignore the clauses. There are 2^n ways to go from s to t, as each diamond can be zigzagged or zagzigged. Now, how are we going to include these clause vertices? I have already marked some blue edges based on what I said. Here, as per the figure, x1 appears in clause 1 because the first vertex goes to clause 1, and from clause 1, you get back to the second vertex.

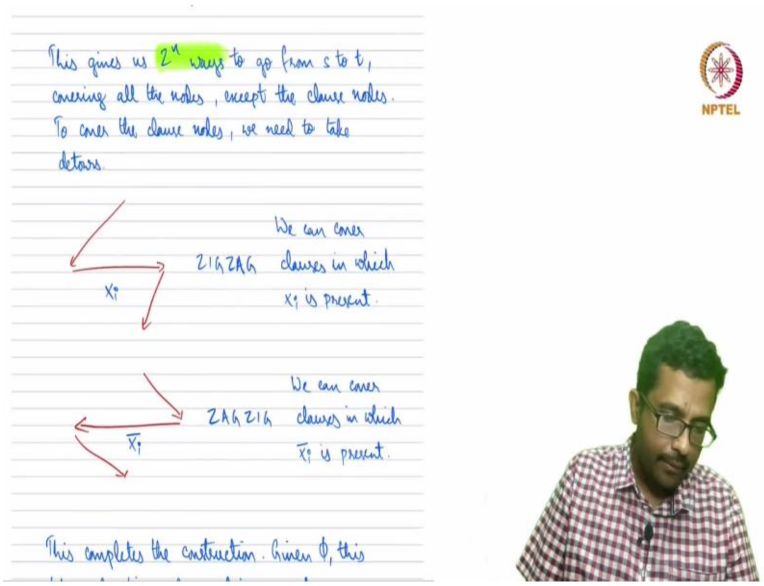
x2 complement appears in clause 1 because the order is reversed: the first vertex is receiving a directed edge from clause 1, and the second vertex is sending a directed edge to clause 1. So, here x1 appears in clause 1, and x2 complement appears in clause 1. If x1 appeared in clause 2, you would connect similarly.

Now, we have the Hamiltonian path, and we need to incorporate the clause vertices. If x1 features in clause 1 and we are zigzagging, it's easy. Zigzagging means we can start from s, come to the left side, enter the box for clause 1, move to the clause vertex, come back to the next vertex, and continue our zigzag journey. This is how we incorporate the clause vertex.

If x2 complement appears in clause 1, zigzagging will not work as the edges are in the opposite order. However, if we are zagzigging, this order is correct. We can come from the right side to clause 1, then come back to the first vertex, and continue. x1 can cover clause 1 if we are zigzagging in x1, and x2 can cover clause 1 if we are zagzigging in x2.

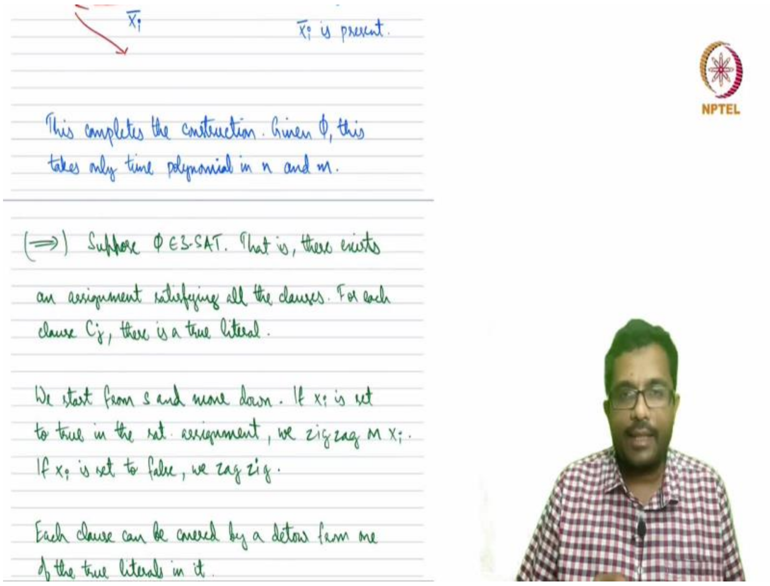
To summarize, zigzagging indicates that x1 is set to true, and zagzig indicates that x1 complement is set to true (or x1 is set to false). Zigzagging allows us to cover all clauses where the variable appears as a positive literal. Zagzigging allows us to cover all clauses where the variable appears in its complemented form (negative literal).

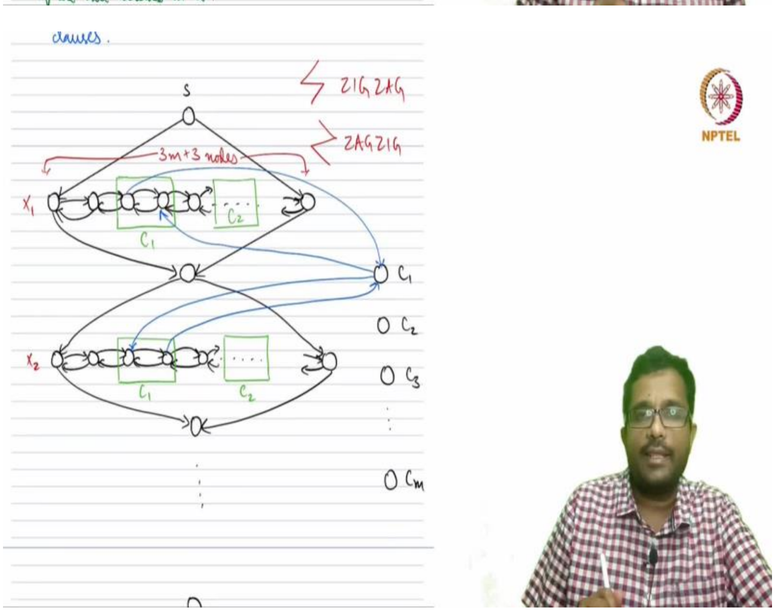
(Refer Slide Time: 21:18)



So if we are zigzagging we can cover the clauses in which xi is present and zagzigging, we can cover the clauses in which xi compliment is present. And that is the construction. So I have explained the diamonds, the top and the bottom, s and t, and how, for each and these clause vertices and how these are connected.

(Refer Slide Time: 21:44)





Now, one point that I am somewhat glossing over is that this construction is clearly polynomial in size and has a very repetitive structure. So, we can build this in polynomial time for sure. As an exercise, you can count how many vertices are there and understand why this instance can be generated in polynomial time given the Boolean formula.

What remains is to show the correspondence: phi is satisfiable if and only if this graph has an s to t Hamiltonian path, including the clause vertices by the side. Suppose the formula is satisfiable, meaning you can set variables to true or false such that every clause has a true literal.

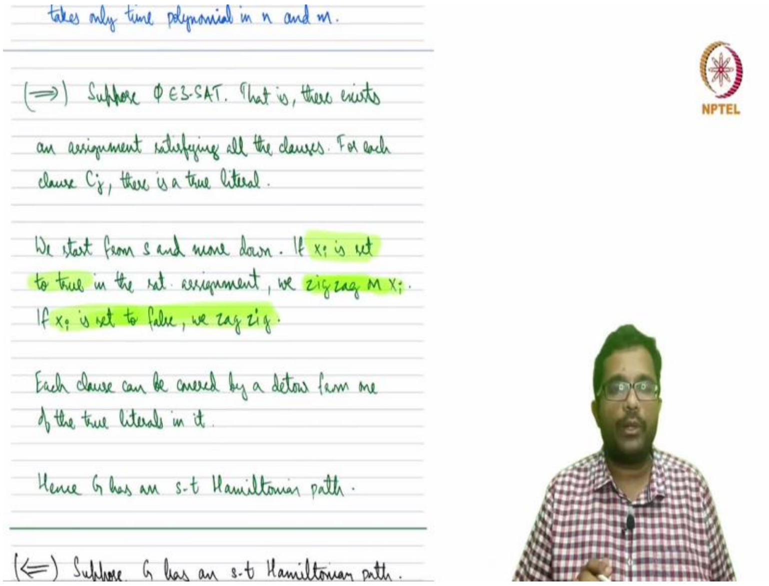
Now, we need to show that the graph has a Hamiltonian path. Suppose the assignment satisfies the formula. If x1 is true, we zigzag. If x1 is false, we zagzig. Now, how do we cover the clauses?

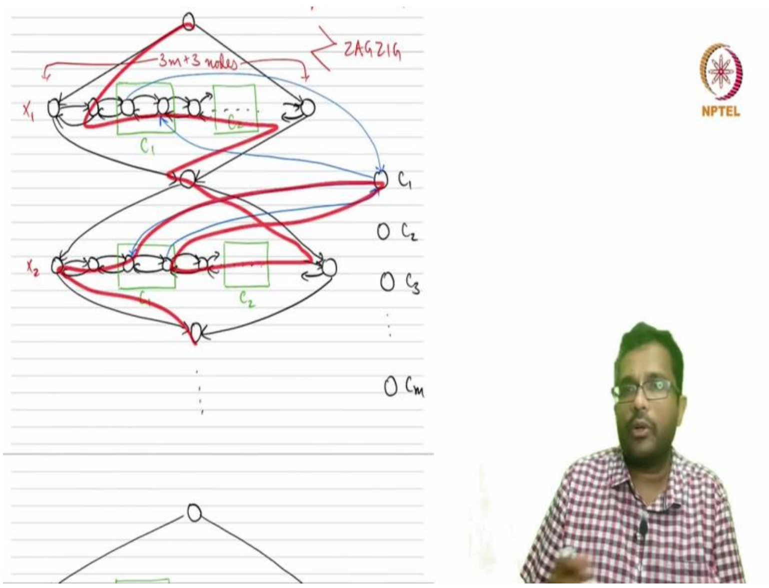
Suppose c1 contains x1. If c1 contains a true literal, and x1 is true in the satisfying assignment, we zigzag. We come down the left side, and within the zigzag, we include c1. If x1 is false, we zagzig and cannot cover c1. But if c1 is covered by x2 complement, then x2 complement must be true, meaning x2 is set to false in the satisfying assignment, and we zagzig in x2. When we zagzig, we naturally visit c1 and come back, continuing our zagzig journey.

Since it is a satisfying assignment, every clause has a true literal. When we reach that literal, we detour to the clause vertex and come back. The way the variables and the gadget are set up allows us to detour to the clause vertex and then return to the next vertex without missing anything, continuing the zigzag or zagzig path.

Finally, we keep going down, and eventually, we reach t. In the meantime, we will have covered all the variables and all the clauses because each clause is satisfied by some variable. Whenever we cross the variable or literal that satisfies a clause, we make sure to cover that clause. Thus, all these clause gadgets will be covered.

(Refer Slide Time: 25:19)





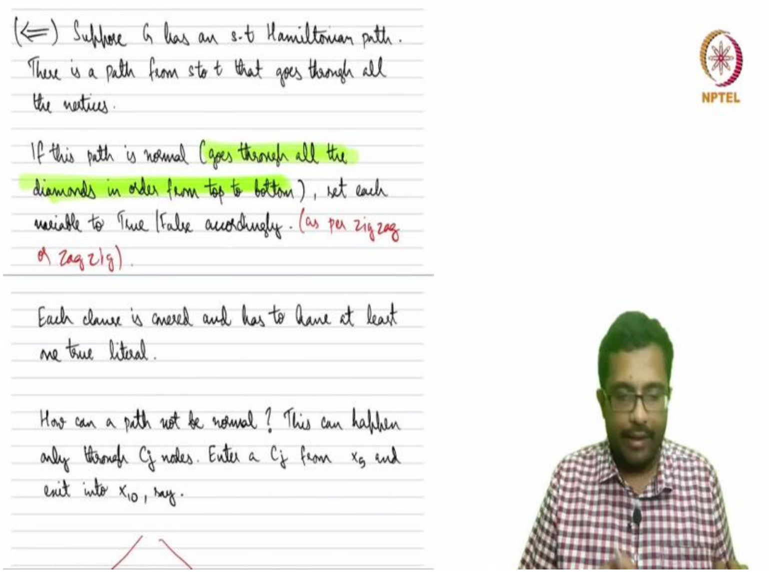
Let us say the same thing: there is a true literal for each clause. We start from s and move down. If xi is set to true, we zigzag; if xi is set to false, we zagzig. Each clause can be covered by a detour, which gives a Hamiltonian path.

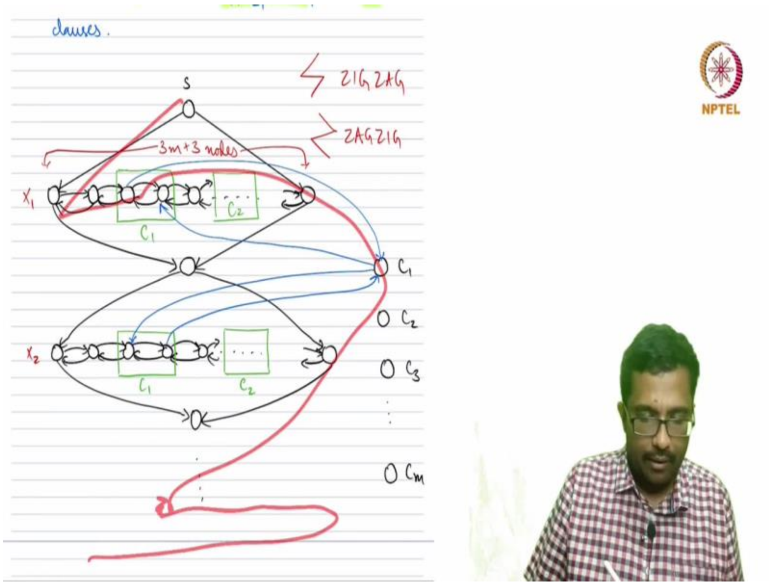
Now, the opposite direction: suppose there is a Hamiltonian path. How do we show that there is a satisfying assignment? Suppose the Hamiltonian path works as described in the forward direction: it zigzags or zagzigs, and whenever there is a clause, we make a detour and immediately jump back and continue. If this happens, it is fairly clear.

If this is the way the path is—zigzagging or zagzigging, and making detours to cover all the clauses—then we set a variable to true if it is zigzagging, and set it to false if it is zagzigging. By the same argument as in the forward direction, this implies that because we cover all the clause vertices, every clause is satisfied. We can only cover a clause vertex if we are moving in the correct direction. Thus, if zigzag means true and zagzig means false, we end up with a satisfying assignment because we are able to cover all the clauses.

To cover a certain clause c from a certain variable xi, we should either be zigzagging on xi with xi present in c, setting xi to true, or zagzigging on xi with xi complement present in c, setting xi to false. If the path is of this form—starting from the top, making a detour, coming back to the same diamond, and continuing—then we know how to generate the satisfying assignment: set the zigzagging variables to true and the zagzigging variables to false.

(Refer Slide Time: 28:00)





Suppose G has an s to t Hamiltonian path. If the path is normal—meaning it goes through all the diamond vertices in order from top to bottom—then we set each variable to true or false as per the zigzag or zagzig pattern. As explained, this means that all the clauses are satisfied.

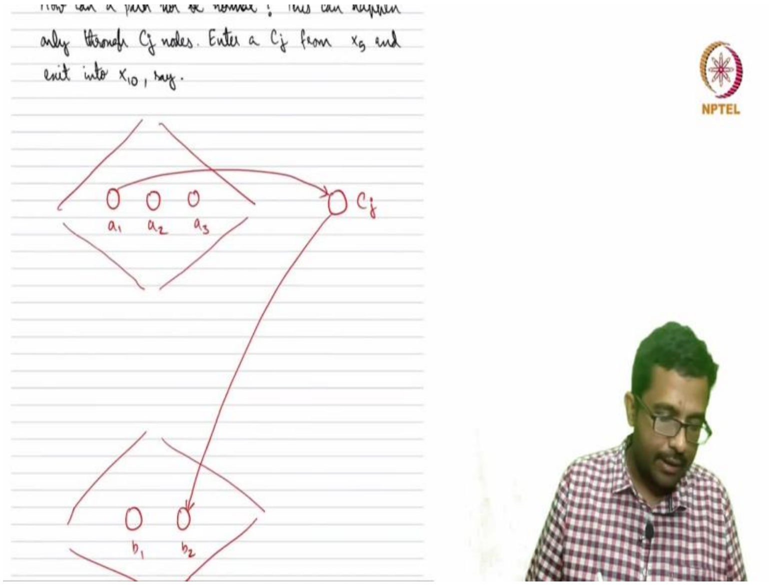
Now, there is an assumption that if this path is normal, everything is fine. But is there a way in which the path may not be normal? The only way a path may not be normal is if we start at the top, go to a clause, then instead of reentering back to the first diamond, we reenter some other diamond, continue for a while, and then maybe come back to the first diamond and to another clause. This is possible.

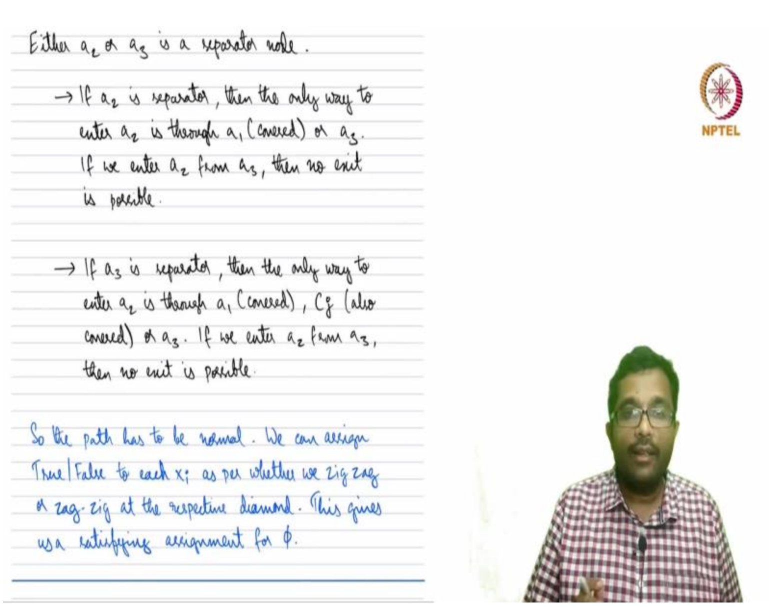
So, if the Hamiltonian path takes the normal standard zigzag or zagzig pattern with detours, we know it satisfies the clauses. The question is, what if the path is not normal? What if it goes somewhere else and then comes back later? How do we determine if it is zigzag or zagzig?

I argue that such non-normal paths—where you enter a clause, go to another diamond, and later return to the original diamond—cannot happen. This is because we have buffer vertices between these clause boxes. These buffer vertices play a crucial role.

Let's see how. I have a zoomed-in figure to explain this.

(Refer Slide Time: 30:17)





How can a path not be normal? That can happen if we leave a certain diamond, go to a clause, and then reenter some other diamond. Let’s say you exit the fifth diamond and enter the tenth diamond, and later maybe from the fifteenth diamond, you go back to the fifth diamond. Can this happen? We argue that if this is attempted, it will not be a Hamiltonian Path. Let's see why.

If we have this arrow from a1 to Cj, there are two possibilities. One is that this diamond a1 and a2 are grouped together, and a3 is a separator. At some point, a2 needs to be covered if it is a Hamiltonian path. If this box corresponds to clause Cj, it means there is an arrow coming from Cj here, and also a3 connects to a2.

Now, if we enter a2 through a3, there is an issue because there is no next step. The neighbors of a2 are a3, from which we just came, a1, which is already covered, or Cj, which is also already covered. So there is no exit out of a2. These two cannot be in one block. The other possibility is a1 and the vertex to its left form a block, and a2 is a separator vertex.

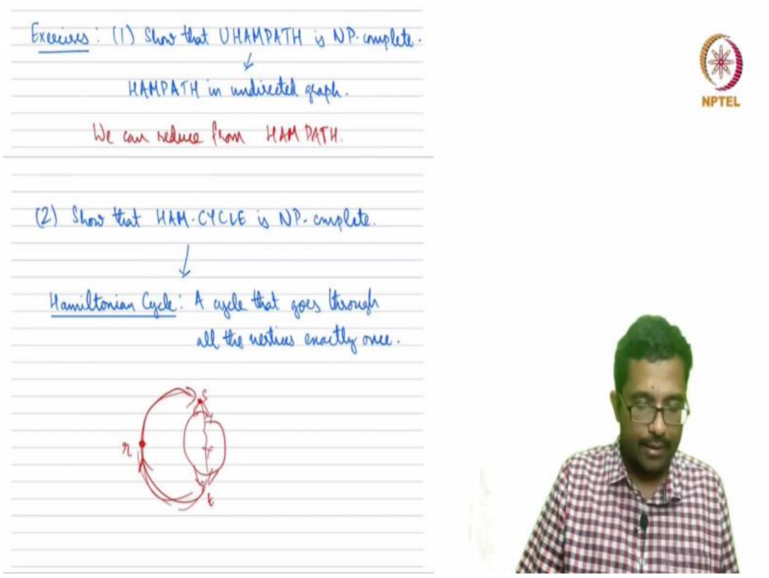
If a2 is a separator vertex, it is not connected to any clause vertex. The only way we can reenter a2 is through a1 or a3. But a1 has already been visited. If we enter through a3, there is no way out again. So if a2 is a separator, entering through a3 means we can only go back to a1, which is already covered, or to a3, from which we just came. Thus, a2 to a3 has no exit.

This shows that regardless of whether a2 or a3 is a separator, we cannot leave a diamond and then enter some other diamond; this will not form a Hamiltonian path. If we try to cover it, we cannot come out and reach the end. There is no exit option.

Hence, the only possible Hamiltonian paths are the ones where we start at the top, zigzag or zagzig a diamond, and then, wherever necessary, go to a clause and immediately rejoin and continue. For these paths, we know how to handle it: if it is zigzag, we assign true; if it is zagzig, we assign false. This results in a satisfying assignment.

Thus, if there is a Hamiltonian path, it has to follow this normal structure, giving us a satisfying assignment. This completes both directions. If the formula is satisfiable, we have a Hamiltonian path. If the graph has a Hamiltonian path, we have a satisfying assignment for the formula. This completes the correspondence and shows that Hamiltonian path is NP-complete.

(Refer Slide Time: 35:11)



Now, two more small things before I end this lecture: one is that we can take the undirected Hamiltonian path. So far, we talked about the directed Hamiltonian path, but the undirected Hamiltonian path is the same problem in an undirected graph. This is also NP-complete. In undirected graphs, we do not have directed edges, so the direction does not play a key role. However, we can reduce from the directed version of the problem. This is explained in the book; essentially, we replace each vertex with three vertices or something similar, ensuring that if a directed graph has a Hamiltonian path, the constructed undirected graph also has a Hamiltonian path, and vice versa.

The second thing is the Hamiltonian cycle. What is the Hamiltonian cycle? It is a directed cycle that goes through all the vertices exactly once, with no beginning or end. This is also NP-complete. This is not covered in the book, but I encourage you to think about it. One approach is to take a graph with an s to t Hamiltonian path and add a directed edge from t to s. If the original graph has an s to t Hamiltonian path, this new edge forms a Hamiltonian cycle.

However, a Hamiltonian cycle does not necessarily imply an s to t Hamiltonian path, as the cycle may not use the new edge. To ensure an s to t Hamiltonian path, we can add an intermediate vertex, r, with directed edges from t to r and r to s. If the new graph has a Hamiltonian cycle, it must include an s to t Hamiltonian path because there is no other way to cover r. This is the basic idea, but I leave it to you to think about and work out the details.

That is it for lecture number 53, where we saw that the Hamiltonian path is NP-complete, with a reduction from 3-SAT using an interesting construction. In the next lecture, we will see yet another NP-complete problem. Thank you.