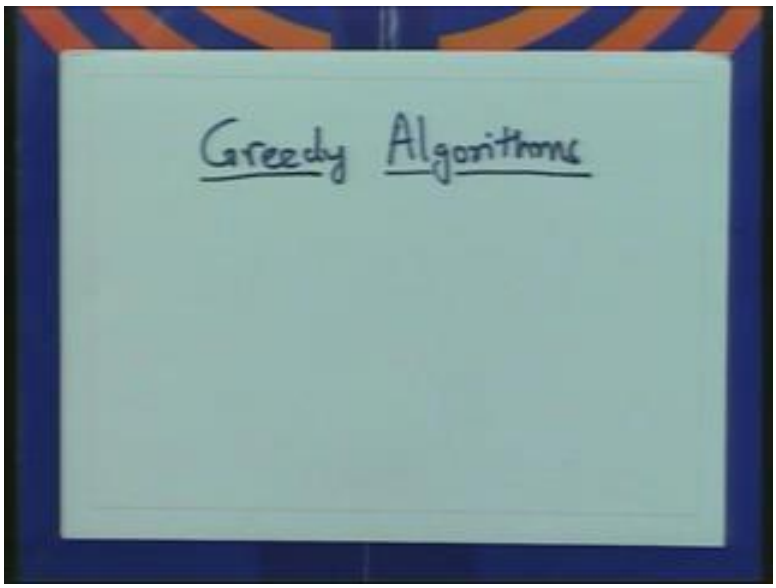
Design and Analysis of Algorithms
Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 10
Greedy Algorithms – I

(Refer Slide Time: 00:59)

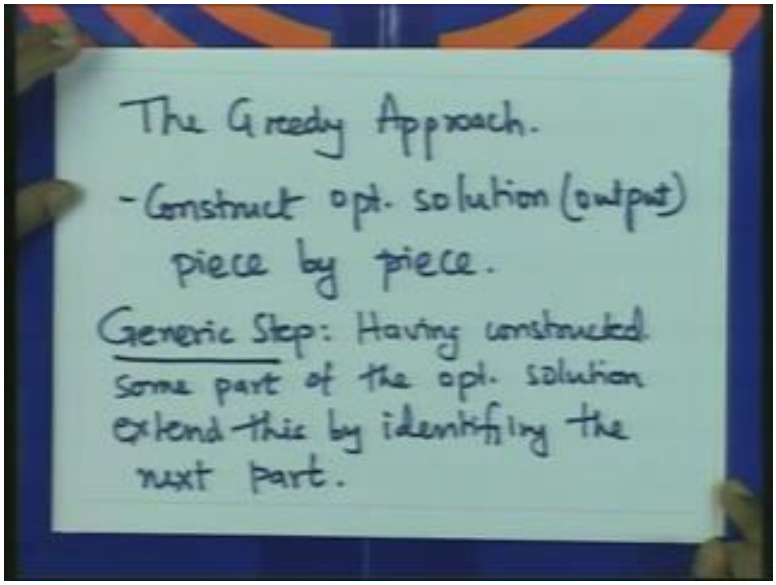


We will look at yet another algorithm design technique. This one is interestingly called greedy algorithm or greedy techniques for designing algorithms. So, the keyword here is greedy and well, it has something to do with greed as we know it is only a little bit. The main technique main sort of technique for these for this algorithm design technique is the somehow piece together your optimum solution small piece by small piece by piece. So, construct the optimum solution somehow by getting a small piece first then sort of enlarging it and so on and so forth till you get the final answer.

So, we will start with an example and we will see how this technique works. In some sense it is a bit difficult to listen you have to some zero in on part of the optimum output. The optimum has some output you have to somehow zero in on some part of it. Then slowly enlarge it and how do you end up with this small piece. How do you enlarge this? That is the crux of design technique here often there are no clear cut guiding rules that I can give you. It is left to your sort of imagination and intuition hard work whatever

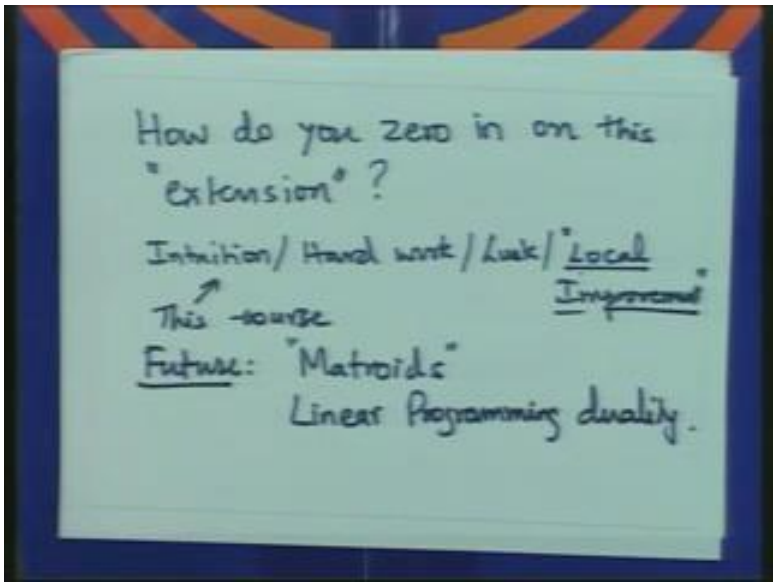
perhaps luck that you come up with the right ideas that build in solutions. So, let me write down the sort of basic ideas here.

(Refer Slide Time: 03:19)



So, the greedy approach, the basic idea the basic step is construct the optimum solution or output piece by piece this is the basic step. So, generic step in your algorithm which will typically be in a loop is having constructed some part of the optimum solution extend this you extend this by identifying the next part. This is the crucial generic step which you will have to design. So, once you get the problem you somehow have to get this generic step. Usually one starts off by figuring out what the first step is. Then the generic step and then you just put the generic step in a loop and that is how the algorithm looks like. Well, the crucial question that you will have to answer for each of these problem is how do you identify you know the next part of the optimum. So, you have got some part of the optimum somehow how do you zero in on the next part? This is the sort of crucial question that you will have to answer.

(Refer Slide Time: 05:41)

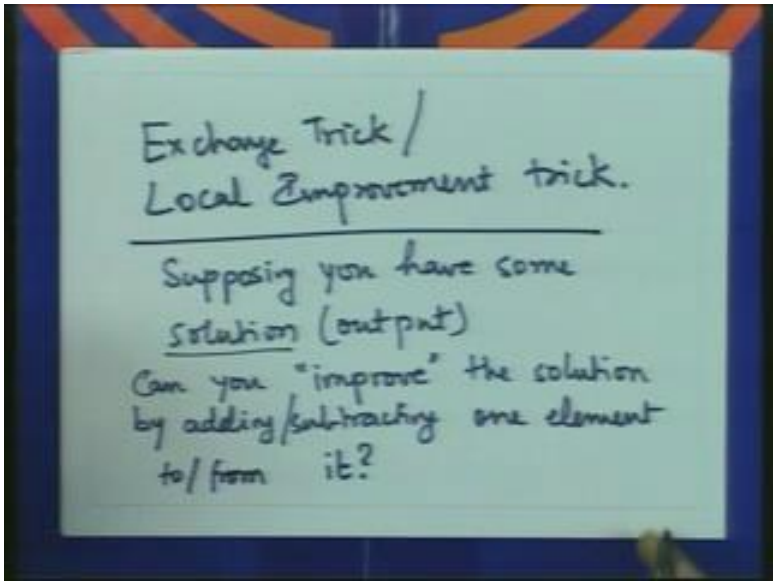


We will have to answer as we go along for this course the question is how you zero in on this extension? By extension I mean extension of this optimum solution like you is building. So, how do you get this? Well, for this course its intuition perhaps hard work well, some amount of luck and I will sort of sort of just say local improvement. Do not worry too much about what this means. I will explain as we go along. This may be the only key I can give you at this point. This local improvement trick is not very formal it is sort of an informal idea that I am going to give you. And we will see that in many of these problems you can sort of use some of these small tricks that I give you to get the algorithm. But often you know you just have to rely on your own engineering as a bigger hint as to where all these algorithms comes from. There is some there are some there is some theory behind it which we will not which as this says beyond the scope of this course, but I will tell you where at least some of it comes from there are 2 subjects.

So, let me sort of tell you what both are and I will strongly encourage you to read about them. The first so, this is for this course so, future I encourage you to read the following. One is Matroids theory this is a theory of Matroids and the second one is linear programming duality. So, that is so, called primary dual method in linear programming duality which gives you a sort of method in using that method many of these so, called greedy algorithms can be devised. In the way you pick up the next part often is dictated by this by this geometric structure. So, but this is for the future forget about this after this course we have the time ((Refer Time: 08:28)). Please do read about these 2 they are

very important algorithm design technique. So, here is the exchange trick which I am going to tell you in a very sort of in a hand wavy fashion.

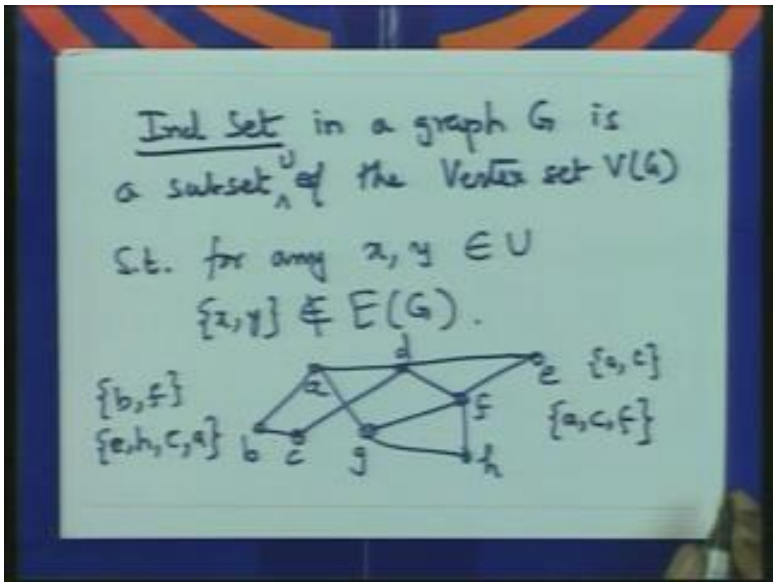
(Refer Slide Time: 08:45)



So, here is the exchange trick or the local improvement trick. So, this trick in this trick what you do is this? So, supposing you have some solution some generic output for the problem. There is an input we want the output. Well, usually we want an output which satisfies which is maximum in some respect or minimum in some respect optimum in some respect. So, supposing you take some solution which may be which may not be optimum.

Well, the question asked is can you improve the solution by adding or subtracting. Let us say 1 element can you improve the solution by adding one element to it or subtracting one element from it? So, this question will sort of dictate how the algorithm proceeds. This question will keep asking and we will see how answers to this question. We will sort have trigger our algorithms; we will sort of push the algorithms forward. So, in some sense we may have partial solution. You see how to improve the solution improves it and so on. So, that is for all the theory behind greedy algorithms. We will now get down into business and solve a few problems. The first problem is the input I need to define a term which you may have seen before.

(Refer Slide Time: 11:20)



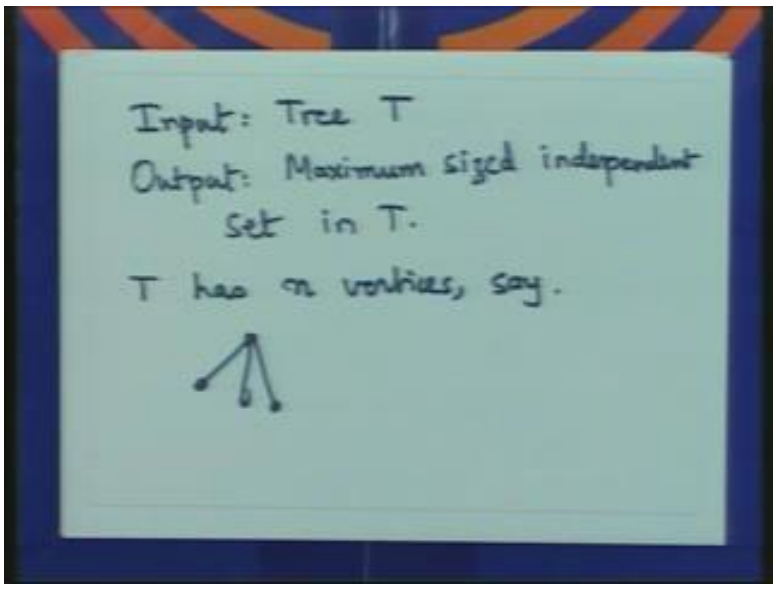
It is a notion of an independent set and independent set in a graph G is Well, it is a subset of the vertex set such that the there are no edges between any pair of vertices. It is a subset let us say U of the vertex at V of G such that if I take any 2 vertices in U there is no edge between them such that for any x y belonging to U, x and y are vertices in U x y does not belong to the edge set of G there is no edge between any pair. So, let us take an example suppose this is a graph let us complicated a graph. So, let us give them some labels a b c d e f g h while you look at this graph let us see 1 independent set could be a. If I put a in the independent set then b and d cannot be part of the independent set, because this is an edge between 2 of them. So, I could have c in it and f. This is an independent set, a c and f is an independent set.

So, I could have other independent. For instance a and c by this is also an independent set a and c this is an independent set. There is no edge between a and c. Similarly, I could have let me put one more let us say b f. So, if I have b and f this is an independent set I could have e h let us see I have e and h then c a this is an independent that is c a. That is another independent set. So, I have sort of ((Refer Time: 14:18)). So, there are many others. For instance each vertex by itself is an independent set. Just take an vertex itself it is an independent set. So, it is just a collection of vertices. So, there is no edge between that is an independent set. Now, for instance if these are radio stations it is a mobile radio stations and they are transmitting they want to transmit and 2 of these cant

transmit if they sort of if they interfere. So, then only an independent set can transmit at a time. If I take if a b c d e f g h are radio stations.

And I suppose I draw an edge showing that these 2 fellows cannot transmit at the same time, because of the same frequency, because of interference then at one particular frequency or at one particular time only an independent set can transmit. So, that is the, that is roughly the motivation where independent sets come in. There are other motivations there are other places where one looks for independent sets. So, general problem is given a graph you want to find an independent set large size. That is the general problem. In fact, an independent set of maximum size that is the problems in that have been traced. This problem tells out to be hard and we look at it later on in this course. For now we will restrict the graphs ((Refer Time: 16:00)). So, here is the problem in totality.

(Refer Slide Time: 16:09)



So, the input the tree T by tree I mean a graph that is connected and acyclic. The output I want a maximum sized independent set in T. I want a subset of the vertex set of maximum size so, that there is no edge between any pair of them. So, this is the problem that we would like to solve. Well, how you solve such a problem? How do you find maximum sized independent set in T? Well, if T has n vertices let us say T has n vertices one way to do it is too we sort of group force way ((Refer Time: 17:24)) sort of compute one to one ((Refer Time: 17:28)) is to look at all possible subset or vertex set. And look

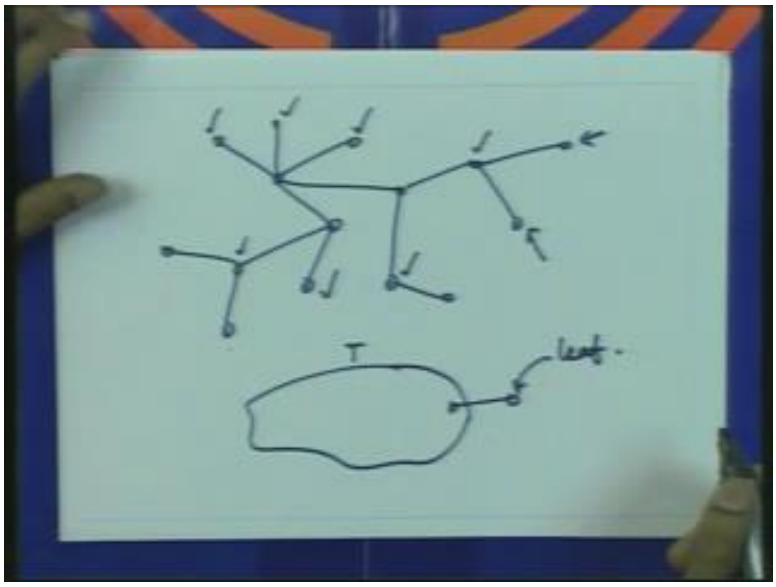
at see they are independent and choose the one which has maximum size. Now, this is just too expensive. The time taken is least number of subsets of n vertices 2 to the n let us do it we are not going to do it much faster what else could one do. Well, pick a vertex if you pick a vertex in the independent set then the neighbors of this vertex will not be in an independent set. So, that is by definition. So, you pick this pick a vertex throughout its neighbors we will look at this graph. Again pick any vertex that remains throughout its neighbors and so on. This for you will get an independent set that is fine, but I do ensure that the size is maximum. If you pick the wrong vertex at any point then you may end up in trouble.

So, here is simple example. So, let us say this is the graph If I pick this vertex if I pick this vertex then what ((Refer Time: 18: 48)) you get is an independent set of size 1 the optimum is of size 3. The solution here the solution to this problem is a subset or a vertex set and you have to somehow pick the right subset. And the greedy solution greedy way says pick it one by one somehow which vertex would you like to put in the independent set ((Refer Time: 19:39)). The independent set is large will you think about it for a while intuition tells you that you should pick a vertex of smallest degree in this graph. So, if a vertex has this small degree then the number of neighbors that you throw up remember when I pick a vertex I have to throw out its neighbors number of neighbors I throw up is small. In particular if I leaf in the independent set. If I pick a leaf in the independent set then the only vertex I just throw out one vertex per leaf right and this actually works. In this case this algorithm works though we need a proof to show that this technique actually it produces maximum size independent set. The algorithm is just this. The just pick a leaf throw out its neighbor and continue. At each stage you will have a forest.

If you have a vertex of degree 0 it is not adjacent to anything else which you may come across as the algorithm proceeds just put it in the independent set blindly. If no such vertex exist then I pick a leaf from there will be many trees as the algorithm proceeds pick any leaf and put it in the independent set and continue. This algorithm actually works. It; however, needs a proof. You need to prove that this simple algorithm that we just described actually works. It is greedy in the sense that we start you piece you put the solution together piece by piece. You pick a piece and slowly sort of proceed in this way. Once you have a part of the solution you do not change it. This is the other sort of property. If I have a few word vertices in the solution I just build the solution up further I

do not sort of throw somehow and then put something and then do back tracking. In this case we sort of; obviously, to extend the solution which is to pick the vertex a minimum degree it works. We will see many examples where simple things are not working. There is yet another way of looking at this algorithm which I would like to describe. This is a bit more general than what we have studied and this is the sort of exchange trick I promised which we will use over and over again.

(Refer Slide Time: 22:28)



So, another way to come up with the same algorithm. So, let us take an example to each. So, let us say this is a tree; tree in question and we would like to pick a maximum independent set in this tree. Supposing you have some subset which is an independent set. Let us say this and so on. This is the algorithm has not picked. This is just to sort of get your thinking process going. Supposing you have you have let us say somebody gives you an independent set this is an independent set somebody gives you this independent set. So, the question is when can I exchange can I sort of add and subtract to this solution to increase the number of vertices in the independent set? Can I add and subtract to this solution to increase the size of the independent set? Well, let us look at this here I can add these 2 and remove this I can add these 2 and remove this. From this example actually its look like I can always do this with leaves the sense that I can add these two and remove this.

So, this is a natural way to lead on to the question will leaves always do all leaves always fall in some optimum independent set. So, this is the question we would like to answer and you can see the answer is yes why is the answer yes? Well, supposing this is the lets say this is the tree this is any tree and I have a leaf here. I have any tree and I have a leaf. Well, what I can say well, one of these 2 better be in the independent set. In the optimum independent set one of these better be in, because if this is not in I can always put the leaf in into an independent set. So, one of them always has to be in the maximum independent set. Well, supposing I have this in the independent set and this is not ((Refer Time: 25:07)) and then you notice that I can remove this from the independent set and put the leaf in. The size of the independent set does not change. The set still remains independent and the size of the independent set does not change. I have picked one element out and I have put a element in and the size does not change and this is the exchange which we did earlier on when we tried to increase the size of the independent set.

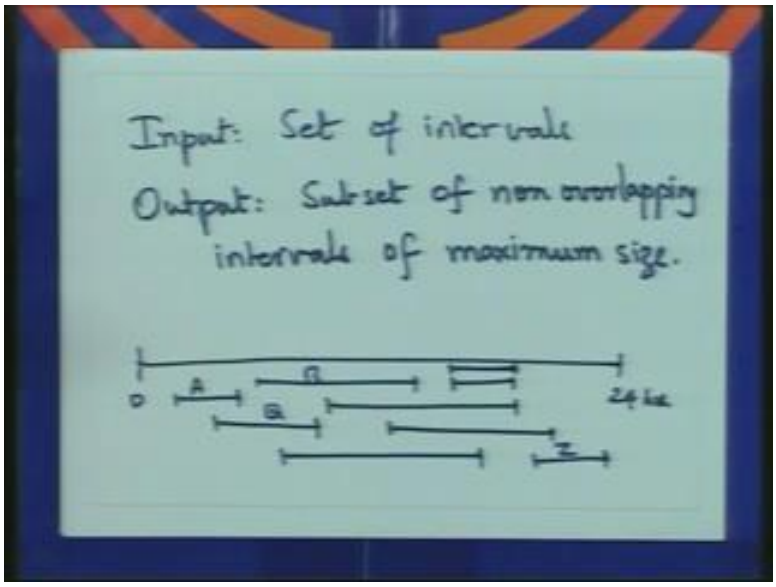
So, this in fact, is a one line proof all leaves must be in you know in some optimum independent set. So, you can start of by putting all leaves into the independent set you know there is an optimum which contains these remove their neighbors now, you can recurse. This is the good thing with greedy algorithm. If you pick a part of the output then you have a smaller input to deal with and you can just apply the same technique over and over again and the proof follows by induction. In fact, the algorithm also is inductive you can right it as a recursive algorithm you move part of the output. Now, you have a modified input on which to work on and ((Refer Time: 26:23)). So, that is the algorithm for this problem we move put all the leaves in the optimum independent set remove their neighbors. The tree will now break into many parts and you just recurs in each part. So, that is the algorithm for this problem. So, here is the next problem.

So, imagine that you are the systems are your administering a system. Let us say you have one really fancy computer at your command and a lot of people want to use it. So, they come and give you one so, you have a 24 hour sort of time slot in which to schedule them and they give you each of them comes and give you a time slot. Somebody comes and says 3 am to 5 am, somebody else says 2 pm to 7 pm these intervals could vary. Somebody could 12:30, 12:31 pm to 12:32 pm 1.05 am to 3.51 pm and so on. So, they give you sort of intervals. Each person gives you exactly 1 interval. We cannot spread his

job we cannot each day he has just one interval in which he can work we can give 1 interval.

So, many people come and they give you intervals in which they would like to occupy this machine. And they do not want shorter intervals they wanted for their entire duration which they have asked for. Your job is to pick these intervals some intervals essentially schedule people on this machine. So, that firstly, at each time just 1 person on the machine 2 people cannot work on the machine and the second thing is maximum number of people get to use the machine in a 24 hour slot there is a 24 hour slot. Your objective final objective is maximum number of people get to use the machine. It should not your policy should not be dictated on by any other ((Refer Time: 28:58)) like the amount of money they are willing to give you etcetera.

(Refer Slide Time: 29:17)

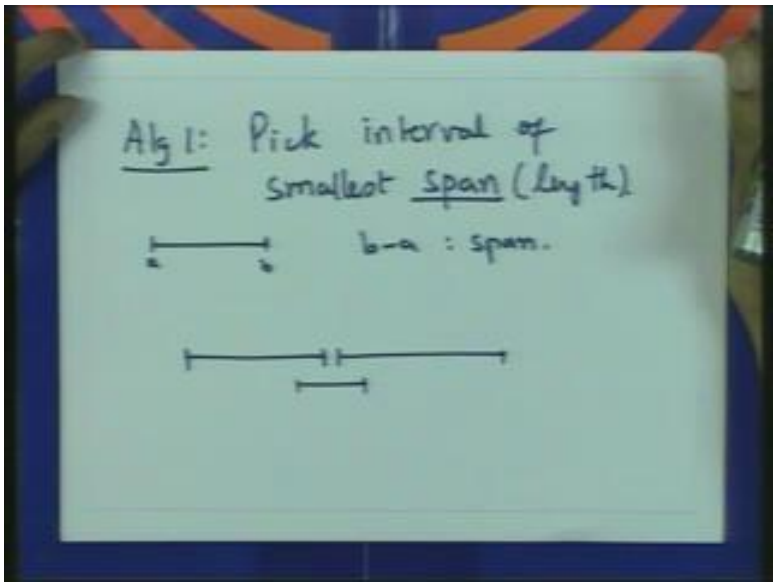


So, here is the, an abstract sort of origin of this problem you are given a set of intervals so, the input is a set of intervals. You can take them as closed intervals if you want. The output is a subset of non overlapping intervals of maximum size. You must output a subset of non overlapping intervals with maximum size. So, let us see what this looks like. So, here is let us say this is 0 to 24 hours so, the input looks like here is an interval. So, people give you various intervals some of them should overlap. All kind of funny ways 2 people may want the same thing that is also possible and so on. So this is how the input looks like. So, this is person A B Q Z each person gives you an interval and you

have to pick a subset of this intervals. So, that they do not overlap. So, if I pick A I cannot pick Q. I can pick B, but I cannot pick Q and I want to pick a maximum number of these intervals we said that I want to subset on these intervals which is of maximum size. So, that if I look at the intervals in the subset they do not overlap. So, a solution to this we will see in the in the next class. So, given a set of intervals you want a subset of overlapping intervals of maximum size our maximum size I mean the subset of maximum size.

So, this is the size of the subset which means number of intervals in the subset that should be maximum. For instance here I can if I can take 4 intervals then that are better than picking 3 intervals. Even though the 3 intervals may be longer that is not our goal our goal is to pick numbers just numbers. As many intervals that I can schedule you should pick that is the problem. So, the problem you are solving is this. The input in a set of intervals and we would like to pick a maximum number of them. So, that pair wise they do not overlap. So, if I take 2 intervals they do not overlap they are separating. So, how do we do this? Well, greedy technique the so, called greedy technique says builds a solution piece by piece. So, I am going to build the optimum solution interval by interval. So, which interval would you pick first which interval can you say in lie in some optimum solution which is what does your intuition say. Well, I want you want to pick intervals. So, that a large number of intervals. So, intuition I think the first thing that sort of strikes you is that if I pick intervals in small size. So, that the span is small maybe I can pick a large number of them.

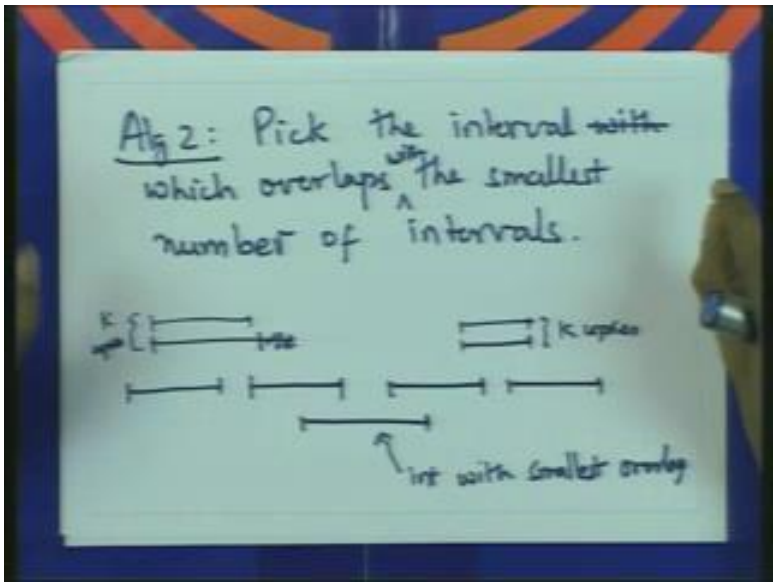
(Refer Slide Time: 33:33)



So, let us consider this. Let us call this algorithm 1 pick intervals interval of smallest size smallest span let us say this span or the interval. Suppose the interval is let us say goes from a to b; b minus a is the span of the interval. So, pick the smallest interval with smallest span. Well, smallest length is the one which you want. This seems to be a reasonable algorithm. This is what perhaps intuition suggests, but this intuition is faulty. It is very easy to construct examples where this algorithm fails. Here is on here is an example where the algorithm fails. Well, let us take a small interval then take 2 large intervals that intersect us the algorithm says pick this one.

So, this is the input; this 3 intervals form the input. The algorithm says pick this once you pick this you cannot pick these 2 fellows. Clearly optimum consists of picking these 2 big fellows these 2 big intervals. The optimum has value 2 value of this picked one. So, here the first sort of first thing that you can try fails. So, picking the interval of smallest length not fails. Well, let me think a bit more. Now, we start thinking a bit more what else could work here. If you look at this example, this small even though this interval is small it overlaps 2 of them while these big ones even they are big they overlap only 1. So, may be the right thing to do is to pick interval that overlaps with smallest number of intervals.

(Refer Slide Time: 35:45)



So, here is the algorithm 2; pick the interval with which overlaps the smallest overlaps with the smallest number of intervals. So, you see that the first algorithm fails you construct an example where the algorithm fails and here is an algorithm that at least works on that example. It seems fairly reasonable also that you pick an interval which overlaps the smaller. So, when I pick this interval I throw away the smallest number of intervals. I pick an interval I throw away the smallest number of intervals and now, I recurse. Once I pick an interval every interval that overlaps with this interval I am not going to pick. So, I throw those away and I recurse this should look very familiar to the previous problem. There you pick vertex throw away its neighbors and go forward. Here you pick an interval throw away the intervals with which this overlaps and you proceed forward.

Well, they are extremely closely related. In fact, we could form a graph right here. For an interval you have for each interval I can you can have an vertex and 2 there is an edge between two vertices if the intervals overlap. Then all you are doing for this problem is picking an independent set of maximum size in this graph. The way you get the graph is for each interval you have a vertex and you join 2 vertices in the intervals overlap and you are picking an independent set of maximum size that is what the problem is. This is an independent set of maximum size. So, like in the previous case why not pick a vertex with smallest degree. That is what this is an interval with minimum overlap minimum

number of intervals that it overlaps is nothing, but a vertex of smallest degree in this graph.

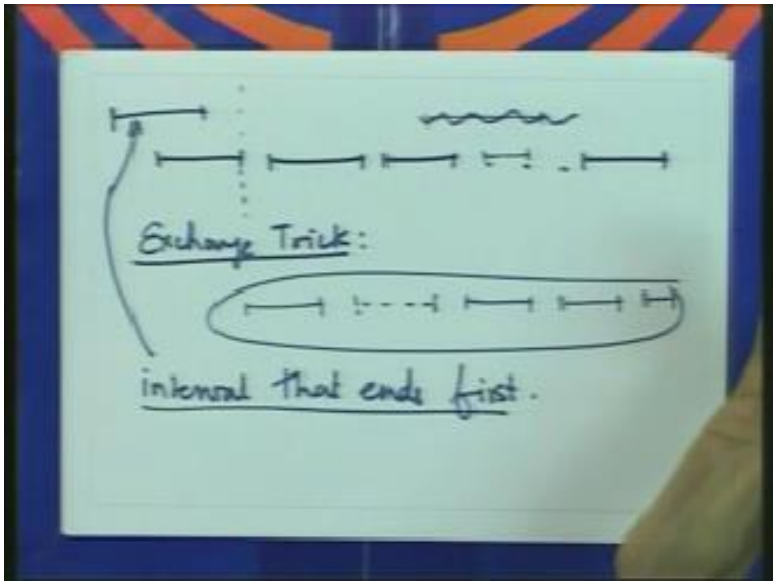
So, pick a vertex of smallest degree then you get its neighbors and recurse while I have written just one sentence there is a recursion going on. You pick this you pick an interval of smallest degree throughout all the intervals that it overlaps. And now you recurse you again pick an interval of. So, that it overlaps the smallest number in what remains throughout everything that it overlaps and so, on and continue. So, that is the algorithm does it this work? Well, it worked in the previous case. In the first problem this worked unfortunately in this case it does not work. So, here is the example. So, this is a bit more complicated the example now, which I have to construct. So, that this algorithm does not work for work in this case is a bit more complicated. So, here is the example.

So, far it looks like the old 1 we are going to now, apply more intervals and interval is 2. This is let me it ends here. I do not want this to overlap with this these 2 do not overlap with each other. Well, similar similarly here, I can have many copies let us say k copies. Similarly, here k copies we can have many copies. I just want to make sure that this interval does not overlap with those. Let us look at this example. So, these intervals there are k copies of these intervals. So, each interval overlaps with you know k minus 1 over here and 2. So, k plus 1 intervals is what this overlaps. This interval overlaps with k of them this interval overlaps with k plus 1 this similarly, these are sort of mirror image. This side is a mirror image of this side.

So, these intervals have a large overlap. The smallest overlap is this interval is a interval with smallest overlap with smallest overlap. In fact, it is 2 these 2. So, your algorithm will say pick this and recurse. Let see what happens when you pick this? So, when I pick this while I will throw away these 2, because these 2 overlap with this. So, now, I am left with imagine that these three intervals have gone. I am left with this side these intervals on this side. You can say that I can pick only one of these all of them pair wise overlap. So, I can pick only one of these I can pick only of these. So, and I have pick any one. It really does not matter all of them have the same degree. So, the solution size of the solution I construct is 3. The number of intervals that I have picked is 3; the first interval that I have picked interval with the smallest overlap and then 2 others after 2 others.

Well, if you stare in this paper a bit more carefully. So, if you stare at this a bit more carefully you can see that there is an optimum which has value 4 1 2 3 4. So, these 4 intervals do not intersect with each other. So, this algorithm produced this I mean 1 2 3 sorry 1 2 3 intervals which do not overlap with each other on the other hand optimum value is 4 1 2 3 4. So, this algorithm also does not work. The algorithm for roughly the same kind of problem as the previous one does not work in this case and you can see that constructing examples become harder and harder. I mean you can come up with really complicated algorithms for which it will be it may not work, but to prove that it does not work you have to really come up with all kind of complicated counter examples. So, now, what do we do what is the algorithm that works? Does any algorithm work? Well, what does the optimum look like? What is the picture of the optimum?

(Refer Slide Time: 43:21)



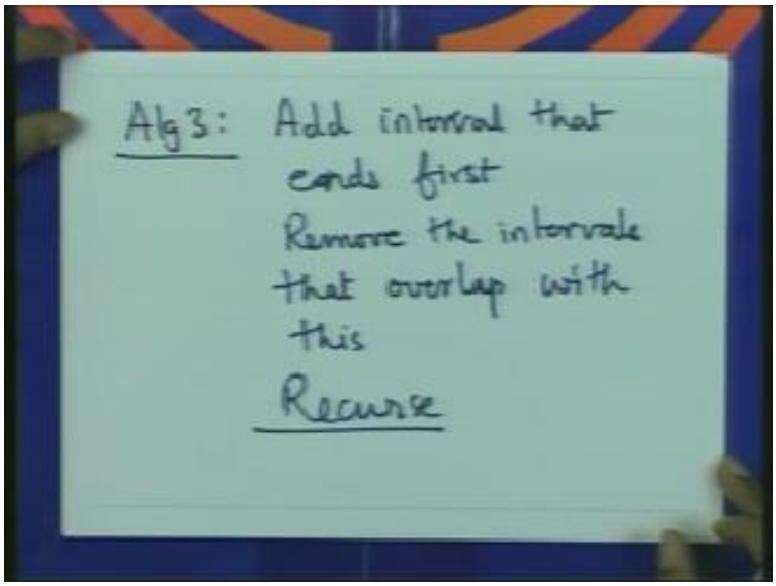
Let us look at this. So, it looks like this. The optimum looks like this. So, any solution must be a set of intervals that do no overlap with each other. So, they have to look like this. Now, the exchange trick what is the exchange trick here what do I mean by exchange trick here? Well, can I add an interval here can I add an interval to this solution well, certainly if I have space. Supposing let us say this is how the solution looks like. I can certainly add one here if there were one. Supposing these spaces are filled out can I add I may not be able to add, but can I add and remove one is that possible? Well, let us see supposing I add let us say add something like this. Now, this could intersect with more than 1 intervals. It can intersect with this it could intersect with well, it may be

something like this in which case adding this makes no sense. There is no exchange that sort of you know keep you going I cannot exchange this with some others. I mean if I exchange this then I need to remove more than one and the size of the optimum falls. Is there an interval that I can always add to this optimum solution and maybe we move at most one? Is there some interval means I can always do?

This I can always you know add to this solution that I have and maybe we move at most one. This requires a bit of thought will tell you that there is an interval and this is the interval that ends first. So, let us see what this mean? So, supposing what do I know about the interval that ends first it has to end before this. So, the interval sends somewhere here. This is the interval that ends first.

Well, now we notice that I can add this and remove at most interval that is the first interval. This will certainly not intersect with the second interval, because the starting time of the second interval comes after ending time of the first interval and ending time of this interval is either this or before. So I can always add the interval that ends first and remove the first interval it maybe. If it ends before ((Refer Time: 46:20) and increasing the size of the solution. So, exchange trick well, exchange trick plus some intuition plus whatever tells us that I can add the interval that ends first you give me any solution. I can add an interval that ends first may be remove the first interval certainly the size does not go down. And this is then this is what, now leads us to the algorithm 3 which actually works.

(Refer Slide Time: 47:00)



Algorithm 3 says add interval that ends first remove the intervals that overlap with this and recurse. This is the algorithm 3 and this actually does give us what we want. So, instead of recursing of course, you could sort the algorithm the intervals by ending time so, you can write this as a iterative procedure. You sort the intervals by ending times and pick the first interval the interval that ends first throughout the interval that this overlaps and then continue the iteration. So, this also this works so, why does this work? Actually it is this exchange trick that makes exchange trick that we saw that is what makes it work and it will prove it by induction really.

First thing to prove would be show the interval that there is some optimum solution which contains the interval that ends first. And the proof we have actually seen so, what you do is look at any optimum and look at the interval that ends first go back to previous figure. Supposing this where the optimum solution? Here is supposing it already contain the interval that first we have done. So, there is an optimum solution that contains the interval that ends first otherwise take the interval that ends first it looks like this. It has to end either at the same point or before add this remove the first interval and we now have an optimum solution which contains the interval that ends first. There is some optimum solution that contains the interval that ends first.

So, I pick this as part of my solution. Remove the intervals that overlap with this and now recurse on what remains, and since I am recursing on a smaller set by induction the

algorithm will find the optimum in what remains and ((Refer Time: 49:40)). So, when I remove this and when I remove the intervals that overlap the interval that ends first. The size of the optimum falls by at most one that you can see just by looking at this figure that we have. I added this interval to remove this and what remains was this you know the rest of it. It just dropped the optimum drops by 1 by induction I will find something of this size. The algorithm finds something of this size and the algorithm output this plus 1 that is this interval. So, the size of the solution that ((Refer Time: 50:20)) output is the same as the size of the optimum which is what we want.