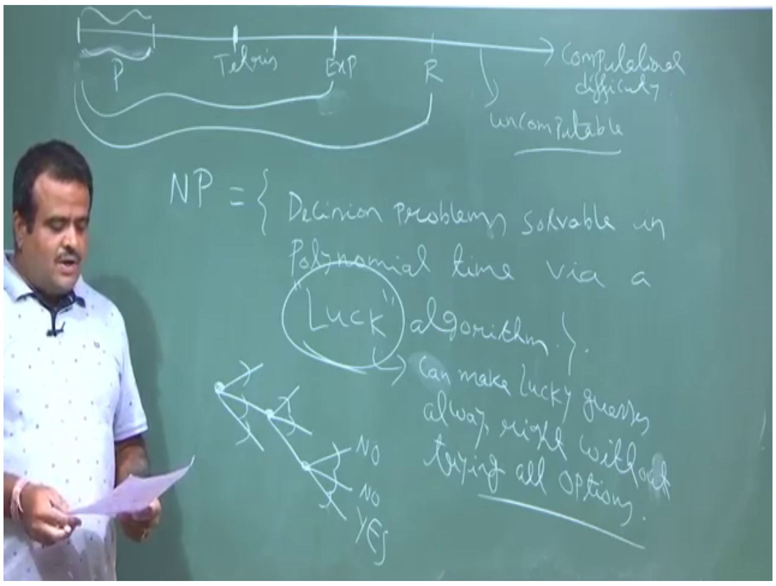
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 60
Computational Complexity (Contd.)

So we are talking about computational complexity. So, in the last class, we have discussed 3 type; 3 class of; 3 complexity class P, EXP and R.

(Refer Slide Time: 00:28)


P
Tetris
EXP
R
computational difficulty
uncomputable
NP = { Decision problems solvable in polynomial time via a "Luck" algorithm }.
can make lucky guesses always right without trying all options.
No
No
Yes

So, P is the set of all problems which can be solvable by polynomial time and EXP is the set of all problem which can be solvable by exponential type. So, P is a subset of EXP and R is the set of all problem which can be solved in finite time I mean and after are all the problems is basically uncomfortable problem. So, the here all the problems are uncomfortable.

So, this is basically we have seen most of the decision problems are basically uncomfortable problem decision problem means whether answer is yes or no I mean whether we can given a chess board given a configure of the chess board whether white wean or not. So, this is a decision problem given a graph directed graph whether there is a negative cycle. So, this is it; this is a decent problem answer is yes or no, but this is in p that detects design detection of in negative cycle.

Now, there are problem like Tetris; Tetris problem is basically this is also a decision problem. So, given a boat and given a sequence of the block sequence now the question is whether we can survive or not; so, these sequence are falling sequentially. So, whether we can survive or not either we can move this or this. So, now; so, these are the complexity class we have discussed now we talk about another class which is called N P. It is not N P is not non polynomial, it is non deterministic polynomial; NP. So, what is NP? This is basically set of all decision problem, decision problem which are solvable in polynomial time (Refer Time: 02:44) in polynomial time via a lucky algorithm via a lucky algorithm. So, what do you mean by lucky algorithm?

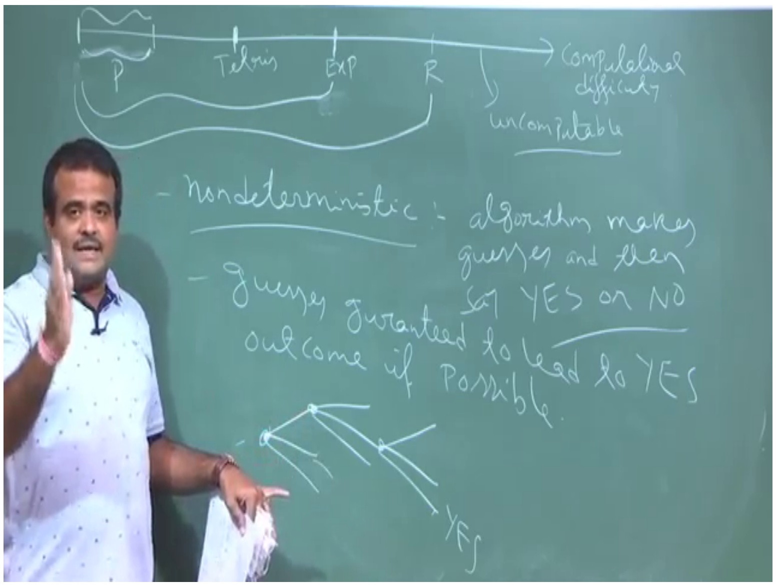
So, lucky algorithm means. So, we can I mean see there are many possibilities from a starting point. So, say this is a decision problem. So, we have there are many possibilities over here. So, from here we can go like this. So, in the dynamic programming problem we explore all the possible all the way now. So, from here say we are going like here, here, here, these are the possible move now suppose this is a decision problem in some. So, this is say no this is no and if we have a yes in summary if we have a no everywhere then whatever move will take whatever path we choose will the decision will be no, but if there is a yes, then, every time we choose a path and that is magic, I mean we choose a path and that is called lucky path that will leads us to yes. So, we will we will not explore all the paths like in dynamic programming we will choose one path at each time and that will lead to that yes and that is the lucky choice and by magic we will always get the lucky choice I mean we always get the lucky part. So, this is a fair in magic you can say. So, so that is the lucky algorithm.

So, lucky algorithm means lucky algorithm means we can we can make we can make lucky guesses. So, every time we guess this, then we guess this then we guess this and this is not random I mean we are guessing and that is happened to be a lucky guess. So, that is basically a magic. So, can make lucky guesses always right without trying all option like in the dynamic polling problem we are trying all option, but here we are just choosing all option and that is the good guess by guessing that my magic sale with without trying without trying all options. So, this is what is referred as lucky algorithm. So, we are not going for all path we are picking one path and that is happen to be a lucky choice; that means, that will leads us to the answer yes so; that means, it is the lucky choice of that.

So, this is called a non deterministic model. So, this is that is why it is called NP; so, non deterministic polynomial. So, this is a polynomial time algorithm, but polynomial time, but non deterministic way it is not deterministic I mean by magic I mean we choose a lucky algorithm. So, lucky algorithm means we make the lucky guesses always without trying for all options. So, that is called non deterministic model.

So, let us write that. So, N P is non deterministic polynomial time.

(Refer Slide Time: 07:01).

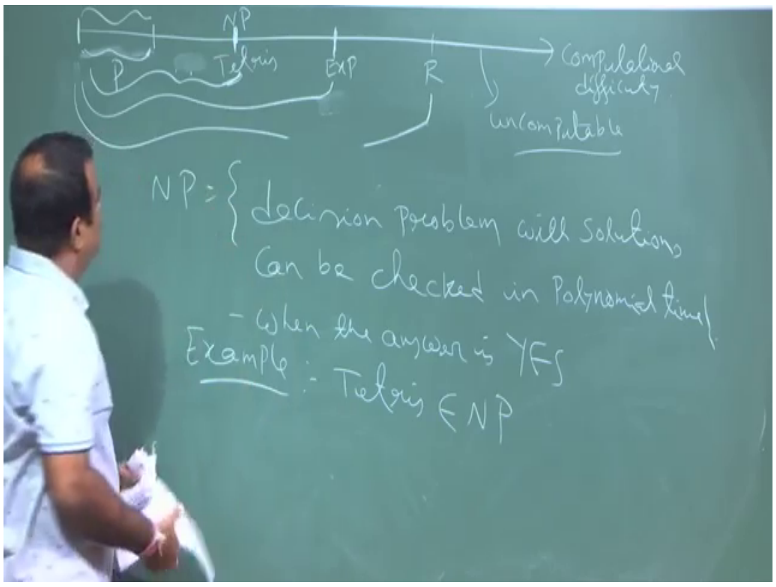

P
Tetris
EXP
R
Computational difficulty
uncomputable
- nondeterministic :- algorithm makes guesses and then say YES or NO
- guesses guaranteed to lead to YES outcome if possible.
YES

So, what is non deterministic non deterministic model is algorithm makes guesses and then say yes or no. So, so this model we cannot be like a real computer, but theoretically you can study this model there is no real computer which for which we can build this model because we cannot I mean what is the guarantee that we will be lucky always. So, this model is not realistic model in the sense we cannot build this in a real computer, but theoretically it is we can have a theoretical study on this.

So, this means. So, this means guesses are guaranteed it is not say random guess guesses guaranteed guesses guaranteed to lead to yes outcome if there is yes lead to yes outcome if possible so; that means, then suppose we start from here there are few possibilities then suppose we start here also few possibilities suppose from here also few possibilities suppose this leads to a yes I this is yes then. So, every step every path every step, we choose one path not all possibilities. So, here also we have some moves.

So, this is our guess and this guess is guaranteed in the sense we will always choose this path which will lead to us yes if there is no yes then there is no I mean then the result will be no, but if there is a yes if possible then our guess is always correct. So, this is sort of magic. So, no realistic computer can guarantee that, but theoretically this is ok. So, this is basically the N P class. Now this is another definition of N P classes another way to view the N P classes.

(Refer Slide Time: 09:51)



NP = { decision problem with solutions can be checked in polynomial time }
- when the answer is YES

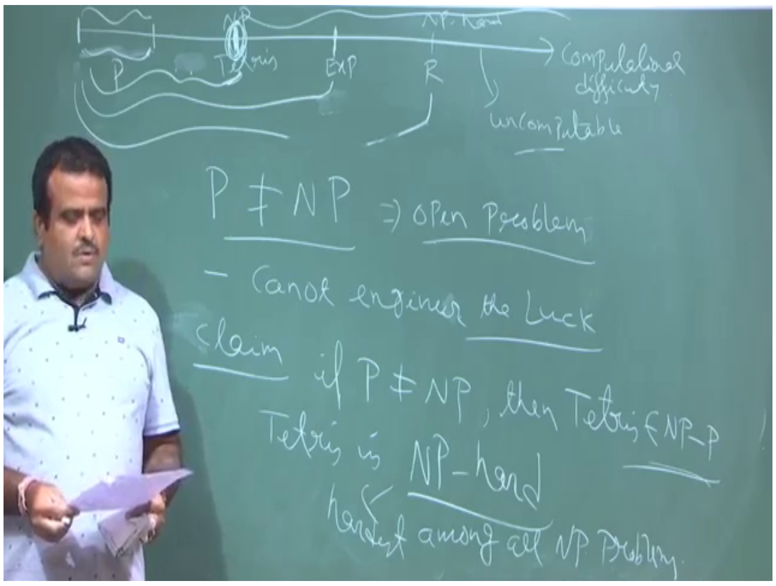
Example: Tetris $\in$ NP

So, it is a set of all decision problem with solution decision problem with solution with solution that can check in polynomial time which solution can be a checked in polynomial time polynomial time. So, when the answer is yes answer is yes we can prove it can prove it and we can check the proof in polynomial time. So, this is the another way to view the N P class. So, N P is non deterministic polynomial time. So, one example of N P is Tetris; Tetris belongs to N P. So, so we can have a non deterministic algorithm so; that means, we can always guess and we can reach to a answer that whether we survive or not.

So,. So, this is basically. So, this is basically our N P class. So, we can just write this up to this is our N P and Tetris is a is belongs to because we can very well have a non deterministic algorithm non deterministic algorithm to I mean always we will choose a by magic whether this can we can fit this way these way other ways we can choose a option I mean always we can have a gate guess algorithm which can lead to us whether we can survive or not. So, that way we can. So, Tetris is in N P.

So, now we want to tell something about this N P and p. So, this is basically up to this it is NP. So, Tetris is belongs to NP. So, now, we want to see.

(Refer Slide Time: 12:58)



So, now there are some conjecture like p is not equal to N P. So, N P is a harder problem right, N P is polynomial time it is a problems which can be solved by polynomial time, but with the help of a lucky algorithm. So, non deterministic algorithm; so, that way it is; it must be a this is; this is sort of this is a conjecture this is a open problem this is the open problem million dollar problem if you want to be famous you can solve this problem this is a still a open problem I mean our intuition is this should not be equal set. So, people have tried whether equal whether people have tried whether they are not equal, but there is no such proof so far P not equal to NP, but this is a conjecture conjectured that we have this.

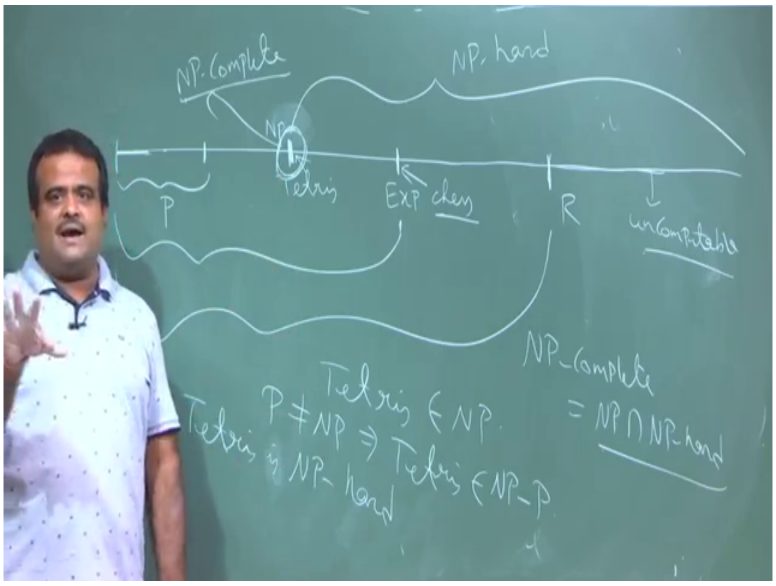
This matrix must be the harder; harder set this matrix should be the bigger set of problems than p another way is to said is P not equal to m plea if we believe this that another way to say this we cannot engineer the guess so; that means, we can believe on case, but we cannot guarantee the guess we cannot develop the case I mean. So, this is cannot engineer the guess sorry cannot engineer the luck we can believe on luck, but we cannot guarantee on luck. So, that is the thing we cannot engineer on luck ok.

So, now our claim is if p not equal to N P then the Tetris belongs to N P minus p. So, if p is not equal to N P then the Tetris belongs to. So, this is the Tetris is the hardest problem our hardest problem in N P and. So, this is called this is why it is called traitorous is N P hard as hard as every N P problem. So, it is the hardest problem among all N P problems. So, this Tetris is N P hard. So, it is the. So, N P hard means it is the hardest problem among all N P problem hardest among all N P problem.

So, this is called N P hard so; that means, from here all the problems are N P hard and Tetris is the is here. So, Tetris belongs to this and this is called N P complete problem. So, N P complete means which is basically intersection of N P hard and np. So, which is the which are the most hardest problem in NP. So, these are called the N P complete, but N P hard is the all the problems which are which are basically hard harder than given any N P problem which are basically harder than all the N P problems are called N P hard.

So, let us draw this picture here. So, most time here yeah it can good. So, let us draw this picture here.

(Refer Slide Time: 17:43)



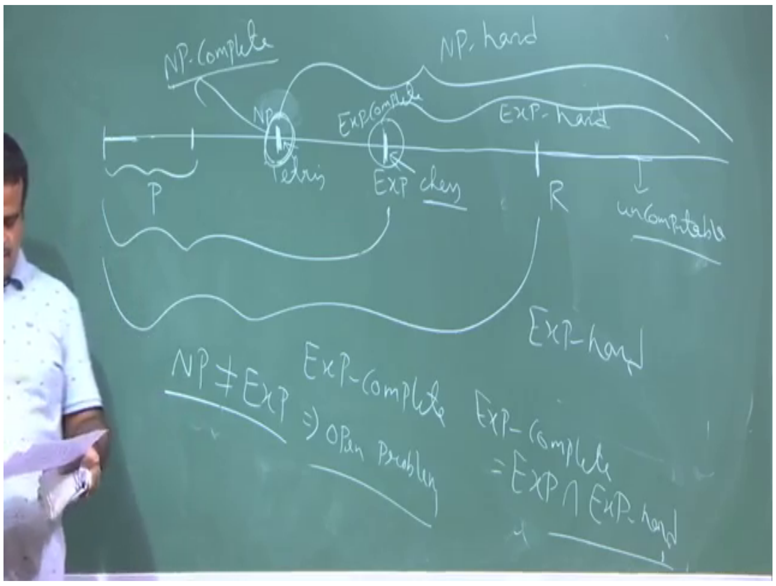
So, this is the line of difficulty computational difficulty the line start from here. So, here we have all pop easy problems and here we have Tetris and here we have exponential problem and this is basically one problem is chess; chess board n by n chess board and here we have r which is basically set of all problem which can be solvable in finite number of times and after all these problems are uncomfortable like most of the decision problems are uncomfortable we have proved that in the last class.

So, what we have seen we have seen Tetris. So, this is basically N P this is N P and we have seen Tetris in N P and we assume if we use the conjecture p not equal to N P if P not equal to N P, then Tetris must be the one of them which is in N P not in p because we are not having any polynomial time algorithm for this decision problem. So, then if this is true then Tetris must belongs to N P minus p otherwise if we have p equal to N P this is p this is N P if they are equal then Tetris must sitting here it is the hardest problem because we have no polynomial time algorithm to solve this so; that means.

So, this is the hardest problem among the as at as every so. In fact, this is basically Tetris is Tetris is belongs to is N P hard N P hard means set of all problem which are from here to here. So, they are basically N P hard so; that means, they are as harder than any N P pop any N P problem. So, these are all problem which are basically harder than any N P problem every N P problem. So, it is basically harder than as hard as every problem belongs to N P and then we have seen that if this N P complete means N P complete means N P intersection N P hard this is N P complete N P complete. So, this is the collection where these are N P problem, but these are as hard as all given n p. So, these are as much hard as any given NP. So, these are all N P complete problem and Tetris is one of the example of N P complete problem. So, there are many N P complete N P complete problem Tetris is one of this example.

Now, similarly from here to here, these are exponentially hard x hard and we define exponential x complete.

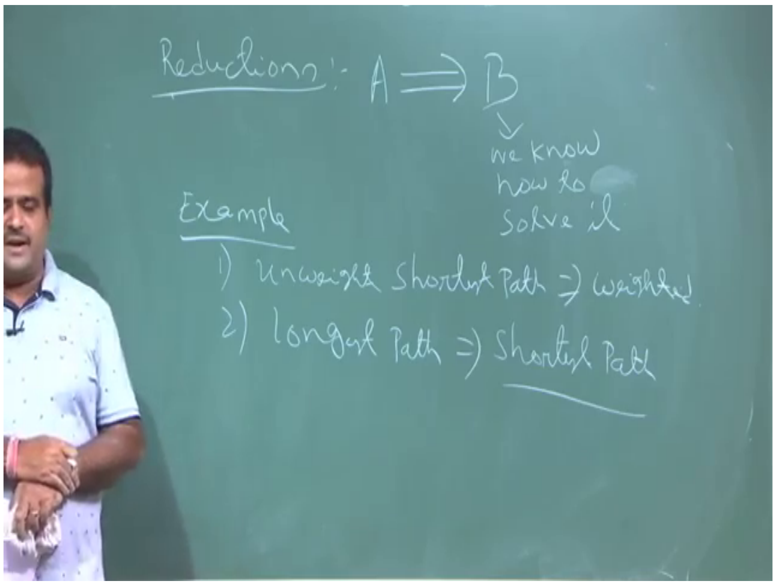
(Refer Slide Time: 22:04).



So, again this is a conjecture N P is not equal to x this is a conjecture this is another conjecture this is also open problem this is also an open problem. So, N P is not equal to ExP if N P is not equal to Exp. So, then we define this is hard and then we define if x complete is basically ExP. So, then ExP hard means x hard means this is the set of problem which is as hard as any given x problem. So, then that is called that is called x hard and this is the set all problems are x hard from this set and if we take the union the intersection x hard then this is called x be complete and chase is one of the example of x complete chase is one of the example of x complete. So, this is the N P complete N P hard.

Now, we will talk about reduction. So, this is the reduction this is the way we design the algorithm sometimes most of the times because the reduction is basically.

(Refer Slide Time: 24:19)


Reductions :- A ⟹ B
we know how to solve it
Example
1) Unweight Shortest Path ⟹ weighted
2) longest Path ⟹ Shortest Path

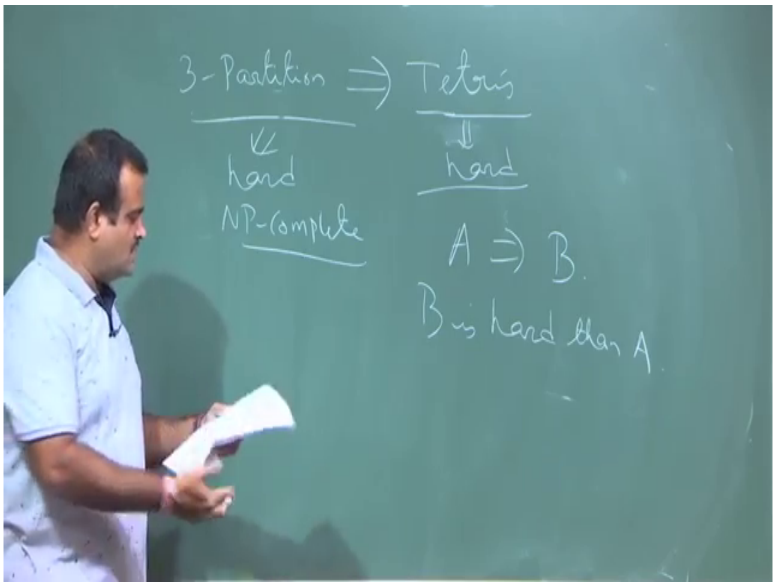
We reduce one problem to another problem mostly we reduce many problems to the graph problems and then we know the solution of this graph problem. So, that solution will give us the solution from this problem. So, this is basically the reduction this is a design technique. So, basically what is this? So, basically we have a problem A; we are reducing this problem to another problem B for which we know the solution; we know the solution, we know how to solve it we know how to solve this how to solve it.

Then in order to solve A, if we can reduce A to B another problem B which we will know the solution which we know the program to solve it, then we know the program we know the solution of A because we have the solution of B. So, like for example. So, this is some example of reduction we have done. So, like we if we have a un-weighted un weighted graph un weighted graph shortest path. So, this is the problem we have a graph on the weighted graph and you have to find the shortest path from one node to another node. So, what we do we know the weighted graph shortest path?

So, what we do we reduce this we convert this problem to the weighted graph problem. So, this is basically we can put the weight of each vertex is weight of each edge is one. So, this is basically we can convert this into weighted graph that is it. So, this is one example; another example could be longest path . So, we have given a directed graph we need to find the longest path. So, we need to the; we know the shortest path. So, what we do we just convert the weight into negative then this will be the basically is the shortest path problem and then we have the solution for the shortest path problem and then. So, this is a design technique basically this reduction and we have used in many places there in order to. So, we basically convert this into a known problem for which we know the solution.

So, now we just talk about how we can use this reduction to prove some something which is called N P completeness like.

(Refer Slide Time: 27:04)

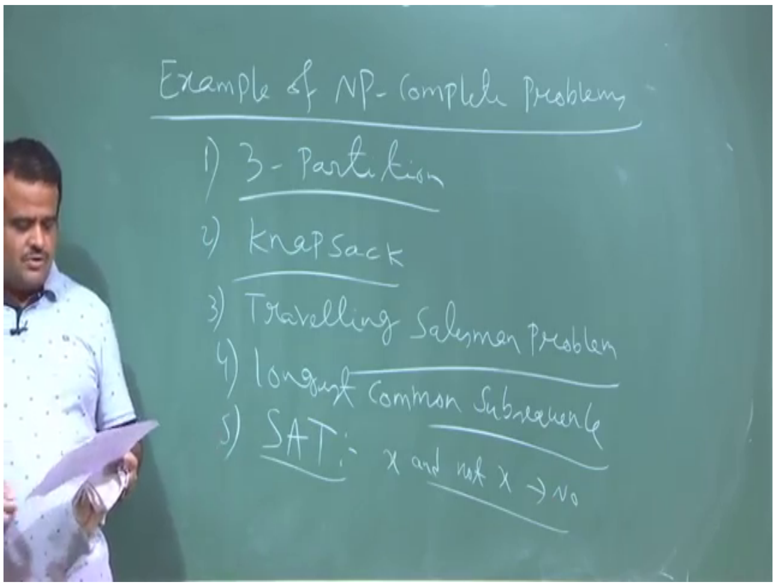


So, say for example, 3 party problem, sorry, 3 partition; 3 partition problem can be reduced to Tetris problem. So, what is the 3 partition 3 partition problem we have given a set we have given the numbers integers n numbers. So, the question is whether we can group this into 3 groups such that some of the each group is same this is the 3 partition problem we can partition this group into 3 groups. So, that sum of each group is same and this we know this is a hard problem this is hard. So, this is N P complete problem.

So, now when we use the reduction is reduced to B; now; that means, B is harder than A because otherwise if you reduce this, if we can solve pay, then we can solve B if this reduction is in polynomial time. So, every N P complete problem there are many N P complete problem. So, we can take any 2 problem and we can reduce one problem to another problem by polynomial time reduction. So, that this is one of the example so; that means, this is we know this is a paper this is we know is N P complete, this is a hard problem. So, this must be harder than this because this we can transfer into polynomial time otherwise if we can solve these easily then we can solve this because we have a polynomial time reduction from this to this.

So, if we can solve this, then we have a instant we have a solution for this because we have given this problem we reduce to this problem. So, if you can solve this problem then we got the solution of this problem, but we know this is a hard problem this is N P complete problem so; that means, this is; so, we can use the reduction to prove the N P hardness. So, this is also. So, since this is N P hard problem this will accompany this is also hard this is also hard.

(Refer Slide Time: 29:41).



Example of NP-Complete Problems

1) 3-Partition

2) knapsack

3) Travelling Salesman Problem

4) longest Common Subsequence

5) SAT:- x and not $x \rightarrow$ No

So, there are some example of N P complete problems and every N P complete problems we can reduce form one to another. So, these are some example of N P complete problems. So, one we have seen is called 3 partition. So, you have given a set whether we can partition into 3 groups such that some of each group is same. So, we have given some numbers sum of each group is same then the knapsack problem. So, knapsack problem is we have given some weight and we have given the target. So, we have to find a subset of this. So, that the addition of this is the target.

So, this is also called substance some problem then the traveling salesman problem travelling salesman problem. So, we have given a graph and we have some edge weight on the graph. So, it a salesman's has to travel; find the shortest path that visit all the vertices from the given graph. So, it is a decision problem again whether we can have.

So, yes or no; so, most of the problem have a decision version; so, this knapsack problem also. So, instead of finding this subset whether with the answer is whether we can have a subset or not. So, that is the decision version of this problem this is also a subset some problem then the longest common subsequence problem subsequence problem we have seen this problem at the starting of the graph algorithm.

So, for 2 subsequence if we have given 2 string; 2 sequence, then this problem we have seen by dynamic programming, but if there are k string if there are k sequence then this problem is hard if k is greater than 2 then we have a problem called sat problem, this is the first problem of N P complete problem this is basically telling us given a Boolean formula is it ever true. So, suppose you have given say x and then not x. So, this is there is some sort of Boolean formula, we have; now the question is; is it ever true. So, this type of problem this is the first problem came into the literature which is proved to be a N P complete problem.

So, all the problems are N P complete further and other many empty N P complete problems, but we can use the reduction to one problem to another problem in polynomial time. So, if we know one problem is hard then another problem has to be hard. So, this is the N P completeness we can prove because if we can solve this problem in polynomial time then this problem is also will be solved in polynomial time because this problem we reduce in polynomial time to this problem. So, this is the sort of computational complexity we discuss.

Thank you.