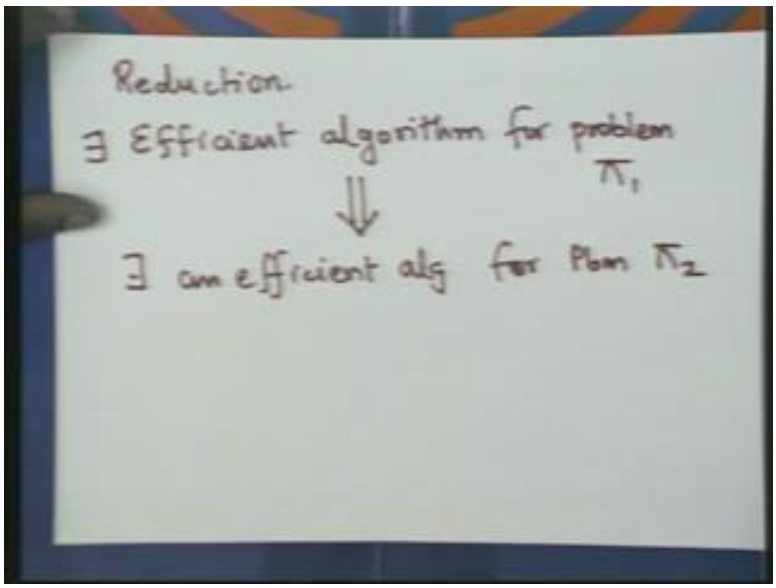
Design and Analysis of Algorithms
Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institution of Technology, Bombay

Lecture - 27
NP-Completeness – 2

(Refer Slide Time: 01:48)



Last time we looked at the concept of reduction. So let me just quickly review, what we mean by reduction? So, reduction means this given an efficient algorithm for a problem pi 1. So, you are given a efficient algorithm for problem pi 1. Then using this, we design an efficient algorithm for a different problem. So, we essentially showed that this implies that there exists an efficient algorithm for problem pi 2. So, given that there exists an efficient algorithm for problem pi 1.

So, one problem we show that there is an efficient algorithm for problem pi 2. The way we do it is imagine that this pi 1 somebody has coded for pi 1 an algorithm for pi 1 and it exists in some library. Then, we use this to design an algorithm for problem pi 2. So, this algorithm that we design it takes as input for problem pi 2. It perhaps does something to the input and then calls this sub routine for an algorithm pi 1 may be once may be twice repeatedly.

And at the end of this it outputs an answer and this answer is for problem pi 2. So, essentially we have solved problem pi 2 assuming that somebody has given a solution for

problem pi 1. The two problems, we looked at last time was pi 1 was given a graph. Does, there exist a perfect matching in the graph that was pi 1. And the algorithm, we designed was given a graph output a matching of maximum size that was pi 2.

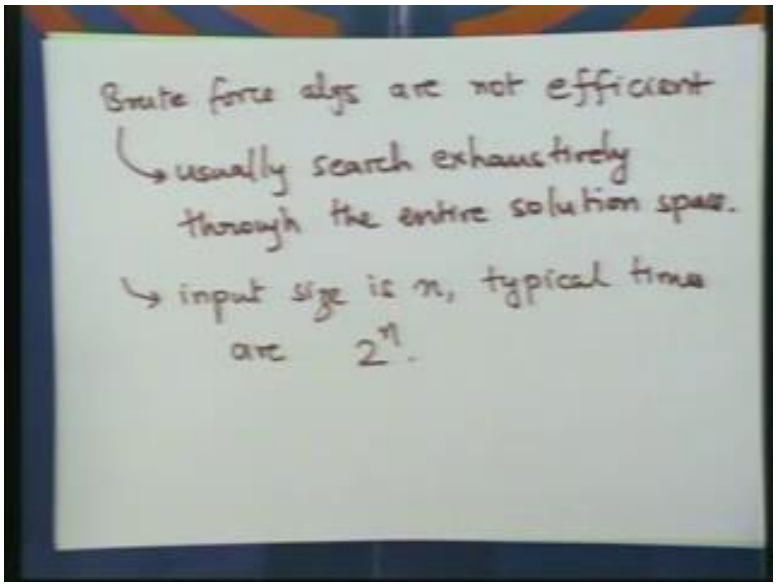
And we did in fact, repeatedly call this pi 1 after modifying the inputs likely. So, why is this called the reduction? It is called the reduction in the following sense that you actually want to call, you actually want to solve a problem pi 2. And you have somehow reduced this problem solving the problem pi 1.

In the sense that you can solve the problem pi 1, you can solve problem pi 2. So, you have reduced the problem pi 2 to a problem pi 1. And that is how the word reduction comes. But, the essential method is this given an efficient problem algorithm for one problem find an efficient algorithm for another problem. And you have to use the algorithm for the problem given to construct this efficient algorithm. So, this was a concept of reduction.

We will in fact, see one more example of this very shortly. Before, we see an example here is a word which seems a bit new which is efficient. Problems have been defined, we have seen many problems, algorithms have been defined. We have seen many algorithms well what does this efficient means.

So, let me define what I mean by efficient? We had seen earlier a solution to one of these scheduling problems, where the time taken was just too much? That was a inefficient algorithm. An efficient algorithm, we would like to be something that on reasonable size inputs finishes in reasonable amount of time.

(Refer Slide Time: 05:03)



So, let me tell you algorithms which are not efficient to start with. So, these are usually brute force algorithms are not efficient. So, what do you mean by brute force algorithms? Brute force algorithms are algorithms that look at all possible solution sets. There may be many possible solutions given your input there could be many possible solutions. And you want to pick one which is good or the best.

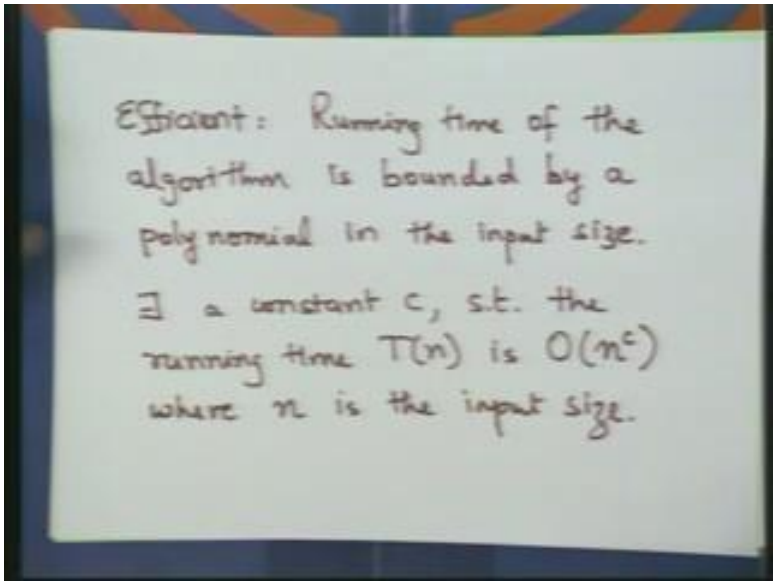
So, the brute force algorithm would look at all possible ways of doing this and pick the best one. For instance, if you want to look for a matching of maximum size a brute force algorithm would look at all possible collection of edges. All possible subsets of edges, then check whether each subset is a matching and also the size. And from this you can certainly find out a matching of maximum size. But this algorithm takes too much time.

So, if there are m edges then you take 2 to the m, you have to look at 2 to the m subsets and we have seen that this is just too much. So, these usually search exhaustively through the entire solution space. And if the input is of length n that is the other sort of characteristics of this brute force algorithm. The input size is n typical times taken by these brute force algorithms is 2 to the n, because we have just looked at all subsets. Typical times are 2 to the n and this is just too much.

And we saw that you design such algorithms you may get fired. So, these are not the algorithms that we are looking for these are not efficient, what do I mean by that. So, by

efficient I mean that the running time of the algorithm is bounded by the polynomial in the input size.

(Refer Slide Time: 07:38)

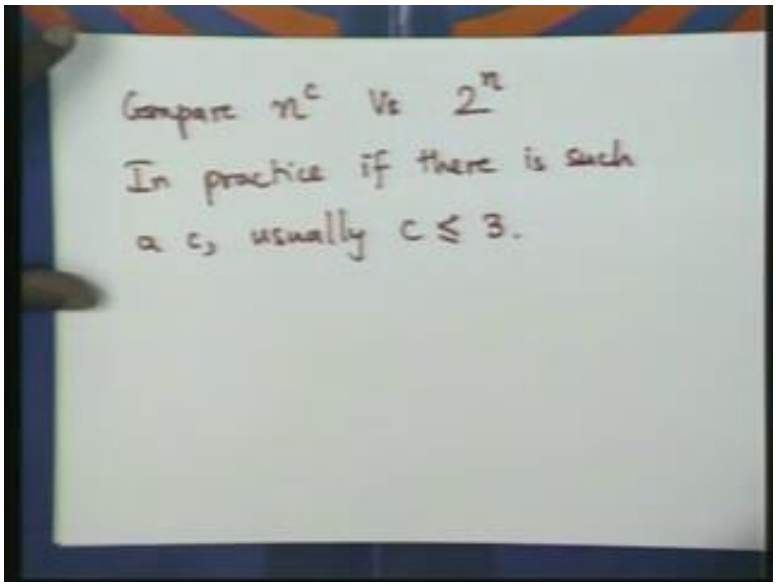


Let me write this down for us this would mean this, the running time of the algorithm is bounded by a polynomial in the input size. So let me restate this again, what do I mean by polynomial? The running time which is bounded by let us say n to the constant. So let me, just state this again which means I am just going to restate this. There exists a constant c such that the running time T of n is big O of n to the c, where n is the input size.

So this is, what efficient for us will mean? Most of the algorithms, we studied so far are efficient in this way. For instance sorting, we can do in n log n times. That certainly bounded by a polynomial n square bounded by n square. The other algorithms which you have studied in your divide and conquer or dynamic programming they are all bounded by a polynomial in the input size n square n cube. Shortest path finding minimum spanning trees all most any algorithm that you have studied so far.

They are all their running times are all bounded by a polynomial in the input size. And these are the algorithms we have studied so far are efficient, why polynomial, why is the notion of efficiency, why not some other function of n.

(Refer Slide Time: 10:19)



So, the reason is this, if you compare n to the c if you compare n to the c. And let us say 2 to the n. So, these are 2 to the n is what brute force algorithm would take and n to the c is what our notion of efficiency is will be. If you look at these two functions then 2 to the n grows just much faster than n to the c. So, even for inputs of size 1000 square and 2 to the 1000 are just two different things.

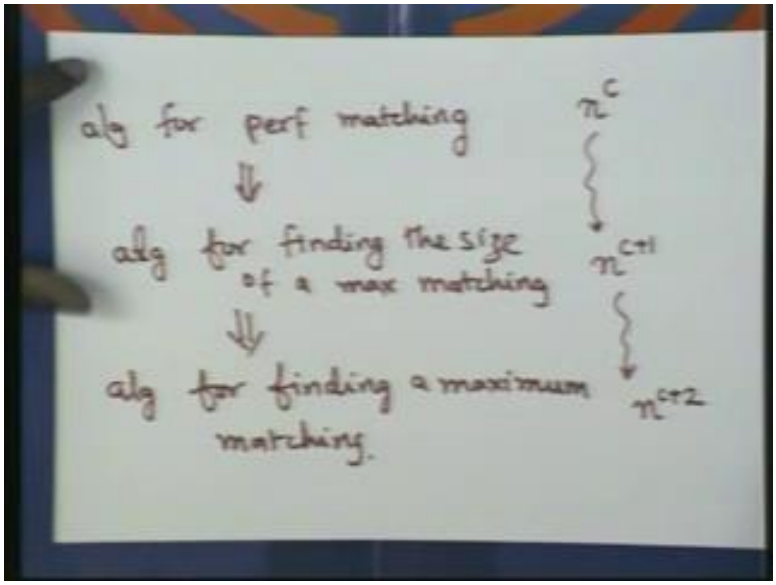
If you have something which is let us say n square then if you program an algorithm, which runs in time 1000 square that will take a few minutes to complete. If your algorithm takes 2 to the 1000. Well, it is not going to stop at least in our life times may be much more. So that is why, this motion of efficiency has been is prevalent.

The other sort of reason is the in practice usually when the problem has an algorithm which runs an polynomial in input size. So usually in practice, if there is a constant, so that you can bound the running time by n to the c. This c happens to be very small 2 or 3, there are very few algorithms, whose running times are more than n cube.

If there is a polynomial time algorithm for this problem which means the running time is bounded by a polynomial. Then, using this polynomial is small like n n square n cube log n. So in practice, if there is such a c, usually less than equal to 3, so that is why one uses this notion of efficiency and this is the notion that we will use for the rest of this course

((Refer Time 1:48)). We said that if there is an efficient algorithm for problem p i 1, there is an efficient algorithm for problem pi 2.

(Refer Slide Time 12:54)

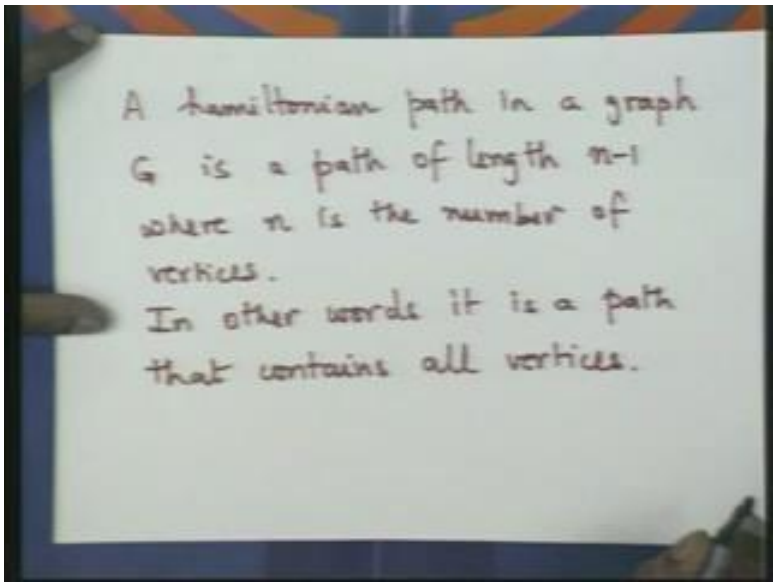


So, let us again look at the matching example that we did pi 1. So, we said that lets assume that there is an efficient algorithm for perfect matching. So this, we said implied an algorithm for finding the size of the maximum matching and this implied an algorithm for finding a maximum matching.

This is what we saw last time? Now even though this algorithm, it is efficient. It is not necessary that when you do this transformation. The new algorithm is efficient. In this case, it actually is true the reason is this algorithm is called at most n times. So, the running time were let us say n to the c this was called at most n times running time here is roughly n to the c plus 1. And this called this at most n times, so this turns out to be n to the c plus 2.

So, if there is a c if there is a constant c. So that this running time is bounded by n to the c. There is some other c possibly may be c plus 2 2 c. So, that the new algorithm runs in time n to the c. So that is what I mean by saying that if there is an efficient algorithm for problem pi 1. There is a efficient algorithm for problem pi 2.

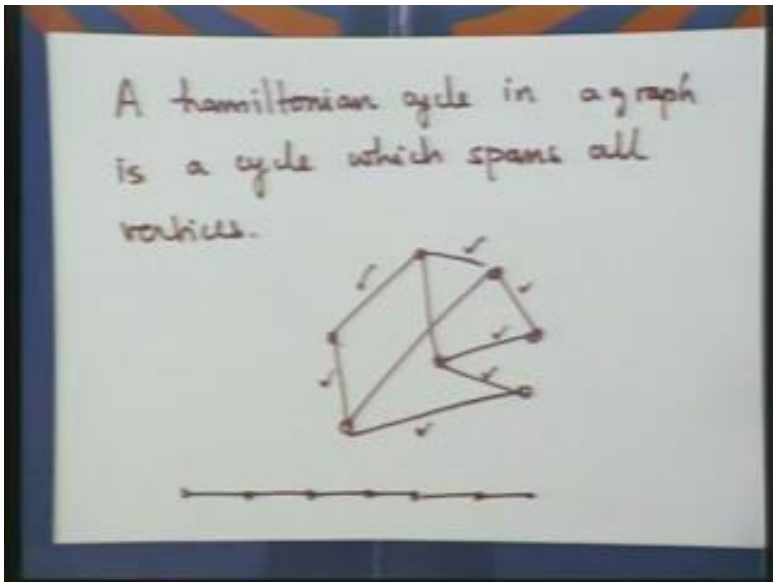
(Refer Slide Time: 14:54)



For our next problem I need to define a couple of terms. These are concepts of Hamiltonian path in Hamiltonian cycle some of you have seen this before, but let me define this anyway. So, a Hamiltonian path in a graph G is a path of length n minus 1, where n is the number of vertices. So, path of length n minus 1, where n is the number of vertices. In other words it is a path which spans all vertices.

Other words, it is a path that contains all vertices. So, it must be a path in the graph and all vertices in the graph must be present. Clearly, the length is n minus 1, the number of edges which is length n minus 1, this is Hamiltonian path. The Hamiltonian cycle is very similar; it is a cycle which spans all vertices one single cycle which spans all vertices.

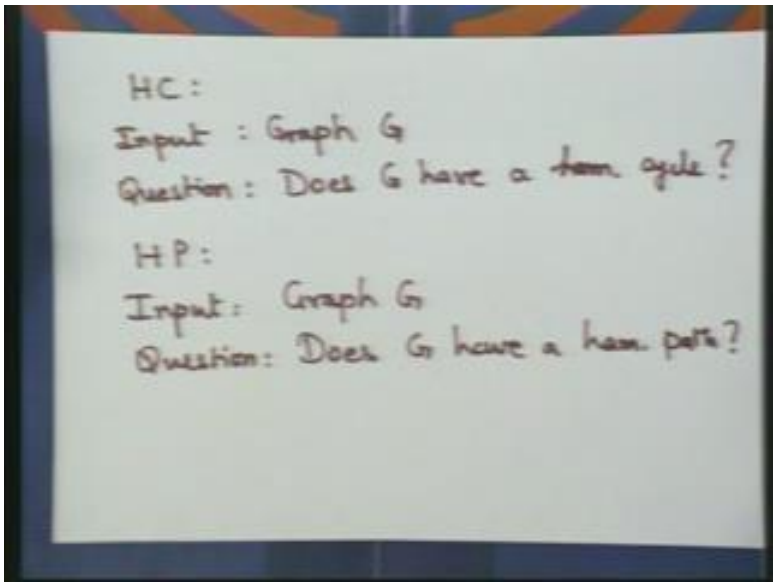
(Refer Slide Time: 16:30)



Let me write this down to a Hamiltonian cycle in a graph is a cycle which spans all vertices. So, let us look at an example, so supposing this is a graph. Now, here is a Hamiltonian cycle in the graph, this edge, this edge, that edge this edge. So, this is a cycle which has all the vertices in the graph. And if I remove any one of these edges it gives a Hamiltonian path in the graph.

Any one of these edges in the Hamiltonian cycle, this gives a Hamiltonian path in the graph. So, if a graph has a Hamiltonian cycle, it has Hamiltonian path. The reverse of course may not be true. So for instance, the graph is just a path. If the graph is this path then it has a Hamiltonian path and it has no cycle.

(Refer Slide Time: 18:43)



So now we have defined these two terms, let us put them to use. We are talking of reductions. So we are going to see, how closely are these two problems or two problems that I am going to define right now related. One is the problem of finding the Hamiltonian paths and the other is the problem of finding the Hamiltonian cycle.

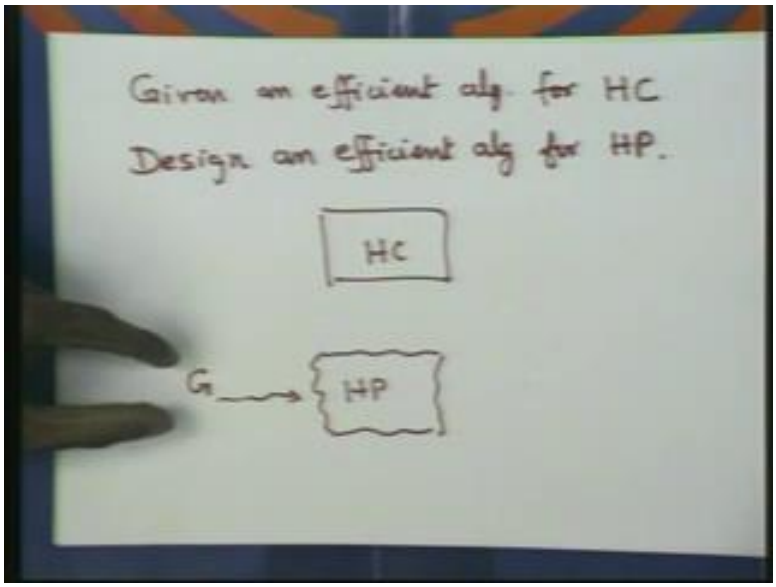
So, the first problem is called Hamiltonian cycle. Let us see, it is this input is a graph G. The questions you ask Does G have a Hamiltonian cycle? This is problem HC. Similarly, I have problem HP not to be confused with Hewlett Packard. And the input to the graph G, the question Does G have Hamiltonian path? So for this problem, you want an algorithm that says you fit in a graph G.

And this algorithm should say either yes the Hamiltonian path or not the graph does not have that is for this problem. For this problem, it should say yes put the graph as Hamiltonian cycle otherwise No the graph does not have Hamiltonian cycle. So, the input is a graph, the output is yes or no, yes if it has this structure either the path either the Hamiltonian path or cycle no, it have.

So, these are the two problems and clearly they seem to be related I mean Hamiltonian cycle is a cycle that spans all vertices. Hamiltonian path is a path that spans all vertices. So, one would expect these two problems to be similar in some way. In fact similar now, we have a clear notion of what similar must be; we want to say something like this.

If we can solve HC which means if there is an efficient algorithm for HC, there is a efficient algorithm for HP. And there is a efficient algorithm for HP, there is a efficient algorithm for HC.

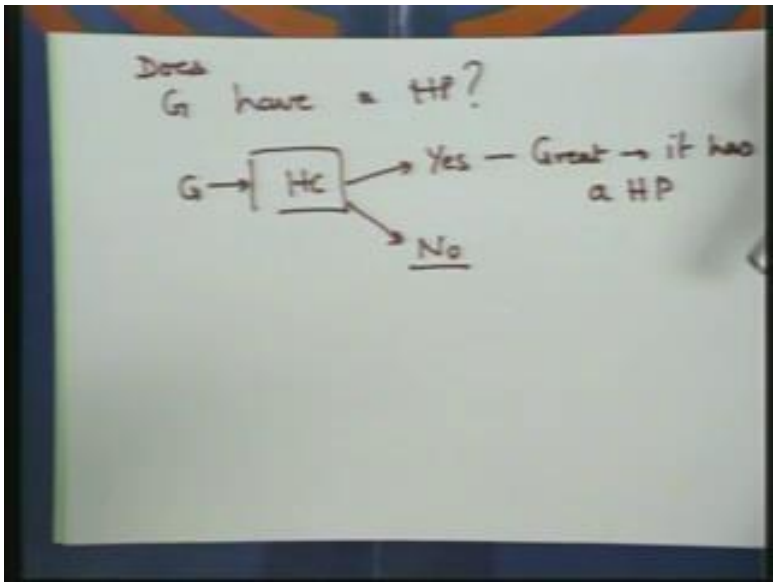
(Refer Slide Time: 21:10)



So, these are the questions that we are going to address now an efficient algorithm for HC which is Hamiltonian cycle. Design a efficient algorithm for HP, this is Hamiltonian path. So, how do we do this? So here is my here is the algorithm HC. If I feed in a graph to this algorithm it will say whether or not it has a Hamiltonian cycle. Now, what I want to design is one for HP. I want this feed in a graph this would say yes if it has a Hamiltonian. So, supposing the input is some graph G.

I want to determine if this graph has a Hamiltonian path or not. And I have to use this algorithm somehow. Supposing I feed the same graph into HC, two things can happen HC can say yes or it can say no. So, let us try this let us see what happens in each of these two phases.

(Refer Slide Time: 22:42)

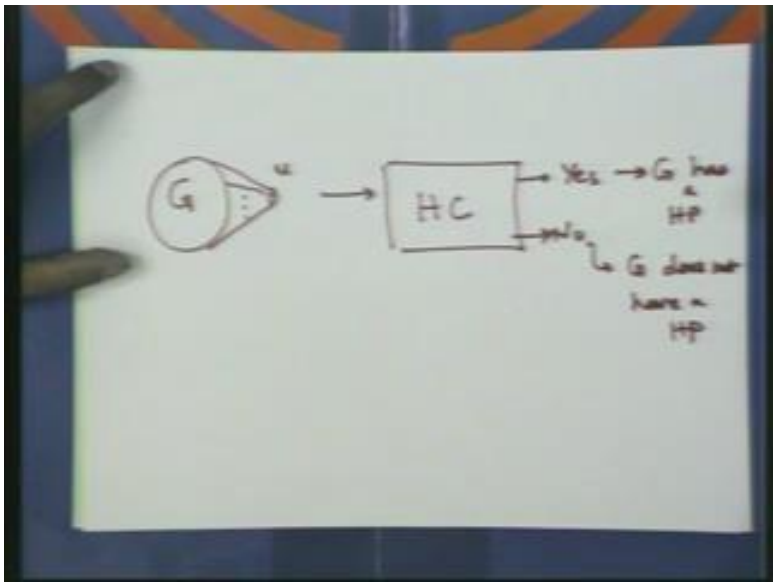


I want to determine if G has a Hamiltonian path, does G have a HP? This is the question. So, I feed G into HC. There are two cases this fellow can say yes and what do you think you can conclude. If it has a Hamiltonian cycle does it have a Hamiltonian path? Well absolutely it must have a Hamiltonian path. So, if G says this HC says yes for G you are done great it has HP. So, you can say with surety that if it has a Hamiltonian circuit it has Hamiltonian path.

What if it says No? Now, we really do not know may be it does not have an HP in which case it does not have a HC and so we are all fine. The problem is the graph could have a Hamiltonian path, but it need not have a Hamiltonian cycle. So that case, we are not just able to distinguish using just a blind way of using this sub routine. So, we have to be a little bit smarter in using this sub routines and that is the duty of the subject.

You have to figure out how exactly do I use? This sub routine HC effectively to get a Hamiltonian path. This is what you do? You should may be try it you see the solution I am going to give right now. In fact, these transformations to the graph that I am going to do you have seen earlier it is a hint. So, what we do is this.

(Refer Slide Time: 24:52)

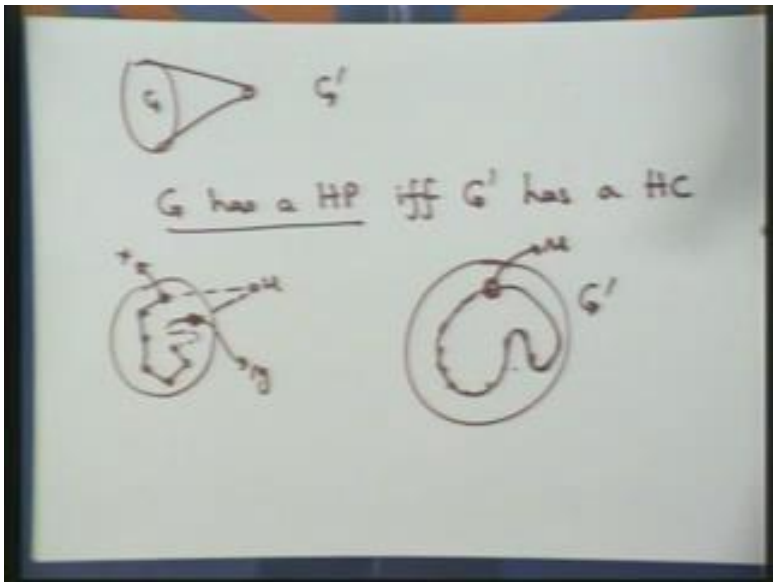


So here is G, I take G as it is I added this is G, I added new vertex u connect u to everything in G, u is connected to every vertex in G. Now, this graph I feed into HC I have an original graph where I want to determine if it has a Hamiltonian path or not. I add a vertex I connect it to every other vertex in G and feed it into HC.

Now, if HC says it could say yes or no. So, what I am going to do is if HC says yes then I will also say yes, the graph G has a Hamiltonian path. If it says no, I say G does not if HC says yes then the output of the algorithm for HP will say yes. If HC says no I will say no on this new graph; however, it is on the new graph.

So, does this work does not this work. Well it does work we will see why there are 2 things we have to show we have to show that if it answers yes. This answer must be correct which means this graph G must have a Hamiltonian path. And if it answers no the graph should not have.

(Refer Slide Time: 26:27)



So, the two statements that we need to prove is the following are the following here is G vertex u is adjacent to every other vertex in G. So, call this new graph G prime new graph is G prime. So, I want to show that G has a HP if and only if G prime has HC. This is an if and only if statement, so if there are two things. The two things correspond to yes in output. For instance, we have to show that G has an HP this implies that G prime has a HP.

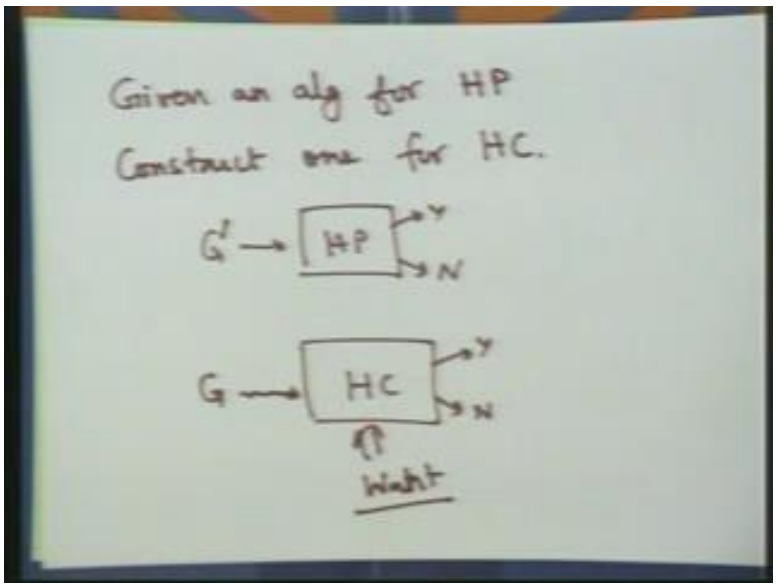
And the no output G does not have a HP this implies that G prime does not have HC. So, these are the two things. So, assume that G has a HP, we need to show that G prime has an HP. And I think this should be fairly obvious, there is G here is my HP. I do not know how this looks; it is a path which runs between, which has all the vertices in G.

Now, I need to construct an HC which is a Hamiltonian cycle in G prime and that is easy I just add these two edges. So, this path starts at some vertex and it ends at some vertex. It starts at x and ends at y, the Hamiltonian cycle, I get by just appending u to x and y. I add the edges u x and u y and I get a cycle in G prime.

And you can see that I can go the other way also. So, given the Hamiltonian cycle in G prime I can construct a path a Hamiltonian path in G and the construction is similar. So here, supposing this is graph G prime and I have a Hamiltonian cycle it goes around like this in G prime.

Now, this vertex u sits somewhere, this is my vertex u. If I remove this vertex u from the graph what I am left with is the Hamiltonian path in G. This path will start from this vertex x and will end at this vertex y. So, the path starts at x it will go around and end at y and this path is in the graph G. So, we have proved statement both ways that if the original graph had a Hamiltonian path the new graph have a Hamiltonian cycle. If the original graph did not have a Hamiltonian path the new graph will not have a Hamiltonian cycle. What about the other way round? We said given an algorithm for Hamiltonian cycle. We could construct one for Hamiltonian path.

(Refer Slide Time: 29:47)

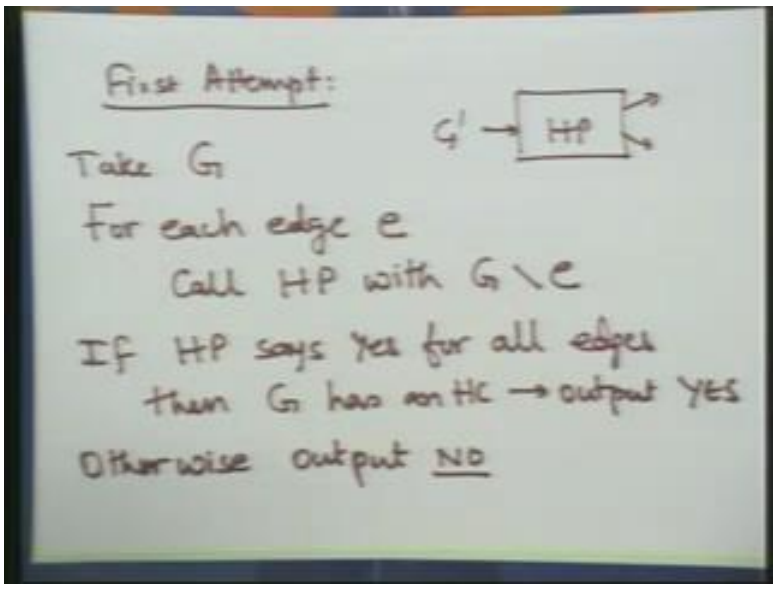


What about the other way round? The other way round would be this given an algorithm for Hamiltonian path constructs one for HC and from now on it, if you are given a algorithm it is efficient. And you want to construct one that better be a HP. So, these are all well I just said algorithm, we want efficient algorithm. The previous one was clearly efficient. I had to call the Hamiltonian cycle routine exactly once.

I do a small modification to the graph the input size does not go up by much and I call the Hamiltonian cycle routine exactly once. So that running time is roughly this the old algorithm was sufficient the new one was. So, how about this? So, you are given a algorithm for HP which means if you are given a graph G it says yes or no and what I want is for HC? The given let me call this G prime to distinguish from this G. So, this is what we want, this is what we want this is given to us.

So, somehow again I have to use this routine for HP efficiently to get this routine for Hamiltonian cycle. So, how do I do? I could let us try the usual trick feed this graph into HP and it says yes. If it says yes then well it could have a HC or it may not have an Hamiltonian cycle. On the other hand if it said no then I am here that the graph does not have the Hamiltonian cycle. In fact, does not have a Hamiltonian part. So, if it says no I have no we are all fine, but if it says yes, again we run into a similar problem. We cannot decide whether it has a Hamiltonian circuit or not.

(Refer Slide Time: 32:20)



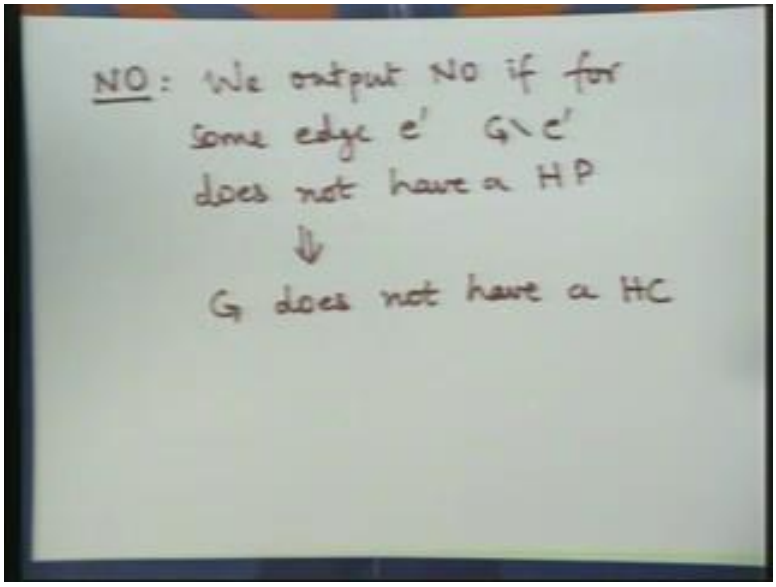
So, what do we do? So, here is there is an attempt take the graph G. So, I want to use let me just put this on the side. So this is, what I want to use and I want to create one for HP. So, what do I feed into HP. So, take this graph there is a input graph G. So, take G now remove edges from G one by one. So, let us say I remove edge e. Now I feed into G prime and I will see whether it says yes or no.

Now, if it says no for any edge, then clearly the graph will not have a Hamiltonian cycle, why is this? So, supposing G has a Hamiltonian cycle then if I remove any edge from G, the graph still has a Hamiltonian path. So, if I take G and remove any edge the Hamiltonian paths must keep saying yes. Is this a good algorithm I mean can I sort of say that remove every edge from G feed into HP? If it says yes all the time then yes G has a Hamiltonian cycle. If at least once it says no then G does not have Hamiltonian cycle.

Let me write this algorithm. For each edge e call HP with G minus e. If HP says yes for all edges then G has an HC. So, you output yes otherwise output no. So, if it says if HP says no for any input then you say no. So, this is the algorithm. Well the question is does this work what do we mean by this question what does it mean for this to work or not.

It means that whenever you output yes. So, whenever you output yes the graph better have a Hamiltonian cycle. And whenever you output no the graph should not have a Hamiltonian cycle. Let us take both of these in turn and see whether we can prove this.

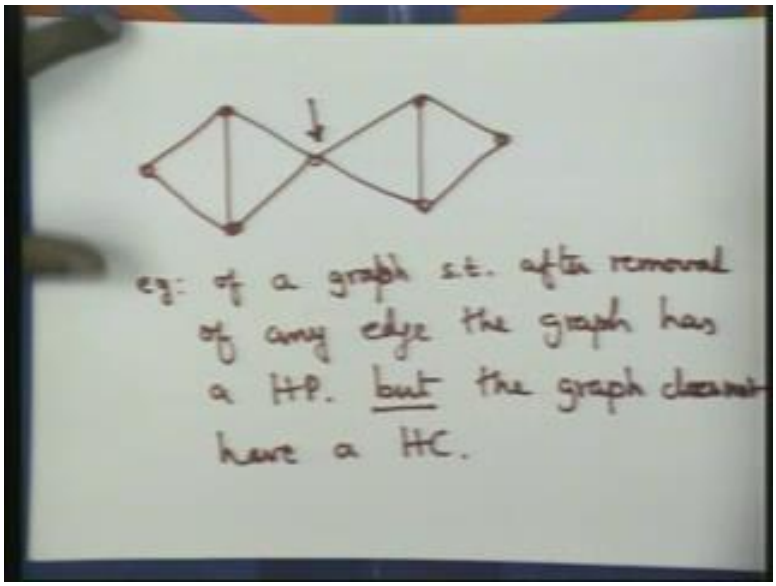
(Refer Slide Time: 35:29)



So, let us take no first, so that is easier. So, you output no if for some edge e prime G minus e prime does not have a Hamiltonian path. So, if I remove this edge e prime then the graph does not have a Hamiltonian path. We have seen that this implies that G does not have Hamiltonian cycle. If it had a Hamiltonian cycle then if I removed any edge out I would still have a Hamiltonian path in the resultant graph.

So whenever we output no, we are in the clears we are correct the algorithm is always correct, what about the yes case? It seems very reasonable that for every edge if I remove it. And ask there is a Hamiltonian path and it says yes there should be reasonable that there should be a Hamiltonian cycle. So, if there is a Hamiltonian path after removal of any edge is a good chance that one feels that graph should have a Hamiltonian cycle, but this is false. So, this statement is false.

(Refer Slide Time 37:11)

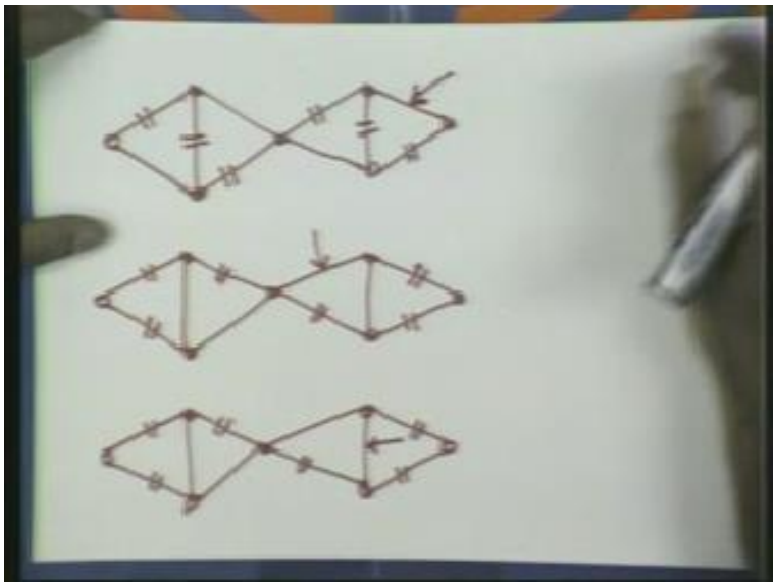


Let us see an example, so here is an example. So, this is an example of a graph such that removal of after removal of any edge graph has a Hamiltonian path, but the graph does not have Hamiltonian cycle. So, there are two things to be checked with this graph. That it does not have a Hamiltonian cycle, but if I remove any edge from this graph any edge at all then this graph must have a Hamiltonian path. Let us check both of them.

Now, this graph does not have a Hamiltonian cycle and to see that we focus on this vertex in the middle. If I remove this vertex, then this graph becomes disconnected. There is a vertex in this graph. So that if I remove this middle vertex and the graph becomes disconnected. This cannot happen if the graph has a Hamiltonian cycle. If the graph has a Hamiltonian cycle, if I remove any vertex the resultant graph will have a Hamiltonian path and will in fact be connected.

So, if a graph has a Hamiltonian cycle if I remove any vertex it must remain connected that is not true for this graph. That is the reason why this does not have a Hamiltonian cycle, we still have to prove one more thing which is that if I remove any of these edges in this graph then it should not have it must have a Hamiltonian path.

(Refer Slide Time: 39:26)



So, let us look at these edges one by one. Let me draw this graph again. So, let us focus on this edge supposing I remove this edge. Does there exist a Hamiltonian path? Well the answer is yes choose this edge choose that edge choose this edge. You come down this way then you choose this edge. Then, you go down this way to this edge that edge. There are other ways of doing this.

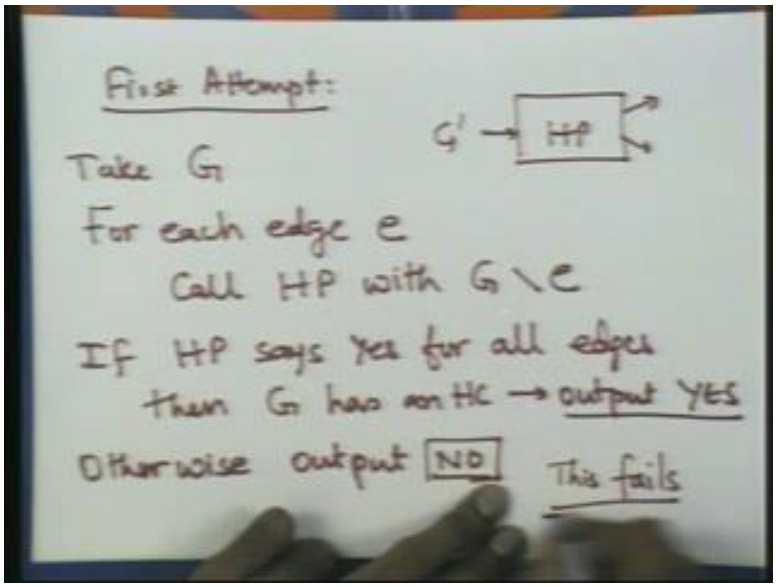
So, you go up and down like this and this you can see the Hamiltonian path. So, removal of this edge here causes no problem. So, let us look at some other edges. So, let us remove this edge what if I remove this edge? So in this case, I can start here come down here go up here there it says this and this. So, this will give a Hamiltonian path. So, you start here and you go up and down like this go up go down up to this and this you can see gives a Hamiltonian path.

So, removal of this edge is also not a problem. Let us look at one more lets remove this edge. If I remove this edge, the graph still has the Hamiltonian path this this this this. So, you go down this way go back keep going down all the way up here go down and back here. So, if I remove this edge too there is a Hamiltonian path. Now we just see that all edges in the graph are similar to one of these three.

For instance this edge is similar to that right this edge at the bottom is similar to this. So, all edges on this side is taken care of. These two edges are taken care of, this is the middle edge these two are similar. And the graph is symmetric about this vertex. So, all

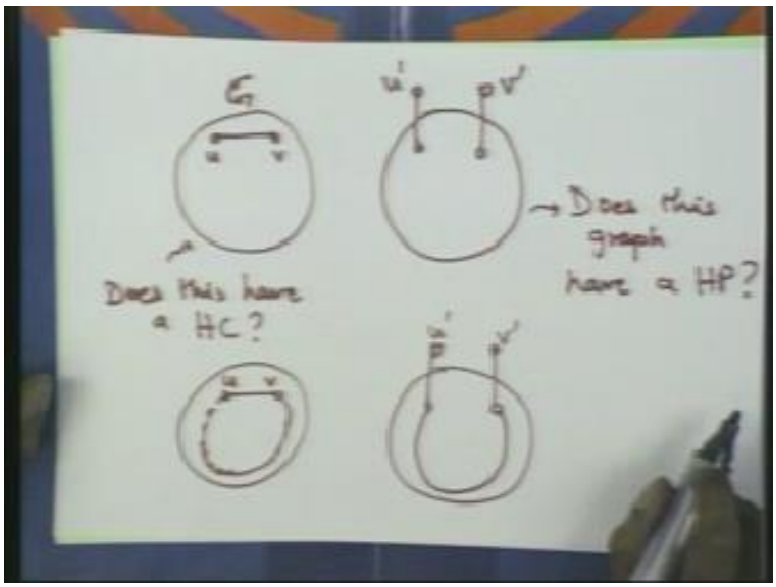
edges on this side on the left hand side are also taken care of. So, these three cases are enough to enumerate to sort of hopefully convince you that removal of any edge in this graph leads the graph with a Hamiltonian path.

(Refer Slide Time: 42:18)



So, this type the algorithm that we described is wrong this fails. So, it looked like a reasonable thing to try, but it fails.

(Refer Slide Time: 42:35)



So, what you do is this? So, here is my graph. So, here is let me take some edge u v. Now, what I will do is remove the edge u v remove this edge and I will attach to other

vertices this is u prime that is v prime. So, I have a original graph G I remove the edge u v. So that is g1 from here and then I attach two additional vertices u prime and v prime.

Now, I ask does this graph have a HP, does this graph has a Hamiltonian path. The question we wanted to answer was does this have a Hamiltonian circuit. Now let us see, supposing this graph did have a Hamiltonian circuit. Supposing this had a Hamiltonian circuit not only that the Hamiltonian circuit pass through the vertices u and v. In order which means the edge u v was present in some Hamiltonian cycle in the graph.

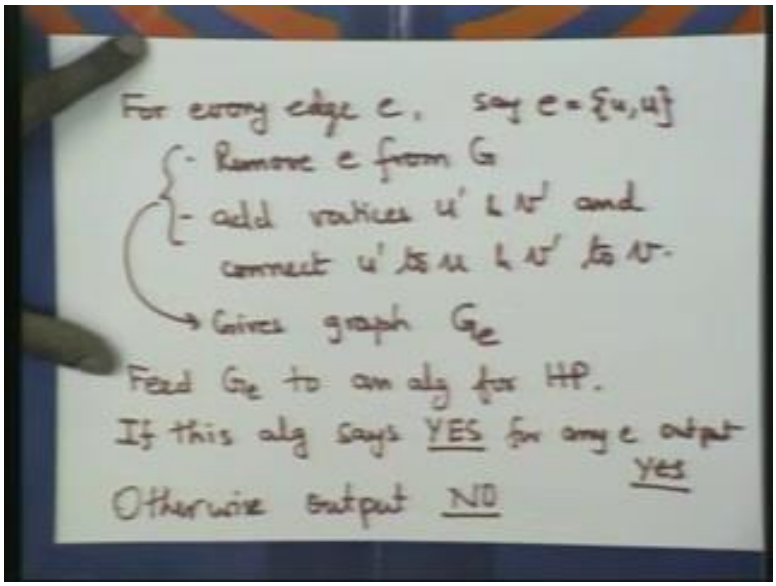
Then, let us notice that this new graph has a Hamiltonian path. So, what does it mean for this graph to have a Hamiltonian cycle passing through u v? It means there must be a cycle this way it goes through all other vertices like this and also u v. So, let us look at the same cycle, where I take the same cycle and I add these two edges and I get a Hamiltonian path in the new graph.

So, if the old graph had a Hamiltonian circuit. Then, if I picked an edge which was present in the Hamiltonian cycle and ask if this has a Hamiltonian path. Then, it will have a Hamiltonian path that is the first thing to notice. Now here is the second thing to notice supposing I took a graph like this. So, I took a graph and here is G. I took a graph G edge u v I removed u v attached u prime and v prime and I ask if this has a Hamiltonian path.

Supposing this says yes what does the Hamiltonian path look like? Now, the path has to these two vertices have degree 1. So, they have to be the end points of the path u prime and v prime have to be the end points of Hamiltonian path which means the Hamiltonian path has to look like this. It has to start at u prime and then go through all vertices in G and then go back up to v prime. This is how the Hamiltonian path should look like.

Then, what we do is this? We remove these two edges. And then put this edge u v which you have removed this will give you a Hamiltonian circuit in the original graph. Now, our algorithm is almost complete, what we do is? We do this for all edges in the graph. We remove that edge attach this u prime v prime and ask if it has a Hamiltonian path. If any edge it says yes then we say yes it has a Hamiltonian cycle. If for all edges it says no then we say no, it does not have a Hamiltonian cycle. So, let me write this algorithm, then we will argue that this algorithm is in fact correct.

(Refer Slide Time: 46:19)



So, the algorithm is this for every edge e, do the following. So, remove e from G, let us say e is equals is u v. Then, add vertices u prime and v prime and connect u prime to u and v prime to v. This lets say graph G lets say e, for an edge e I get this graph G e. Now feed G e; that means, input G e to an algorithm for HP. If this algorithm says yes for any e output yes, if it says no for all e you output no. So, otherwise output no.

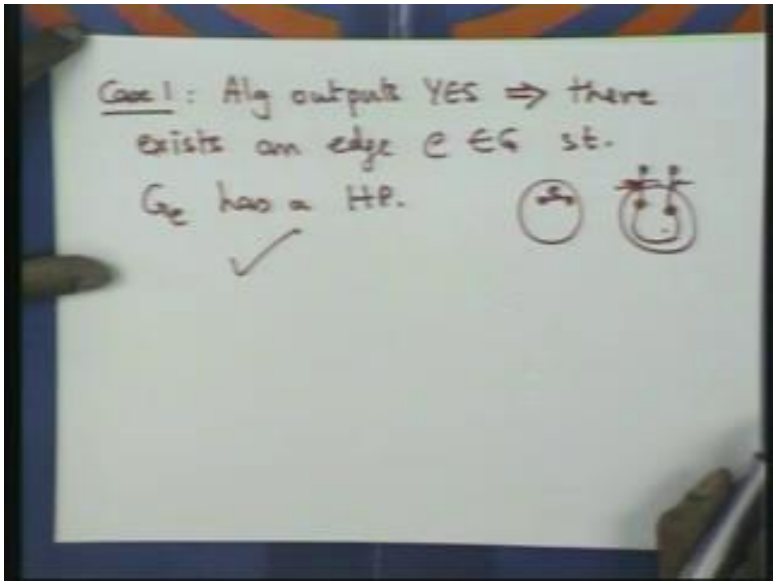
So you output no, if it says no for all edges you do this you remove this edge add these two extra vertices and ask whether it has a Hamiltonian path. If it says no for all these edges then you output no. So, this is the algorithm and we have to show that this is correct which means if the original graph had a Hamiltonian circuit. Then, this algorithm will always say yes.

If the original graph did not have an Hamiltonian circuit, it will say no and it is efficient. If the Hamiltonian path algorithm runs time polynomial in the input size, so does the new algorithm. So, these are the three things to check, let us just make sure that the algorithm is efficient first. You call it once per edge the Hamiltonian path routine is called once per edge and you do not change the input size by much.

You just remove one edge and you add two more edges. So, the number of edges goes up by one. So, the input size does not go up by much and you are calling it at most m times. So, the original running time was bounded by some n to the constant. It is still bounded by some n to the constant again there are two cases. Case 1 is when the algorithm outputs

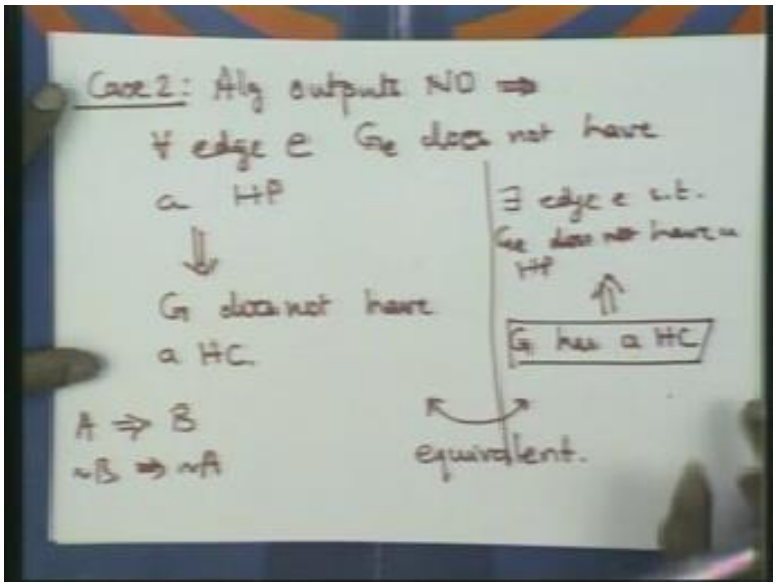
yes then it has to be correct. Second case is the algorithm outputs no this no answer also has to be correct.

(Refer Slide Time: 49:54)



Let us take both these cases. So, case 1, so the algorithm outputs yes. So, this implies; that means, for some edge HP must output yes. This means there is there exists an edge e in G such that G e has Hamiltonian path. So, here was G and here is e I get G e by removing and adding these two. So, this had a Hamiltonian path. And now we have seen this argument that if this has a Hamiltonian path then this has a Hamiltonian cycle. Essentially you take the Hamiltonian path here remove these two edges and add this edge back together Hamiltonian cycle here. So this case is done.

(Refer Slide Time: 50:59)

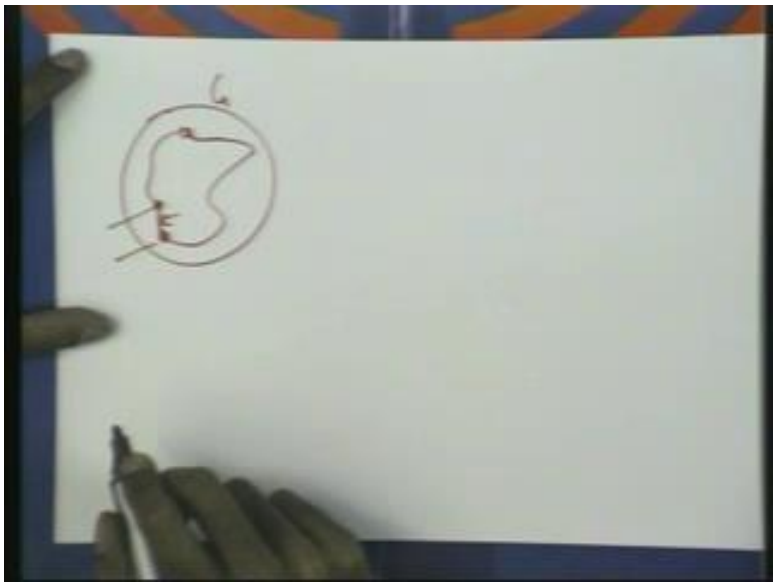


The algorithm outputs No this means for every edge e G e does not have Hamiltonian path. So, we need to see that this implies that G does not have Hamiltonian circuit. So, this is what we need to show? So, we need to show that for every edge e if G does not have Hamiltonian path, then G does not have a Hamiltonian circuit. Now, it is easier to prove the contra positive, which means this statement, which I have written is equivalent to saying the following that this let me write the equivalent statement on the right hand side.

So, I am going to say that G has a Hamiltonian cycle this implies if G has a Hamiltonian cycle what should happen? This means this statement cannot happen. It is not the case that for every edge e G does not have a Hamiltonian cycle. It means there exist an edge e such that G e does not have HP. So, these two things are the same. So, these two are equivalent. So, let me just sort of say what I am doing. So, you want to prove that A implies B you are proving that NOT B implies NOT A. I want to prove that this implies this is A and this is B.

So, I want to prove that A implies B all I am doing is NOT B is this implies NOT A which is this because I am going to prove that if G has a Hamiltonian cycle. There exists a edge such that G does not have a Hamiltonian path. And you can see this also we have proved.

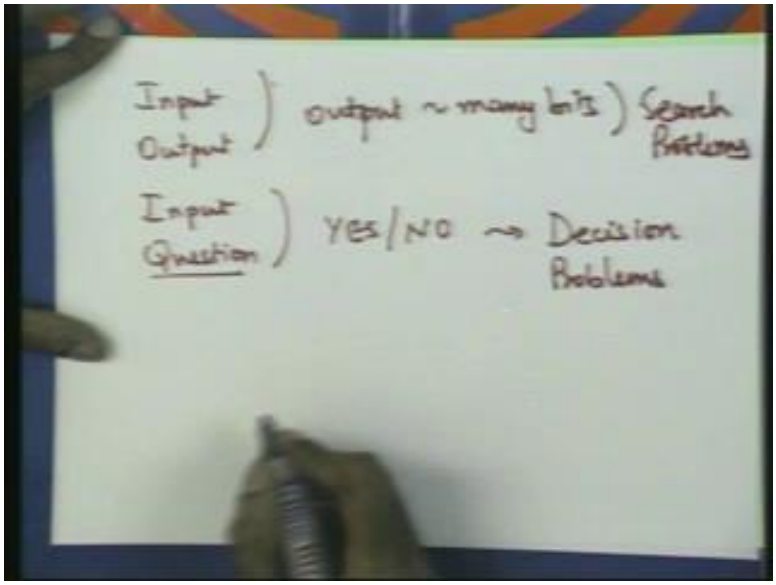
(Refer Slide Time: 53:37)



If G has a Hamiltonian cycle, so that is G. So that is my Hamiltonian cycle, if I remove any edge in the Hamiltonian cycle. So, if I remove this edge and attach these two vertices this resultant graph has a Hamiltonian path which is what we wanted to prove. So, we want to prove that G has a Hamiltonian cycle.

There exists an edge such that G does not have sorry G does have a Hamiltonian path. I am sorry about this G does have a Hamiltonian path and this we have just proved. In fact, this edge e can be any edge in the Hamiltonian cycle. So, remove any edge in the Hamiltonian cycle for that edge removal of that edge G e will have to remove that edge attach those two vertices. This new graph G e will have a Hamiltonian path.

(Refer Slide Time: 54:37)



So, when we looked at these problems there were two kinds of problems we looked at. For some problems we said that we wrote input output. For certain other problems we said input question, now when I looked at problems like with finding a matching of maximum weight of finding a matching with maximum number of edges. They were of the form input and output. Because, the output was edges which were there in a matching of maximum size.

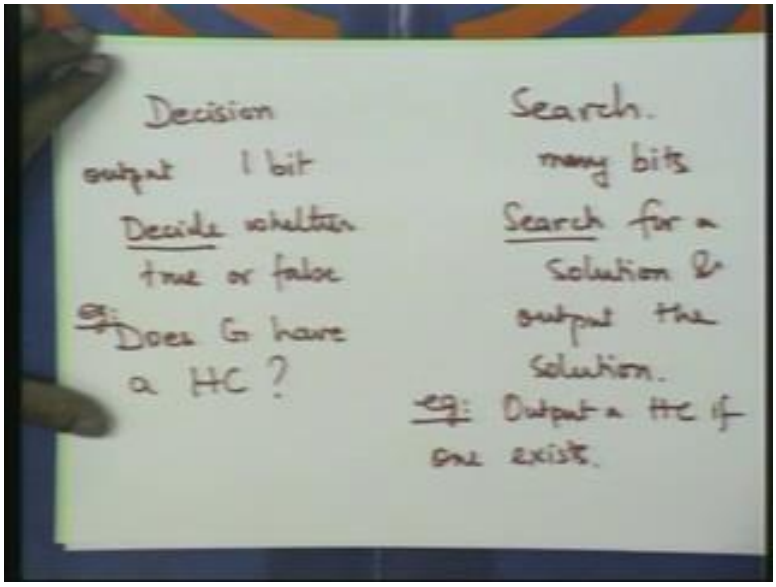
Now, for instance does the graph has a Hamiltonian cycle that is the question the input is a graph. Does the graph has a Hamiltonian cycle I ask a question the answer should be yes or no. Now, the difference was for these kinds of problems. The answer was a single bit, it was yes or no. For these problems the output spend many bits. So, these had a single bit as the output these had many bits as output for instance edges in a matching.

In fact, let us look at the Hamiltonian cycle problem. We said input is a graph G, does G have a Hamiltonian cycle if yes or no. So, the answer is yes or no. On the other hand I could have asked for a output for saying output the Hamiltonian cycle input is a graph G

output the Hamiltonian cycle. Now, this problem seems to be somewhat similar to the previous problem only here the output has many bits. We have to output every edge in a Hamiltonian cycle.

We have to choose one Hamiltonian cycle and output the edges from the Hamiltonian cycle. So, we would like to distinguish between problems, where we require the output to be 1 bit and problems, where we require the output to be many bits. These we call decision problems and these we call search problems.

(Refer Slide Time: 56:58)



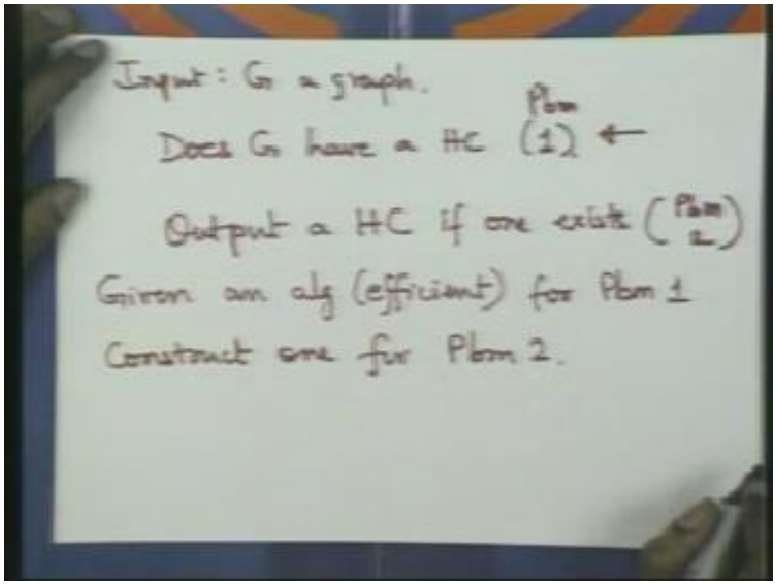
So, typically here for decision problem let me write decision and search. So here, output is 1 bit here is many bits. So, here you want to decide whether true or false here you want to search for a solution and output the solution. That is why this I guess that is why these are called decision and these are called search problems.

So example, does G have a Hamiltonian cycle. So, this is an example of a decision problem. Example of a search problem output a Hamiltonian cycle if one exists. The input is same to be both which is a graph G in one case you just want to know G has a Hamiltonian cycle or not. The other case you want Hamiltonian cycle output you want actually the edges we have.

Now, how do these two things, how do these two problems relate to it. Is one easier than the other is one harder than the other? Now it turns out people have just observed it that

they are related to each other quite closely in the sense that one is easy the other one also turns out to be hard.

(Refer Slide Time: 59:08)



So, let us take this Hamiltonian cycle problem and see this. So, I have so the input is a graph G. Now there are two problems, one is does G have Hamiltonian cycle, this is 1. And the second thing output a Hamiltonian cycle if one exists this is problem 2 that is problem 1. Now supposing there is a algorithm for problem 2. So, which means given so if you feed in a input graph the sub routine outputs the Hamiltonian cycle.

Can we find a algorithm for problem 1, yes I hope all of you answered yes. So, the answer has to be yes. So, you just feed the graph in so this sub routine and look at the edges which have been output. So, if it forms a Hamiltonian cycle, then you just output saying that yes it does in fact have a Hamiltonian cycle. And if it says no it does not the original routine says no it does not have Hamiltonian cycle, I just then you just output things, so that is trivial.

Our job is to look at the other way which means I have an algorithm for problem 1. I have an algorithm for this. Now I want a algorithm for this so given. Let me add efficient just to remind you that we are talking of efficient algorithms given an efficient algorithm for problem 1. Construct one for problem 2. So, this is what we really want to do and how do we do this.

We have actually done something like this before which was with respect to matching's when we wanted to find the edges in the matching of maximum weight. We used this sub routine which answered something else and we somehow got these edges out and we used absolutely the same trick here. So, what we do is this? We will look at the edges one by one let us say the edges e 1 e 2 e 3 e 4 and so on.

So, we look at the edges one by one I remove e 1. Now I ask if the graph has a Hamiltonian cycle or not. If the graph has a Hamiltonian cycle it does have a Hamiltonian cycle which does not use this edge e 1, so I just throw away e 1. Similarly, I look at e 2 I remove e 2 and ask if the graph has a Hamiltonian cycle. If it says if the sub routine says yes I just throw away this edge and I keep doing this.

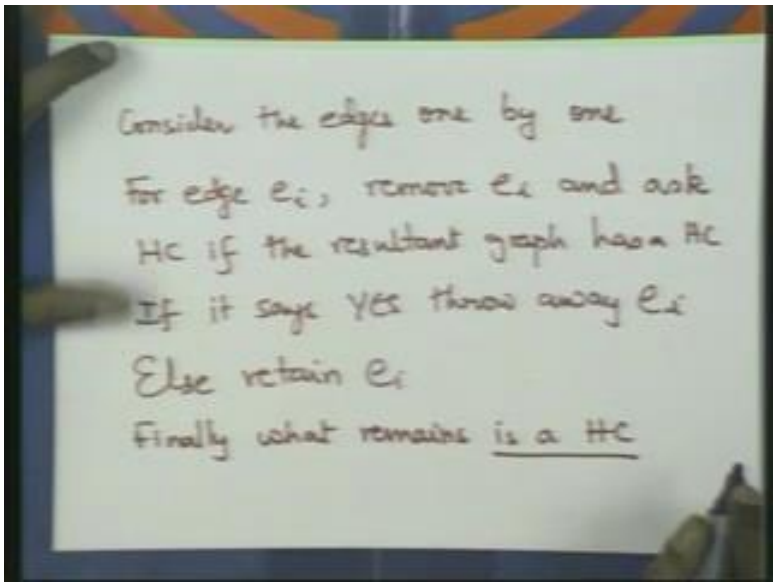
Supposing I remove an edge and it says the graph does not have a Hamiltonian cycle then you also know that this Hamiltonian cycle uses this edge. So, these edges you put back in and you do not want to throw these edges. So, you just do this look at all edges one by one for those edges, where the algorithm says that it does not have a Hamiltonian cycle.

You throw away those edges, but edges for which the algorithm says that there is if you remove there is a Hamiltonian cycle. You throw away those edges for edges when you remove those edges and ask the question it says no there is no Hamiltonian cycle; that means, this edge must be present you put it back here.

So, you do this for all edges and at the end of it you throw away some of edges and some edges remaining. The claim is that the edges which remain must form a Hamiltonian cycle. Firstly, the graph must have a Hamiltonian cycle, because whatever remains whenever we ask whether it has a Hamiltonian cycle it says, yes.

So, the graph that remains must have a Hamiltonian cycle, why should not there be other spurious edges floating around, why should it not have apart from the Hamiltonian cycle other edge. Supposing it had some other edge when you remove this edge and ask whether the graph is Hamiltonian cycle. The answer should have been yes. So, the answer is yes you would have thrown this edge away. So, there are no spurious edges. The only edges which are present are Hamiltonian.

(Refer Slide Time: 1:04:29)

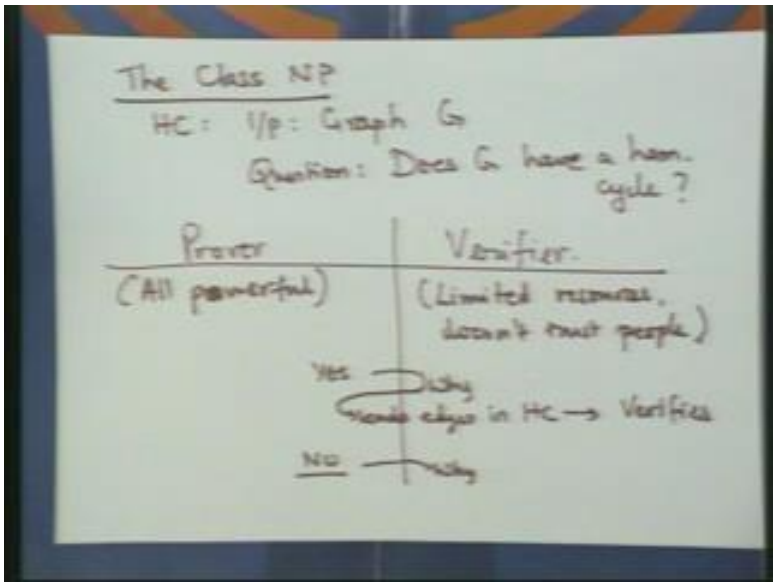


So, let me write this. So, just to recap we are given a algorithm that says whether a graph is a Hamiltonian cycle or not. We are constructing using this as a sub routine, we are trying to find out edges in some Hamiltonian cycle. So, consider edges one by one for edge e i remove e i and ask HC if the resultant graph has a Hamiltonian cycle. If I says yes throw away e i else retain e i that is it you just do it for all edges.

Finally, what remains is a Hamiltonian cycle which means what remains are edges from a Hamiltonian cycle. So, you can see that both problems are quite related I just need to call the other algorithm n times able to actually extract the Hamiltonian cycle. So, it is been observed for most problems there are of course, problems which do not fall into this which do not fall into this property. But for most problems if I can sort of decide then I can also search for a solution. So, if the decision version of a problem is easy then even the search version is easy.

On the other hand, if the search version is hard then social decision word, so this is the reason they are going to focus on decision versions of problem from known. So, we are just going to look at the problems which have whose output is 1 bit and say yes or no.

(Refer Slide Time: 1:06:53)



We are now ready to define the class NP. So, before I formally define this. Let me, sort of give you a informal definition. Let us take our favorite Hamiltonian problem. So, HC the input is a graph G and the question does G have Hamiltonian cycle. So, let us look at this one just consider now we are going to do something slightly different from what we have been doing. We are not going to design algorithms for the time being what we are going to do is have a game between two players.

There are two players now one I will call a Prover and one I will call a Verifier. These are two people Prover and verifier, Prover is all powerful; he just knows everything. The Verifier has limited resources and it does not trust anybody sort of does not trust. Especially he does not trust the Prover he has limited resource.

Now, the Prover and Verifier sort of lets sitting in a room. And there is a huge graph in front of just imagine that somebody has drawn a huge graph in front of them. The Prover looks at the graph and he says this graph has a Hamiltonian cycle he is powerful he can figure these things out and he says this graph has a Hamiltonian cycle.

Now, the verifier is skeptical he does not believe the Prover. So, he says the why should the graph have Hamiltonian cycle and what is the deal, then what should the Prover do. The Prover wants to convince the Verifier that he is all powerful and he has made the right statement he said the graph is Hamiltonian cycle and the graph has a Hamiltonian cycle. So, he wants to convince the verifier that the graph has Hamiltonian cycle.

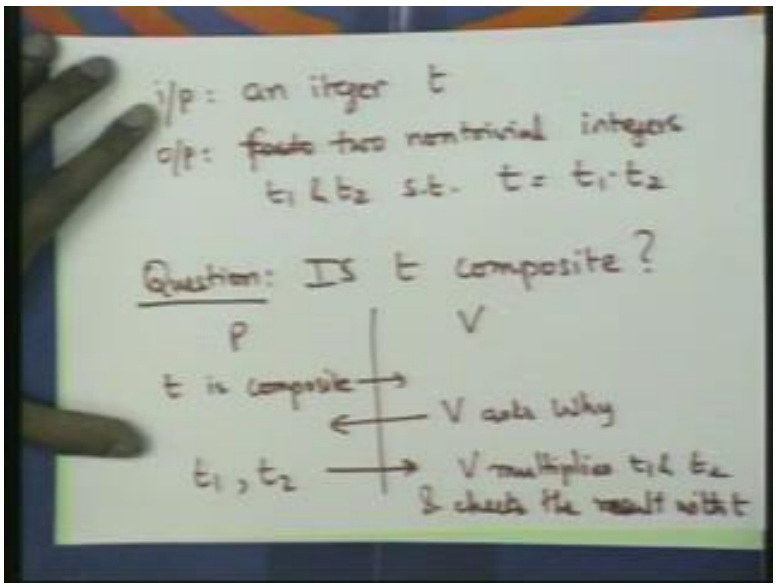
So, what could he do? Well if you think about this for a minute what the Prover could do was pick out the edges from the graph which form some edges which form some Hamiltonian cycle. So, pick out some Hamiltonian cycle from the graph and tell the Verifier that here are a set of edges which form a Hamiltonian cycle.

Now all the verifier needs to do is look at these edges make sure they form a make sure that they form a cycle. You just have to verify that these set of edges form a Hamiltonian cycle which is easy to verify even though he has limited resources and he is not too intelligent this much he can do. So, the Prover has somehow convinced the verifier that this graph has a Hamiltonian cycle.

So, the Prover says that there is a Hamiltonian cycle in this graph the Verifier asks why the Prover then picks out the edges. These are the edges look at these edges they form a Hamiltonian cycle in a graph. The Verifier then verifies that yes indeed the graph did have a Hamiltonian cycle. And we have just proved that Hamiltonian cycle is in the graph is in the class NP. I have not defined the class NP.

So, you really do not know what it is. But, this is all there is to proving things proving the problems are in the class NP. So, let us do this Prover Verifier game for some other problem just to get sort of just to get used to it. Let us say the input is an integer t. So, you want so the output factors let us say two non trivial integers t 1 and t 2 such that t equals t 1 times t 2. So, you want to find factors of these.

(Refer Slide Time: 1:12:08)



Now, this is a search question it is not a decision question. So, the corresponding decision question is the following. So, here is the question is t composite which means can t be written as t 1 times t 2 where t 1 is not 1 and t 2 is not 1. So, these two are non trivial factors of t. So, let us play the same Prover verifier game. So, here is the Prover whom I will say P here is the Verifier V.

So, P says t is composite, now V asks why, now t has to convince V that in fact, it is composite. So, Prover sends t 1 and t 2 across. So once, he sends t 1 and t 2 across Vmultiplies t 1 and t 2 and checks the result with t. If its equal to t of course, he is convinced that t is composite. And in fact, t 1 and t 2 are two factors go back and look at this Hamiltonian problem again.

So, let us look at this again, now if the answer was yes it does have a Hamiltonian cycle which means the Prover says it has a Hamiltonian cycle then the Prover could convince the Verifier that the graph does have a Hamiltonian cycle, what if the Prover says no it does not have a Hamiltonian cycle. So, usually he says yes this guy asks why, so he sends edges in HC and this guy Verifies.

Now supposing, he says no, it does not have a Hamiltonian cycle. Now the verifier asks why and well what does the Prover do could he convince, how could he convince the verifier the given graph does not have a Hamiltonian cycle. Now, think about this for a little, while you see that it is a difficult question to answer, how could he convince somebody that this graph does not have a Hamiltonian cycle.

Well the only way seems to be you just try all possible subsets of edges whether they form a Hamiltonian cycle or not. If known of them form then it does not have, but this is a brute force method. And the verifier does not have so much time he does not have. So, much time to check all possible subsets of edges. So, there is no easy way for the Prover to sort of convince the Verifier that G does not have a Hamiltonian cycle. So, it looks like the answer to the question, whether it is yes or no. If it is yes the Prover can convince the Verifier if it is not the Prover is not able to convince the Verifier. So, they behave sort of differently.