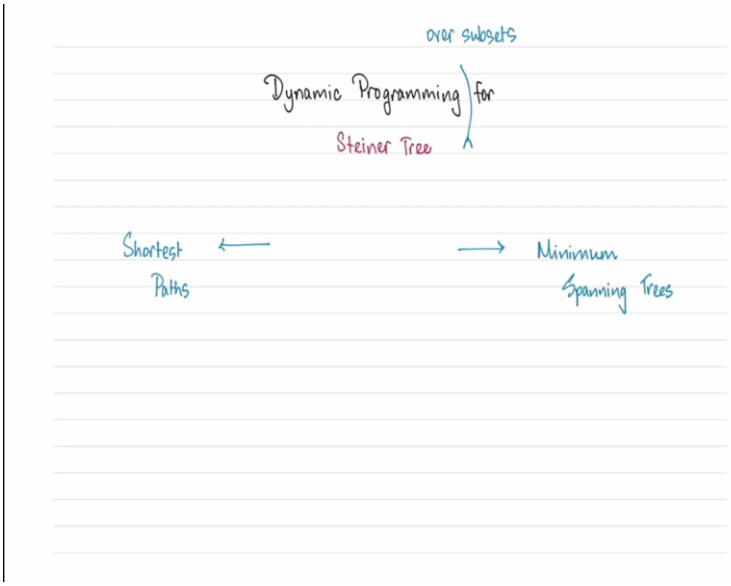
Parameterized Algorithms
Prof. Neeldhara Misra
Prof. Saket Saurabh
Department of Computer Science Engineering
Indian Institute of Technology – Gandhinagar IMSC

Lecture – 32
Dynamic Programming over Subsets for Steiner Tree

(Refer Slide Time: 00:11)

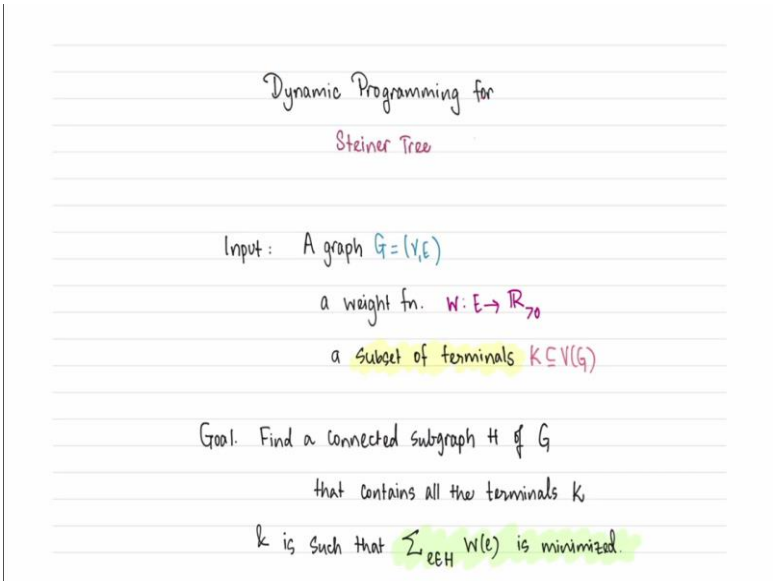


Welcome back to the second segment of the first module of week 7 in parameterized algorithms. As you might remember we have been talking about dynamic programming and in particular a style of dynamic programming that involves going over all subsets of an appropriate set in the problem. So, last time we illustrated this using set cover and for this segment I want to focus on a problem called Steiner tree.

So, Steiner tree is again a natural graph optimization problem which simultaneously generalizes both shortest paths and minimum spanning trees. Both of these are classic problems which admit polynomial time algorithms and you have likely seen these problems already in the first algorithm course that you have probably done before this, but let us now talk about Steiner tree and let us try and see how it generalizes both of these ideas.

I should say that these generalization does come with a price in the sense that Steiner tree in its full generality is envy complete problem which is why we will be interested in coming up with FPT algorithms for it.

(Refer Slide Time: 01:17)



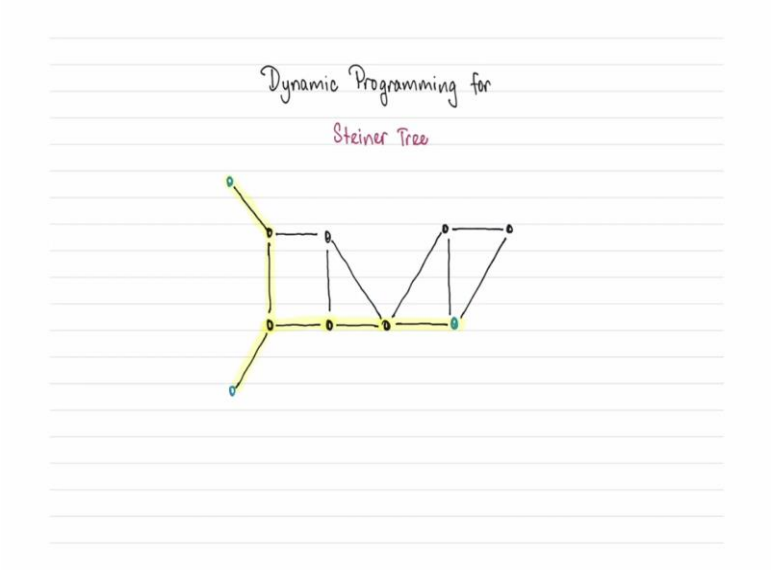
So, the input to an instance of Steiner tree is an undirected graph with edge weights and these edge weights will be assumed to be positive and the key thing that we are given is a subset of vertices which we identify is being special. We call these vertices terminal vertices throughout and every vertex that is not a terminal vertex we will call it a Steiner point or a Steiner vertex.

So, given this solution what the goal? Well, the goal is to figure out the cheapest way to connect all of these special terminal vertices. So, in particular what we want to do is find a connected sub graph H of G which contains all the terminals K such that the total weight of the edges that are involved in the sub graph are minimized. So, this is the cheapest way to bring all the terminals together.

Now, notice that if you just had two terminal vertices than the cheapest way to connect them is by a shortest path. On the other extreme can you think of what subset of terminal vertices will make Steiner tree the minimum spanning tree problem? So, if you thought about that for a moment you may agree that when K is equal to the entire set of vertices than this really boils down to finding a minimum spanning tree.

But on the other hand when we want to just connect some given subset of terminals then it is not necessarily either a shortest path kind of a task nor spanning tree task. In some sense a spanning tree is going to probably be much more than what you need and let us say that you try to use shortest path to some sort of a sub routine to connect terminals pair wise you might find that it is easy to get into situations where your answers are going to be suboptimal.

(Refer Slide Time: 03:09)



Now just to make this a little bit concrete let me show you an example of a graph with 3 vertices that are colored differently you could think of these as your terminal vertices and you could think about what is the best way to connect them and assume for now that all the edge weights are uniform so you do not have to worry about optimizing for weights in particular, but you just trying to get the structure that has now the smallest number of edges.

The first thing to notice is that whatever structure you have is going to be cyclic because since you want to minimize the total weight of the edges and in this case the total number of edges it does not make sense to have any redundancy in whatever structure you identify. This is partly where the name Steiner tree comes from because whatever you output without loss of generality is always going to be a tree because of our assumption about weights being positive. Now, for this example let me just highlight a one Steiner tree for you.

Now you could come up with bunch of different ways of doing this and what I have here may or may not be optimal. So, we will leave it as an exercise for you to check if you can come up with a tree that has smaller number of edges, but here is one which is essentially a connected sub graph of the original graph G that does have all the terminal vertices. So, this is definitely a Steiner tree.

As I said it may or may not be the best possible one for this example. So, I will leave you to think about how you would establish either that this is the best that you can do or that there is something better, but you do have edges with weights then you have to be careful also about

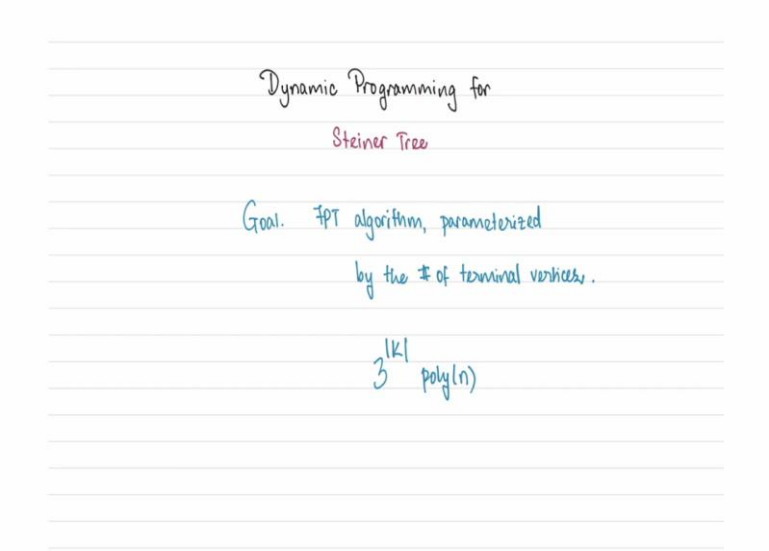
not just thinking of it in terms of optimizing for the number of edges. It is possible that you have two competing structures one of which has lots of edges, but they are all relatively light.

And the other one has the small number of edges, but they are very heavy and it is entirely possible that the first one wins out over the second in terms of total edge weights so do keep that in mind. On the other hand when the edge weights do happen to be uniform then you could equivalently optimize for the number of Steiner vertices instead of the total edge weight as we are doing in the more general version of the problem.

So, our goal here quite predictably is going to be to come up FPT algorithm for the problem, but the question is with what parameter. If we go back to the problem statement you will see that there are a couple of natural choices here since you just had such a long discussion about tree you might want to say let us parameterize structurally by the tree width of the graph G. On the other hand it is possible that you would like to parameterize by either of the highlighted quantities here in the problem statement.

One would be the size of K which is the number of terminals and the other natural choice is what we have been calling the standard parameter which is the quality of the solution that you are able to come up with. So, for this discussion we will be focusing on the size of K as our parameter.

(Refer Slide Time: 06:12)

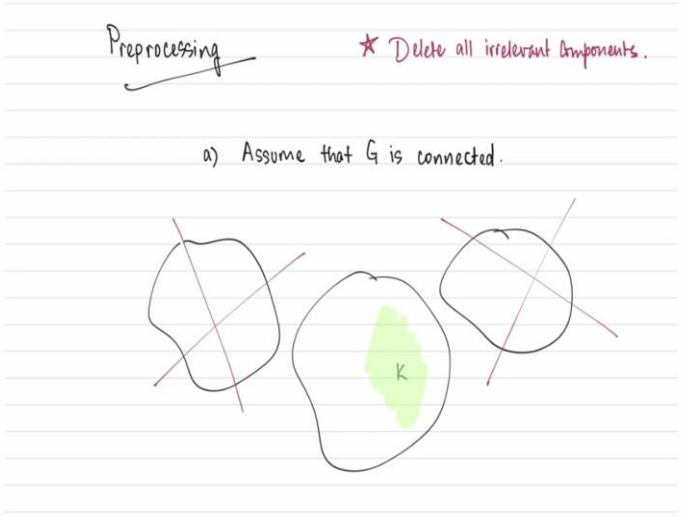


And our goal will be to come up with an algorithm whose running time is 3 to the K with polynomial overhead. So, if you were to take a hint from the running time itself you might

guess that this has something to do with taking subsets and then taking subsets of the subsets that are under consideration. So, somehow something like this might come into play and if you are anticipating that then you would be quite right.

Before we get to a description of the algorithm though let me first present some simplifying assumptions that we can make without loss of generality. I will present these are as preprocessing steps that you can employ before you actually get to the main algorithm.

(Refer Slide Time: 06:54)



So, first we are going to assume that G is connected. Feel free to pause here for a moment to convince yourself that this is indeed an assumption that can be made without loss of generality and comeback when you are ready. So, hopefully you thought about that for a moment. Now think about what happens if G is not connected? There are two possible situations that can arise with respect to the terminals.

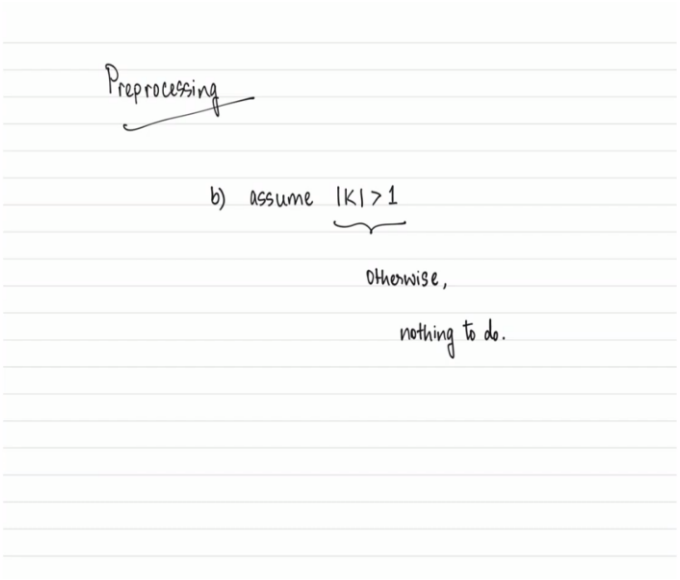
Either the terminals are spread out across more than one connected component or they are all sitting inside just one single component. In the first case if the terminals are spread across more than one connected component then the answer is clearly no and you can just say as much and your algorithm can stop here. The reason for this is that if the terminals to begin with are already in different connected components.

Then no matter what throw into your solution you will never be able to connect them anyways; so the answer here is a clear no. On the other hand if your terminals are all concentrated in just one component then you could simply get rid of the ones that do not

contain any terminal vertices. These will never be relevant to any optimal solutions so you can just throw them out.

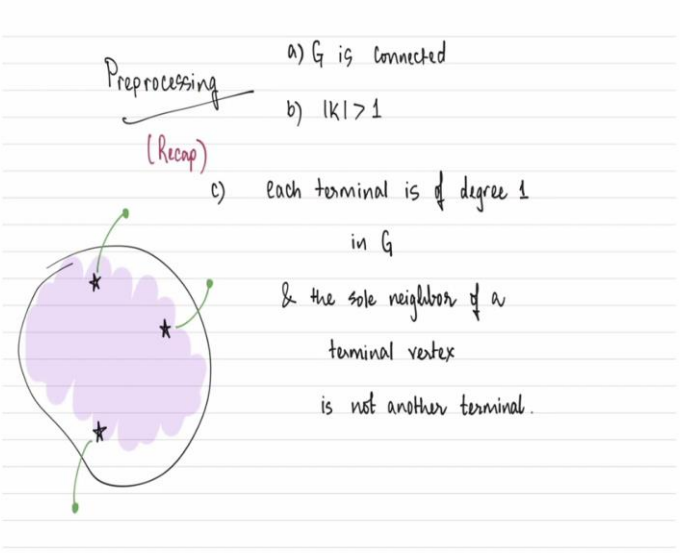
Therefore, in either case you end up with a solution where either you still have to solve the problem when G is connected or you have already solved the problem. So, this is why we can assume that G is connected.

(Refer Slide Time: 08:14)



The second assumption is really quite straightforward which is that we are going to assume that there is more than one terminal vertex. Otherwise this problem is not even interesting if there is just one terminal vertex there is really no work to be done at all.

(Refer Slide Time: 08:27)



Now the third and the last assumption is the following. We assume that each terminal vertex is a degree 1 vertex in our graph and it is only degree 1 neighbor is not another terminal vertex. This may sound like there is a lot going on, but it turns out that this assumption really does simplify the way we talk about certain aspects of our algorithm especially when we are trying to argue the correctness.

So, it is useful and it is very doable. So, this is a fun one to puzzle out. So, feel free to pause here and think about how would you ensure this To be clear your graph or your instance may not have this structure out of the box. So your instance may not look like this and what you have to do to ensure that this assumption can be made without loss of generality is to come up with a way of transforming your instance into an equivalent one where the property is in fact true.

So, how would you (()) (09:23) the terminal vertices just get that thought and comeback when you are ready. So, one way of doing this would be the following. You can sit all the original terminal vertices and simply give each one of them a new degree 1 neighbor. So, these are artificially introduced dummy vertices all of which have degree 1 and now you declare each of these new vertices to be the actual terminal vertices.

Once you have done this notice that in the new instance you do have the property that you want which is that every terminal vertex has degree 1 and its degree 1 neighbor is not a terminal vertex. It is an old terminal vertex which is no longer a terminal vertex in this new graph. Now, if you look at it structurally you will see that basically there is a matching structure between the old and the new terminals.

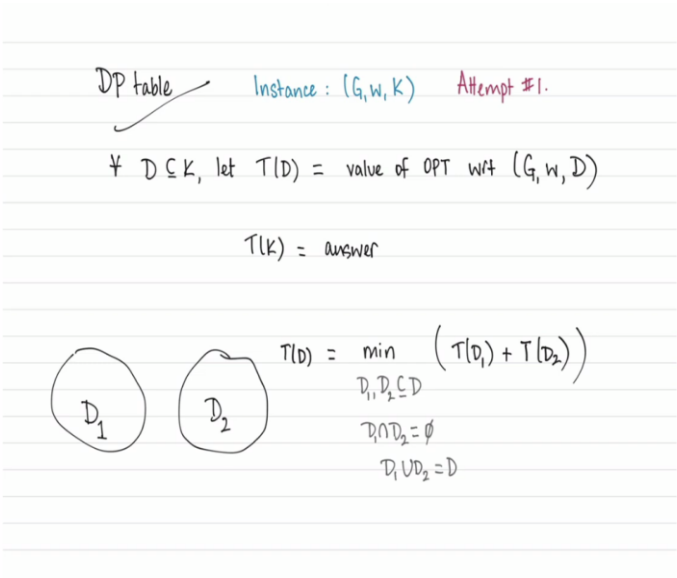
And you can convince yourself that this Steiner tree that connect the old terminal vertices in the original instance are in one-on-one correspondence with the Steiner tree that connect a new terminal vertices in this new instance. You could declare the weights of these matching edges to be just one and once you found a solution in this new graph you could just subtract the number of terminal vertices from the solution that you obtain to obtain the cost of the best Steiner tree in the old instance.

I am not going to formally establish the equivalence of these two instances here, but hopefully it is reasonably intuitive and it is a good exercise to actually write it out formally to

be sure. So, that brings to the end of our discussion on the preprocessing steps. So, just to recap after you are done with the preprocessing you can assume that G is connected there are at least two terminals and every terminal has degree 1.

And its degree 1 neighbor is not another terminal. With this, let us go back to our goal of coming up with a 3 to the K poly n type of an algorithm for this problem with K being the number of terminals. We said we are going to be doing dynamic programming over subsets. So, subsets of what? Here in this problem the most conspicuous, the most obvious set that is starring at us seems to be the set of terminals. So, natural thing to try and do is to work with subsets of terminals.

(Refer Slide Time: 11:46)



So, here is a first shot and what your DP table may look like. So, for every subset of terminals let us say that our table stores the value of the best way of connecting that subset of terminals. So, D denote some subset of terminals. We are saying that T of D is the value of the best Steiner tree for the subset D. Notice that with this definition you can get the answer that you are looking for by simply looking up T of K where capital K denotes the set of all terminals.

So that is the final answer. So, these sub problems are certainly working with they do lead you to the final answer if you can compute them, if you can figure them out. So, what would be a good recurrence for this proposed DP table? Give this a thought for a moment and comeback when you have a proposal and we can exchange notes. Hopefully, you had a chance to think about this for a moment and natural thing seems to be to consider all possible partitions of the set of terminals.

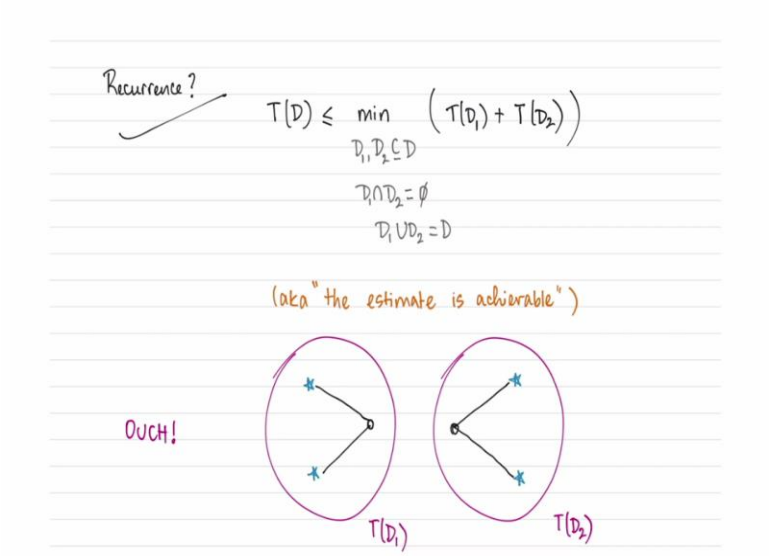
Remember even to begin when we were looking at the running time we said that maybe this has something to do with taking subsets and then to compute the value for each subset maybe we have to do another layer of looking into all possible subsets. So, let us look at all possible subsets of D while computing the value of T of D. In particular let us look at all possible ways in which D can be portioned into two sets D 1 and D 2.

So, by a partition I mean that the union of D 1 and D 2 is D and the intersection of D 1 and D 2 is empty. So, we go over all possible ways of portioning the terminals and then we try to see can be separately connect D 1 and separately connect D 2 and just combine them. So, the best way of connecting D 1 combined with a best way of connecting D 2 and you take the best over all possible ways that you can partition on the set of terminals into two parts.

So, this seems like we are doing a fair amount of work and hopefully we will catch the best solution. So, think about whether this works and think about how would you show the correctness of a proposed recurrence like this. Can you spot something that is problematic I mean the top of the slides say attempt number 1 which maybe suggestive that there are more attempts coming up and perhaps there is something wrong with this or maybe this is just a missed direction and a silly trick for me to misguide you.

So, I am not going to reveal whether this proposed recurrence here works or not for at least another 10 seconds. So, please feel to pause the video here and think about whether you can prove the correctness of this claimed recurrence here. So, hopefully you had a chance to think through this and you have come to a conclusion one way or the other. So, typically when we prove an equation like this we split it into two parts the lower bound and the upper bound corresponding to the estimate being achievable and the estimate being tight.

(Refer Slide Time: 14:53)



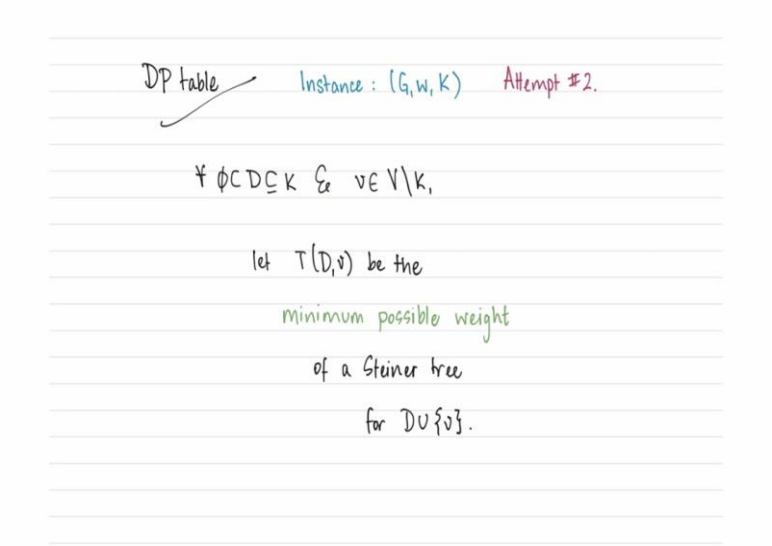
And in this case let us start with the upper bound which is basically saying that the estimate is achievable. It is a reasonably estimate we can at least manage what is being claimed on the right hand side maybe even better, but at least we can get to this. So, how would we show this? The typical strategy is to take the solutions that basically witness the numbers T of D 1 and T of D 2 and try to somehow combine them to get a solution for D.

So, let us say that we have some partition which actually witnesses this minimum let us say that some fixed D 1 and D 2 and let us say T of D 1 ad T of D 2 are two Steiner trees that connect D 1 and D 2 respectively. So, for instance, you could have D 1 being the two terminals to the left of your screen and D 2 being the two terminals to the right of your screen and here are the Steiner trees that witness the best ways for these terminals to be connected.

Now do you spot a problem here? The problem is that the natural object, but has a cost of T of D 1 + T of D 2 is simply the union of these Steiner trees, but notice these Steiner trees are actually not connected and therefore they do not form a Steiner tree of the full terminal set consisting of all the four terminals. So, the problem with this recurrence is that what you have on the right hand side may not actually be valid in terms of the patching process.

So, this equation simply may not be true. In some sense to fix our recurrence what we need is some information about piece of glue that holds the two paths together. So, we know in some sense that the solutions for these two pieces D 1 and D 2 must share something in common because ultimately it is a part of a larger connected piece if that is one way to think about it and that motivates a revised attempt at a DP table semantics.

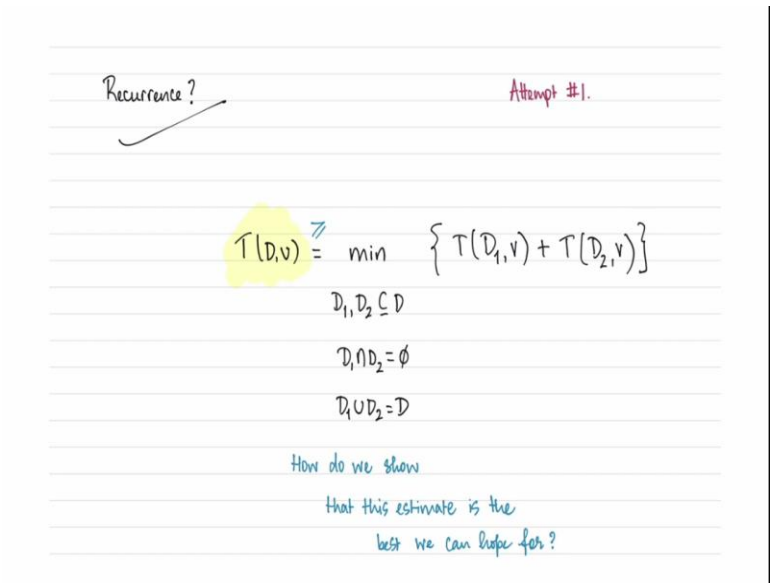
(Refer Slide Time: 17:00)



So, let us say that we allow for another index in our DP table which is a vertex which is not from the terminal set and basically what we want is the minimum possible weight of a Steiner tree for these subset of terminals that we are working with D and this extra vertex v. So, now the hope is that the solutions that you get for D 1 and D 2 both contain these common vertex. So, when you patch them together you get something that is connected at least via the vertex v.

So, hopefully this new semantic will give us a little more control over how we do the upper bound. So, let us try and come up with a recurrence for T of D, v hopefully the semantic here are clear.

(Refer Slide Time: 17:50)



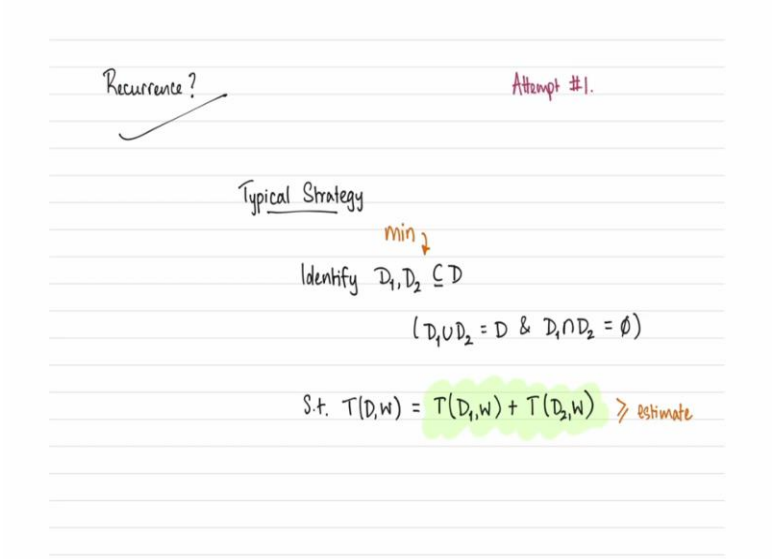
And natural recurrence again just like last time would be to say let us go over all possible partitions of the terminal set and again combine the solutions as we were doing before. Again there is an attempt number 1 sitting on the screen so maybe that is already a hint that this may not be the final thing that we are hoping to get to, but let us think about it. So, here is a proposed recurrence it looks fairly natural.

And I want us to spend a minute here thinking about whether it works or not and why would it work and why it would not work? So once again take a moment and really try to grab some pen and paper and work through a potential proof of this recurrence. Either you get to a proof or you get to a point where you feel stuck and then we can come back and think about whether we need to revise the recurrence or whether we need to revise the DP table semantic or whether we are actually done at this point.

So, come back when you are ready and we can exchange notes. So, by now hopefully you have had a chance to play around with this and you have your own thoughts about what is going on here. Now just like before let us try to prove this equation and as we have been saying proving such an equation typically amounts to breaking it up into a lower bound and an upper bound kind of inequality combination which then ultimately implies the truth of this equation. This time let us go for the lower bound.

So, what I want to say is that this estimate that we have here on the right it is tight, it the best that you can hope for and whatever you do, you must incur a cost of at least what is specified by an estimate. How do we typically prove something like this? Well, what we would want to do normally is to work with solution that witnesses the value of T of D v. We say look let us say the best Steiner tree that we can come up that connects the terminal subset D using the vertex v and hopefully we can split this up neatly into two subsets of terminals D 1 and D 2.

(Refer Slide Time: 19:58)

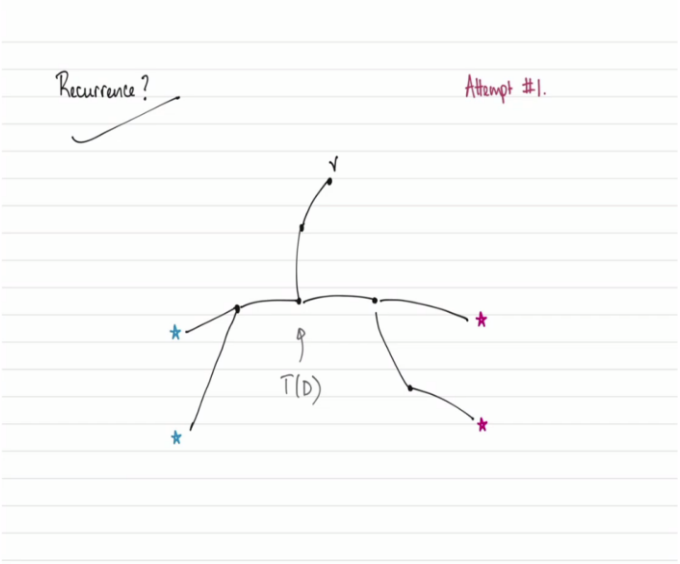


And from our solution hopefully we can extract two different solutions. One for D 1 and one for D 2 both involving the vertex v. So that if these two solutions are such that this expression here holds that T of D, W equals T of D 1 W + T of D 2 W. By the way I think there is a change of variable here which was unintentional you could read this as v and that would be probably more appropriate here.

But anyway the point is that if you can show something like this then you would be in business and the reason for that is that if you recall the expression that we have it is really a min over all possible choices of D 1 and D 2. So, if you are able to come up with one choice of D 1 and D 2 for which this equation holds then you know that because the expression that we have on the right hand side which is our estimate is a min over all of these choices.

We know that our estimate is either what you have here or it is even smaller because it is a min. So, it is either there is a smaller, but that is exactly what we wanted to show. We wanted to show that T of D W is at least the estimate that we have. So, if you are able to do this then we are in business.

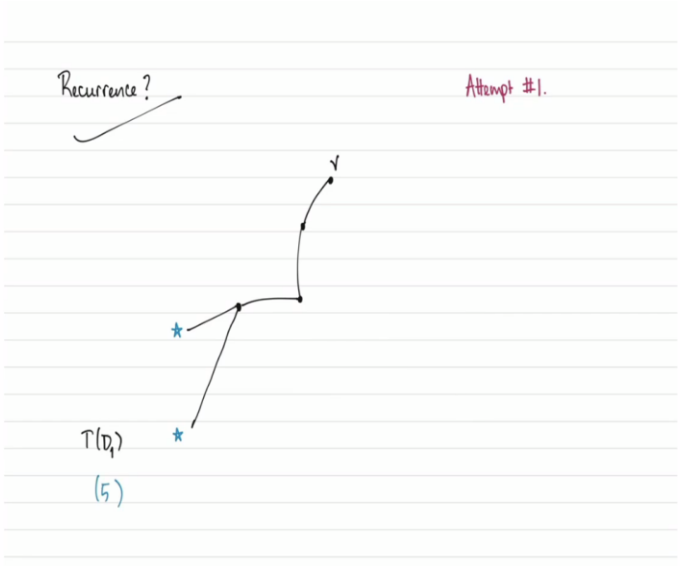
(Refer Slide Time: 21:12)



So, let us try to look at a typical solution for a set of terminal. So, your solution could potentially look something like this and now the point is that we have these 4 terminals that are highlighted for use these orange star vertices. Now you could try to split up these terminals in whatever way you like, but let me show you how you might get a little bit stuck when you are trying to show an equality of the form that we have here an expression of the form we have here.

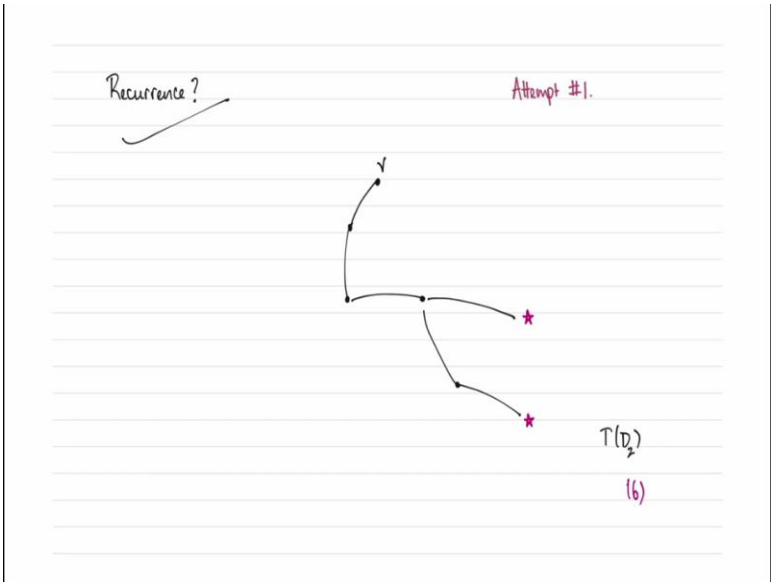
So, in particular let us say that we try to split the terminals D 1 and D 2 with D 1 being the two blue vertices to the left of your screen and D 2 being the two pink vertices to the right of your screen. Now from this structure that we have here we want to pull out solutions for D 1 and D 2.

(Refer Slide Time: 22:00)



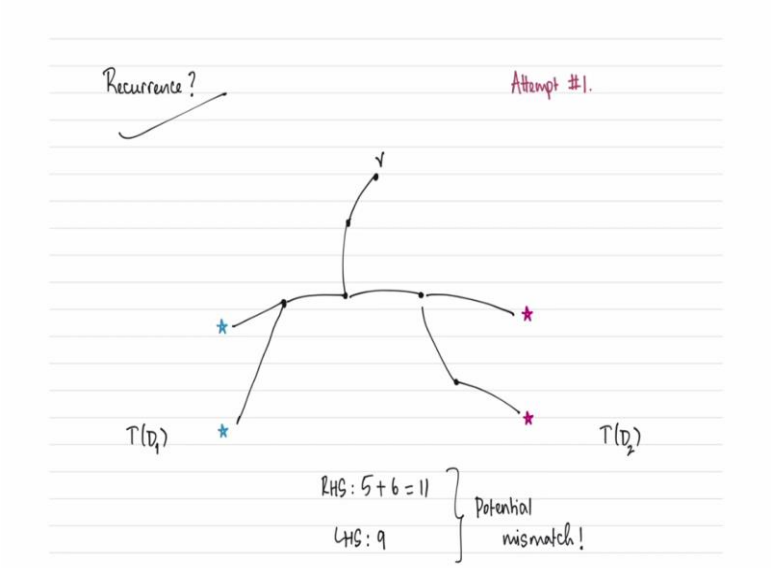
The most natural way to do this seems to be to say that okay let us throw away everything that we do not need when it comes to connecting D 1 and v and this is what we get and notice that the cost of this solution is 5.

(Refer Slide Time: 22:14)



And similarly if we throw away everything that we do not need for connecting D 2 then this is what we get and notice that the cost of this solution is 6.

(Refer Slide Time: 22:24)



So, when we put these two solutions together we get 6 + 5 which is 11, but now if you consider the value of T of D, v from just this picture you know that the value of that is going to be 9 there are 9 edges in here and what is happening is that when you combine these solutions you have two edges that are getting over counted twice. So, you have a mismatch in the expressions that you were hoping to work with.

And this is something that would happen even if you split the terminals in a slightly different way as well. So, I could go on with actually explicitly enumerating all the possible ways in which you could split your terminal vertices, but I am going to leave that for you to explore a little bit from this example. Now, I will admit that this discussion that we have been having may come across little bit vague because perhaps you might argue that this is not the only way to prove such an equation may be we need a more concrete counter example showing why the inequality does not hold if you were to work with the recurrence that I proposed.

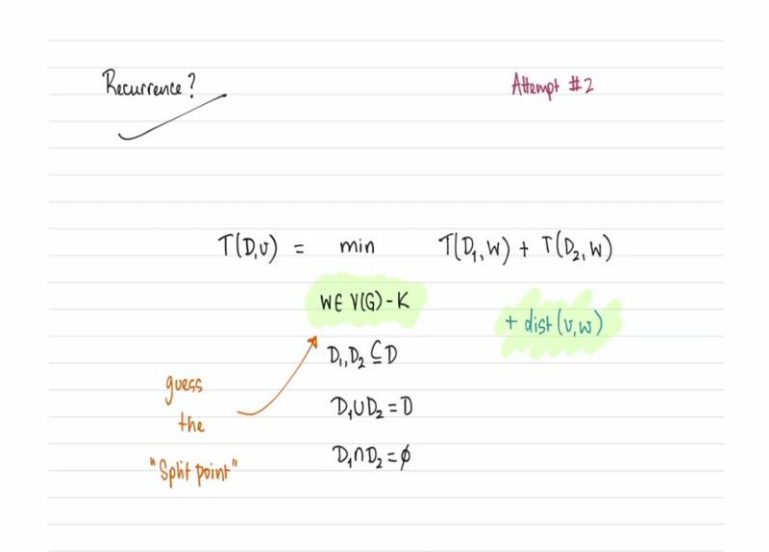
And I claim that you could take an example like this and turn it into a more formal counter example, but I really wanted to emphasize more how the most natural proof approach might lead you to a place where you get stuck because this motivates the recurrence that we will actually be working with and hopefully when you see the proof for the recurrence that ultimately works you will be able to relate to how the modification in the recurrence helped with cleaning up how the proof goes.

So, I hope that you found this little D 2 to be useful as for coming up with more formal explanation of an actual counter example. I will leave that to you as an exercise to work through, but we will look forward to discussing this with you in the comment section. In the meantime let us go back and actually try to fix the recurrence now that we know what the problem maybe.

Notice that the main issue seems to be that when v is a non branching vertex and this sort of a common path up to some point then that is something that is getting over counted in both of these solutions in the proof that we were trying to come up with. So, what if we could guess their first point where the branching does happen so that we can separate out the way the cost work for these two subtrees.

And the cost for the common path the path that seems to be getting over counted with the way we were doing it right now.

(Refer Slide Time: 24:59)



So, that motives our second attempt that coming up with our recurrence. So, instead of simply going over all partitions, let us go over all partitions and also guess for the choice of the branch points. Now it could well be that the branch point is the vertex v itself in which case actually even with the previous recurrence we would not have gotten stuck the way that we did, but we cannot be sure that the branch point is the vertex v.

Now you might say that for a Steiner tree it seems like you should never have a picture like the one that we were looking at in the previous slide because it seems like the whole piece from V to W is redundant and you can throw it away to get a better Steiner tree, but when you throw it away you get a Steiner tree that does not involve the vertex v and this will somehow a requirement for us.

If you think about intuitively the point is that these intermediate Steiner trees may be useful pieces that go into the largest Steiner tree. I do agree that the Steiner tree that you produce the very end will not like this, but it is pieced together these are like puzzle pieces that come together to form the whole picture and it is possible that the smaller pieces may need to have this sort of a structure so that they can go and fit into the bigger landscape.

So, hopefully that little bit of attempted intuition is useful. So, with that said let us come back to our fixed recurrence here. So, remember what we are trying to do is guess the branch point and allow for the possibility that this branch point is different from the vertex v. So, what we were saying is that it seems like whenever you have a situation where your branch point is

different from the vertex v than you are doing some redundant work and why even doing it to begin with.

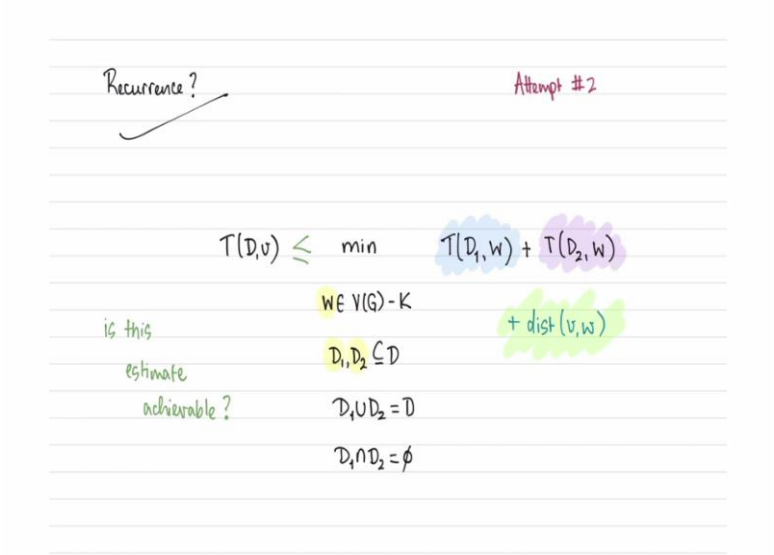
But our point here is that it is conceivable that when you are working the restriction that you must include v than a structurally the only way for v to connect these dissipated pieces is to go through this non branching sort of a section before it can truly become a more tree like structure. So, that is conceivable and we need to account for that possibility we cannot say that the v's for which this sort of a behavior happens are not worth considering.

So, these may be important pieces to your final puzzle and that is why we would like to account for them very explicitly. So, we go over all possible choices of these branch points whether they are the same as v or not, but when they are different from v than the cost of branching at W means that we have to take a straight path from v to W and the cost of doing that is going to be the distance between v and W.

This is something that you can compute in polynomial times. So, we set this piece aside. Remember this was the path that was getting over counted before now we are careful to count just once and the cost of the rest of the structure can be obtained by simply looking up T of D 1 W and T of D 2 W. So, this is essentially patching up the terminals in D 1 and the terminals in D 2 via the gluing vertex W.

So, hopefully this recurrence make sense some sort of an intuitive level, but I think given all the false starts that we have had so far it would be a good idea to formally prove the correctness of this equation.

(Refer Slide Time: 28:24)

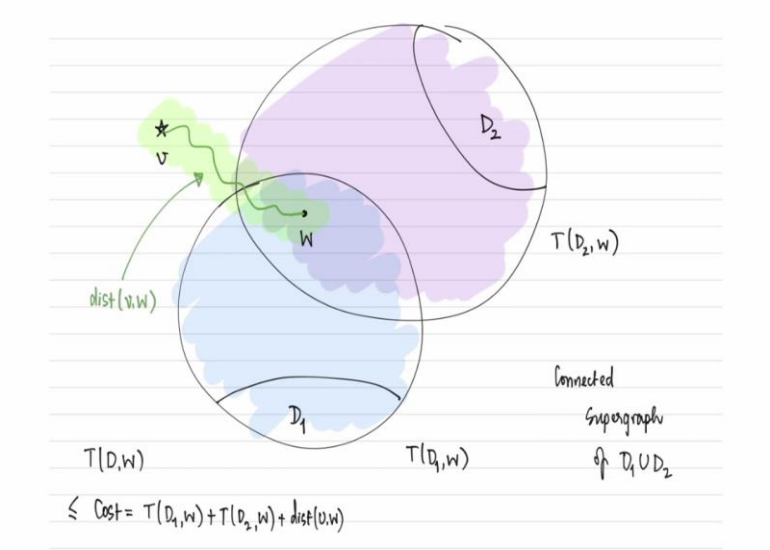


So, let us start with the upper bound let us say that we will first argue that this solution that is propose of this number that we are estimating is a valid upper bound for the thing that we are trying to compute. So, suppose this expression on the right hand side this minimum is witnessed by some fixed W D 1 and D 2. So, let us just fix a choice of W D 1 and D 2 which realize this minimum and let us go ahead and look at what is going on there.

So, T of D 1 W and T of D 2 W these numbers are both backed up by the induction hypothesis by the assumption that things work out properly for smaller values. These expressions highlighted in blue and purple are backed up by actual Steiner trees of these corresponding cost that connect D 1 and D 2 respectively and that also have the vertex W. On the other hand we also know manually by just computation that distance of v, W is the quickest way of getting from v to W.

So, knowing what we know from the induction hypothesis and the computation of the shortest path distance let us just piece these things together.

(Refer Slide Time: 29:40)



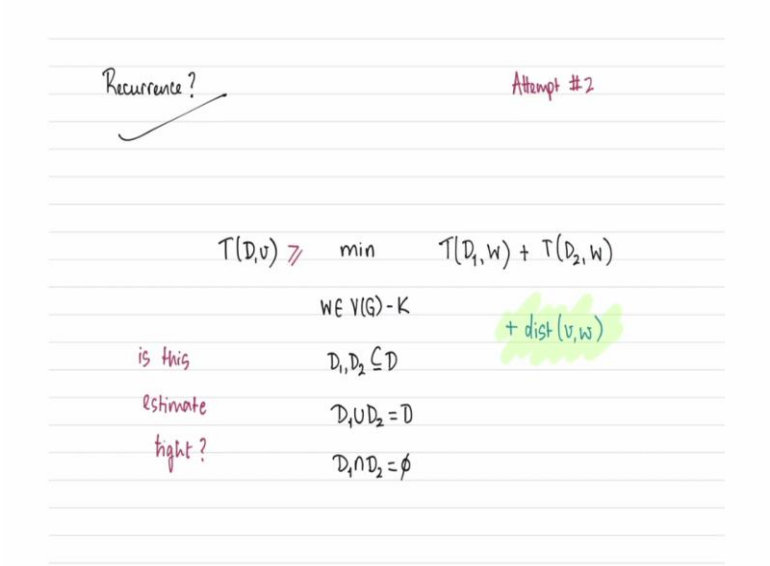
So, we have the shortest path from V to W than we have T of D 1 W being witnessed by some Steiner tree that connects D 1 and then we have D 1 D 2 W that is witnessed again by some Steiner tree that connects all the terminals in D 2. Notice that these two Steiner trees definitely have the vertex W and common by definition and it is conceivable and that they have more things in common as well it may or may not happen.

But in principle it is possible that when you put all of these pieces together what you have is not even a tree, but nonetheless what you do have is the connected super graph of D 1 union D 2 which contains the vertex v. So, remember that we are tasked with finding the best possible Steiner tree for D which is D 1 union D 2 which also contains the vertex v and this if not already a Steiner tree is a super graph of some Steiner tree.

So, you could throw things away, but if you throw things away you only improve the cost. So, what we have here is a solution whose cost is at most the expression that we have on the right hand side which was essentially our estimate. So, we definitely have a solution whose cost is either exactly what was specified by our estimate or it is even better if you can afford to throw things away from here.

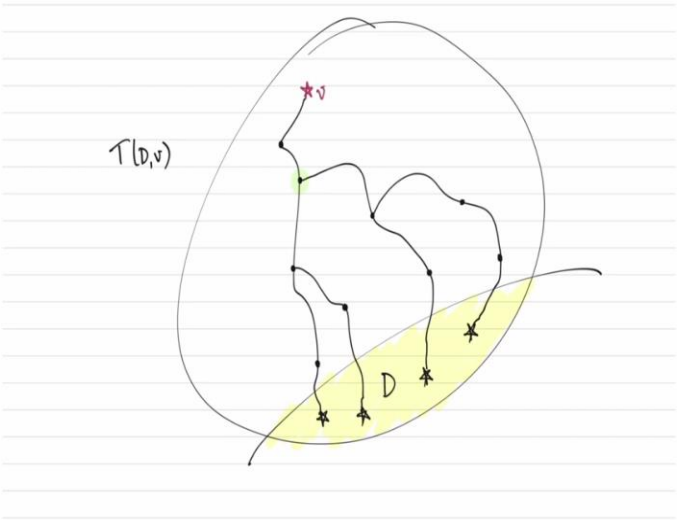
So, basically this shows the upper bounded. It shows that our estimate is realistic it is achievable.

(Refer Slide Time: 31:05)



On the other hand we want to show that the estimate is also tight that you cannot do any better. So, to do this let us again start with a solution that witnesses how these being connected via the vertex v.

(Refer Slide Time: 31:21)



And let us say that solution looks something like this. So, what you see here is a connected subgraph of the original graph which contains all the terminal vertices that we are interested in which are the vertices D and it also contains the vertex v and this is all that our solution is supposed to do. So, this is what we have here and among all such feasible solutions what we have here is the one that has the smallest cost.

So, now to be able to actually establish the inequality remember our standard technique what we would want to do is somehow split this structure that we are starring at here into three

pieces that accounts for the 3 terms here. Of course, it is possible that the distance term is 0 if there is no meaningful choice of W that would happen if v turned out to already be a branching vertex in which case you do not really need to invoke this third term.

And that would be fine, but for instance in this picture here you will see that v turns out to be not a branching vertex so we do need to actually identify an appropriate choice of W and actually use the distance term here. Remember this is exactly where we got stuck previously when we were not using this more nuanced approach. So, hopefully the difference that this is making will become clear as we work through the proof.

So, if you like take a moment here to pause and reflect on what would be a natural way of breaking up this solution that you see here into 3 pieces so that each of those pieces account naturally for one term each in the expression that we have here on the right hand side which was our estimate for what the answer should be. Just think about that for a moment and come back when you are ready.

So, one of the key things that we have to do is identify this vertex W. What is this vertex W? Well, it seems like what we need to do is figure out what is the first point at which we can get to a branching vertex in some sense starting from the vertex v. So, take a look at the vertex v if it is already a vertex whose degree is two or more then you are done. So, W, S just equal to v and the distance term can be thought of a 0 and we will come back and talk about how to split the terminals into two parts in this case.

It is actually going to be the same as what you would do if you have a vertex W distinct from v so we will just handle all of that together, but on the other hand that v had degree 1. So it has just one neighbor then you could just walk up to that neighbor and check if that neighbor happens to be a branching vertex that is in the graph that we are considering right now in this sub graph if you like does the neighbor of v have degree at least 3 or not.

If it is a degree 2 vertex it had a neighbor v and it just has one other neighbor then we continue our walk and just see if you know maybe we get lucky with the next vertex and so on. Is it possible that this just ends up in a situation where you never find a branching vertex which is to say that your tree just happens to be a path? Well, in that case you must basically end the terminal vertex if your tree was a path.

Remember that v by choice is a Steiner vertex it is not a terminal vertex and we have assumed that all terminal vertices have degree one. So, none of the intermediate vertices on the path can be terminal vertices either. So, your whole structure would have just one terminal vertex. So, you might be tempted to say something like oh that contradicts the preprocessing that we did.

Remember in the preprocessing steps we said that you have more than one terminal vertex because that is when this problem is interesting, but remember we are working with sub problems here. So, the original instance must have more than one terminal vertex, but when you are considering sub sets of terminals you may have to consider single terminal subsets so that is entirely possible.

But actually that is going to be our base case when the sizes of the terminal subsets are just one than that is essentially a shortest path computation. So, when size of these one and you are looking for the best way to trivially connect this set of terminals there is nothing to do as far as connecting the terminals is concerned, but you want to do it in a way that involves this specific vertex v that you have identified than that basically amounts to the computation of the shortest path between v and the sole terminal vertex.

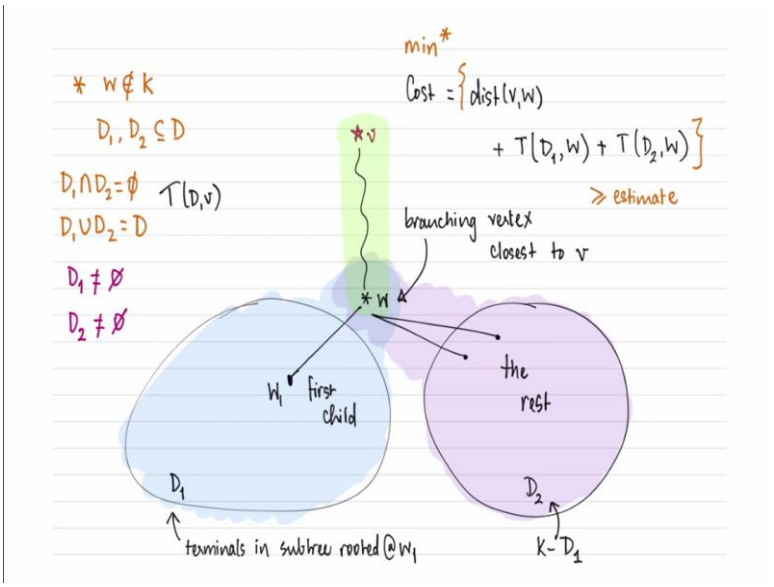
So, I had not explicitly mentioned the base case earlier and I am doing it now in the middle of this discussion somehow. So just keep this in mind, but when the size of these one then we handle it separately so we can assume that D has at least two terminal vertices, but now if D has at least two terminal vertices then this picture right here cannot be a path because if it is a path we just said it starts with v it has to end the terminal vertex for it to account for even one terminal vertex, but none of the intermediate ones can be terminal vertices.

So, clearly a path is not going to be robust enough to connect two terminals for the vertex v two or more terminals. Therefore, we know that this tree must branch somewhere so when you start this process of exploring starting from v at some point you will encounter a branching vertex. The first time that you do let us call that vertex W. If you want to be little more formal about how you identify W here is another way of looking at it.

Go and look at this tree and identify all the vertices that are branching when you route the tree at the vertex v. Among all the branching vertices identify the one that is closest to the vertex v and this is the vertex that we want to call W. So, remember that there must be at least one branching vertex because if not then you are dealing with the situation with just one terminal which we have taken care off separately.

So, that is what we have so we have a choice of W and now from W we want to get to identifying D 1 and D 2 and as I said this process of identifying D 1 and D 2 will be the same irrespective of whether W is equal to v or whether W is distinct from v. So, let us think about what would be a natural way of getting to a partition of D into two subsets.

(Refer Slide Time: 38:17)



So, for that let me just try and redraw this picture a little bit here. Once again we have v and W and we identify W 1 as the first child of W and then we have the rest. So, we know that W has at least one other child because we just defined W to be the branching vertex that was closest to v so it has at least two children and W 1 is the first child and then all of the other children are clubbed together and in a group that I am just calling the rest.

So, now let us look at all the descends of W 1 and all the descendents of the rest and I want to talk about D 1 and D 2 as being the set of terminals that we find among the descendents of W 1 and among the descends of the rest respectively. Now, observe that both D 1 and D 2 must be in non-empty subsets of terminals. The reason for this is that what we are looking at here is a min cost solution to the problem that we were poised to begin with which is the best way of connecting D using the vertex v.

So, if for instance, D 1 was empty then what is this whole sub tree routed at W 1 doing. We could just get rid of this blow upon the left hand side and still be left with a Steiner tree that beautifully connects all the terminals we did not miss anything and the vertical v and this would be cheaper because we know that all the edge weights are positive. So, certainly D 1 is non-empty and for completely symmetric reason D 2 is non-empty as well.

So, now try and understand the quality of T of D, v in terms of these 3 component pieces that we have identified by just writing out the cost and adding them up. Notice that these three pieces here are edge disjoint and in particular let me color code (()) (40:19) to see and we just need to add up the cost of the edges in each of these 3 regions which essentially partition the sets of all the edges in T of D, v.

So, if you look at the green chunk up there that is essentially the cost of shortest path from v to W so that is the cost of the green piece and if you look at the blue chunk then that is essentially a cost that is given by T of D 1, W because notice that T of D 1, W captures the cost of minimum weight Steiner tree that connects D 1 and has the vertex W. So, certainly the cost of all the edges in the blue chunk is at least this quantity because T of D 1, W is the best that you can do by definition.

But on the other hand it is exactly this quantity because if it was worse if it was more expensive then we could just cut and paste this. We could remove the blue chunk and replace with whatever is the best way to connect D 1 with the vertex W in case that happens to be cheaper and then we would get something that is cheaper overall contradicting the fact that we have something that is witnessing T of D, v which is the best way supposedly of connecting all the terminals D while using the vertex v.

So, hopefully that made sense it is a standard cut and paste argument and also applies to the purple chunk of the right. So, the cost of all the edges in the purple chunk is for very similar reasons given by T of D 2, W again. So, putting everything that we said so far together we see that the cost of the structure that we have here which is T of D, v is really given by this sum of these 3 quantities that we discussed.

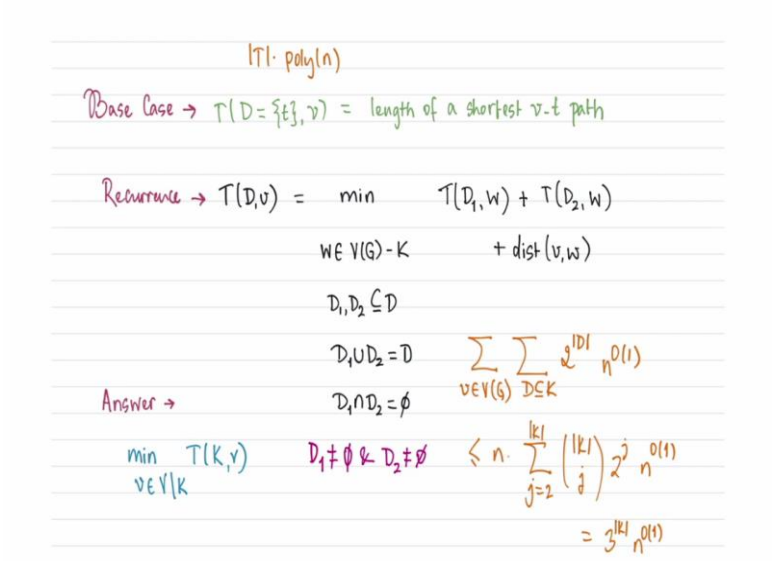
And now I claim that at least the original estimate because if you think about what the estimate was it was really a min taken over this expression and to be more specific this min is taken over all possible choices of W as long as we are looking at non-terminal vertices and it is over all non trivial partitions of the terminal set D. So, in particular we want D 1 and D 2 to be disjoint and we also want the union to be the entire set of terminals that are under consideration right now which is D.

And although I did not emphasize this much when we were coming up with the recurrence let me point out that this min is taken over non-trivial partitions of the terminals in the sense that we do want both D 1 and D 2 to be non-empty and the reason for that is remember this recurrence really gets implemented as DP algorithm. So, you cannot be self referential you do want to break up your problem into non-trivial sub problems so that you can rely on things that are already pre computed.

So, if one of D 1 or D 2 turned out to be trivial like say the empty set and D 2 was all of D then this problem this recurrence would end up being circular and that would be quite problematic. This is why when we were talking about the proof in this direction we did emphasize that the way we have broken down our structure we do get a non-trivial partition of D.

We said that both D 1 and D 2 are non-empty here and I just want to emphasize once again that was actually an important thing to show. So, with that we come to the end of our discussion of this recurrence.

(Refer Slide Time: 44:10)



And let me just summarize the overall algorithm. So, remember we are trying to track the best way to connect a subset of terminals using a specified vertex v and if your set of terminals has size 1 you considering single ton subsets of terminals then this just amounts to finding the shortest path between the specified vertex v and the (()) (44:36) terminal that you are working with.

So, that is just a polynomial times shortest path computation and this is basically going to constitute the base case of your DP table. Now after this we come to the generic recurrence and what we have here is what we just discussed and what I would like to point out again is that we only work with non-trivial partition so that the recurrence here does not get circular and notice that the two expressions that you have that our table look up are straightforward to do.

And the third term is again the distance computation that is just a standard application of a shortest path algorithm once again. So, that is essentially the overall algorithm when you are ready to look up the answer essentially you just figure out what is the value of T of K, v for already this is v which are not terminals. So, you just loop over all the Steiner vertices and look at the values of T of K, v and pick up the best that is going to be your final answer.

So, now the only thing that really remains to be discussed is the running time of this algorithm and so to do that let us just continuing studying what is going on the base cases and the main recurrence. So, the base case is relatively easy to talk about. There are size of K many entries to be computed here and each of them is a polynomial time computation. On the

other hand for the recurrence you could basically do a quick back of envelope calculation to say that the number of entries is due to the size of K times n.

And for each of these entries where it is essentially two table looks up and a polynomials times computation, but this min is going over all possible ways of portioning the set D into two parts. The size of the set D is upper bounded by the size of K you could say that this is never going to exceed 2 to the size of K and for each such choice of a split or a partition we know that is only a polynomial amount of work.

So, for each table entry we never need more than 2 to the size of K times poly n. So, you could combine all of this to get bound that looks like 4 to the size of K for instance, but you could do a more careful analysis and you might have seen this form of analysis in for instance when you were dealing with iterative compression. So, let us just think of this a little more holistically.

So, the running time is given by this double sum here. So, the double sum naturally captures what is going on with the two indices of our DP table. The first one ranging over all Steiner vertices the second one ranging over all subsets of the terminal set and this is the time that you need to compute each such entry it is exponential in the size of the terminal set that is currently under consideration.

So, the outer summation of course contributes a factor of N and the inner summation can be rewritten in terms of the sets just grouping them according to their size so that you get this standard binomial expression here which works out to 3 to the size of K with a polynomial overhead and that was the target that we set out for ourselves when we started. So, at this point we could say that we are done.

So, that is pretty much all that I had to say about this particular approach to solving the Steiner tree problem. This can be improved in terms of running time, but I think this is a very clean example of how a technique like dynamic programming over subsets can be really useful in coming up with FPT algorithms. So, that is a wrap on the DP based examples for this week and now we are going to switch gears and talk about integer linear programming as again a very broadly applicable technique for coming up with FPT algorithms and that is the subject of the next module. So, I will see you there.