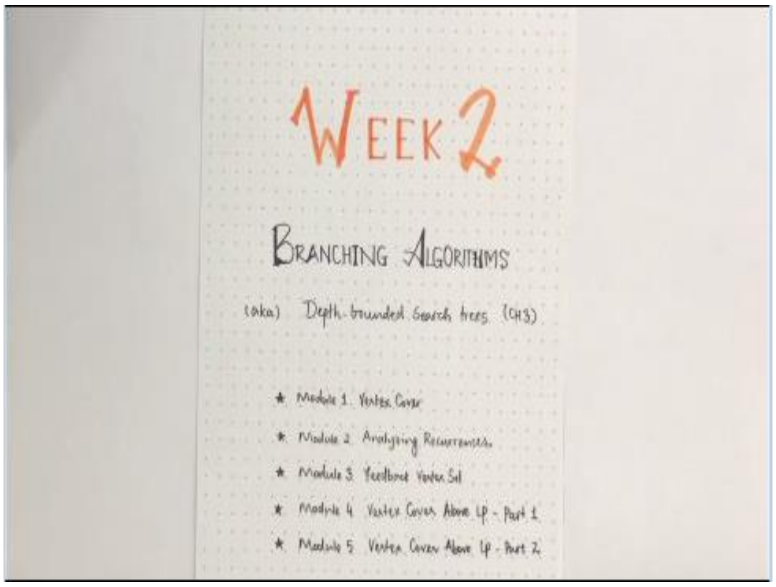
Parameterized Algorithms

Neeldhara Misra and Saket Saurabh
The Institute of Mathematical Sciences
Indian Institute of Technology – Gandhinagar

Lecture - 09
High-Degree Branching for FVS

(Refer Slide Time: 00:11)



WEEK 2

BRANCHING ALGORITHMS

(aka) Depth-bounded Search trees (CH3)

* ★ Module 1 Vertex Cover
* ★ Module 2 Analyzing Recurrences.
* ★ Module 3 Feedback Vertex Set
* ★ Module 4 Vertex Cover Above LP - Part 1
* ★ Module 5 Vertex Cover Above LP - Part 2

So, welcome to the third module of the second week. **(Video Starts: 00:15)** In this discussion, we will be exploring a problem called feedback vertex set. And as what vertex cover, the goal will be to come up with a branching algorithm for the problem. So, since this is the first time that a meeting feedback vertex set, at least in this course, let us begin with some introductions.

So, a feedback vertex set is simply a subset of vertices in a graph whose removal leads to the destruction of all cycles. In other words, when you delete FVS, the graph that you are left with is essentially a forest. And the natural computational question associated with this definition is the task of finding feedback vertex set of the smallest possible size. So, here is the question stated as a decision problem.

So, given a budget k, can you find FVS of size at most k. Now, FVS turns out to be one of the earliest problems that was shown to be NP complete. It is got a long and rich history. It also has numerous applications and diverse areas ranging from VLSI chip design to operating systems and so on. So, if you are curious about the problem, definitely check out the Wikipedia page for more history.

Just to give you a since, for example in operating systems, you might think of modelling dependencies between tasks as a graph. So, it is quite natural, your tasks are vertices and you have an edge from a task A to task B if A depends on B’s completion to make progress. Now, this is a graph where you clearly do not like cycles, because cycles naturally correspond to deadlock situations.

And if you do have these deadlocks, you want to resolve them by probably killing some of these tasks. And since that is an unpleasant thing to do, you want to kill as few tasks as possible to resolve all the deadlocks. And that is exactly the feedback vertex set problem. Although applied to a directed graph, so, the problem is defined in pretty much the same way for both directed and undirected graphs. But in this discussion, we will focus on the undirected case.

Also, in most of our discussions, when we talk about undirected graphs, we assume that they are simple in that that are not multiple edges between vertices that are no self loops and so on. But this is one of those discussions where we are going to drop that assumption. So, our graph need not be simple. We will allow for this more general situation where there might be multiple edges between vertices and there might be self loops and all of that adjust is going to be allowed. So, that is fair again.

Now, let us just to get feel for this. Let us continue by looking at an example. So, here is a graph. And I want you to think about what is the smallest FVS in here. And to guide your thought process, let us think about these 2 very related questions. The first question is, how many vertex disjoint cycles can you find? And the follow up question is, what does that tell you about the size of the FVS? The smallest FVS, any FVS, just think about that.

So, this would be a good time to pause if you want to think about this yourself for some time. And assuming that you have thought about it, let us get to answering the first question. So, it turns out that this graph that we drew before, has 3 vertex disjoint cycles. You could ask if that is the largest number of vertex disjoint cycles that there are, maybe if I drew them differently, you could have found 4 or more. And we will get to that in a moment.

So, there are 3 vertex disjoint cycles. And it turns out that there is also a feedback vertex set of size 3. And let us just make sure that I am not cheating here by showing you the original graph as well. So, if you were to pick these 3 vertices, then you actually get spheres. So, that also means that there could not have been 4 vertex disjoint cycles. Because notice that if you have in general k vertex disjoint cycles, then any feedback vertex set must have size at least k.

Because every cycle is going to demand separate position in the feedback vertex set. You cannot use a single vertex to kill more than one of these cycles from the collection of vertex disjoint cycles. So, that brings us to the natural question w is the connection between the largest number of vertex disjoint cycles and the size of a small this feedback vertex set?

Now, you can probably conclude from what we just said that the size of a smallest feedback vertex set at least the number of vertex disjointed cycles. But the question is, can you bound the size of the FVS as some function of the number of vertex disjoint cycles? And let you think about that you can probably see a parallel between the size of a smallest vertex cover and the size of the largest matching.

And for vertex cover, you have perhaps realised already that the size of smallest vertex covers at most twice the size of a largest matching. So, this is a question in that spirit. What is, can you make some connection or not? And I am not going to get into that here. Because, it is probably not going to be as relevant to the algorithm that we will develop, but it is a nice question to think about.

So, you can think about it when you have a chance. So, with that said, let us get back to the main gender, which was to develop a branching algorithm for FVS. So, let us begin by asking ourselves, what is the natural branching strategy for the problem? So, do we try some of the things that we tried for vertex cover? Or should we just try something different this time? So, notice that for FVS, the main forbidden object is a cycle.

So, when you see a cycle, you know that any FVS must include at least one of the vertices from that cycle; maybe this leads to some short of natural branching strategy. So, please take a moment here to think about what would be your first short at a branching algorithm for

FVS. And come back to this video, once you have had a chance to think about it. So, you have probably concluded that branching on cycles is the right thing to do. It is certainly exhaustive.

And at every stage of the branching your measure, which again could be thought of as k drops by 1 since, you are including one vertex in your solution at every step. So, it certainly seems like a perfectly valid depth bounded search tree. But recall that the running time of your overall branching algorithm is governed by both the depth and the breadth of your search tree.

And when you are branching on cycles, you do have to be careful about how many parallel universes you are proliferating at every stage of your algorithm. Notice that the number of instances that you generate is going to be proportional to the length of the cycles that you branch on. So, you could be lucky. And maybe you always find cycles that are short; maybe their lengths are bounded by even a constant that would be a very good situation.

But if you think about it for some time, you can easily come up with graphs, which are not so nice. And maybe all the cycles you encounter, do not have their lengths bounded by any function of k, their lengths could be for instance n over a constant. In that case, the branching algorithm is forced to create too many instances and it becomes a very unwieldy search tree. Now, I will say that this idea can in fact, be pushed a little bit further with some additional observations.

And you can actually make it work and eventually get to FPT running time. But we will be taking a slightly different approach in this lecture. So, I am going to leave this as a puzzle. Although, I will say that some of the things that we are going to do will probably be relevant to you as you solve this puzzle as well. So, do think about that but let us just take a small the tour here and come up with some ways of simplifying the instances of FVS that we are working with.

Now, what we are going to do now might remind you quite a bit of what you have seen last week for kernelization, but let me just emphasise that we are not going to be developing a kernel. But nevertheless, these reduction rules will help us get to a point where we can do a certain kind of branching and that will become clear as we go along. So, what I am going to do now is present a few scenarios.

And if you like, you could just pause the video right after we describe the scenario and before we describe what we are going to do in that situation. So, here is the first situation, what if you see a vertex with a self loop on it? Well, what this means is that we have a cycle of length 1 and this cycle must be destroyed and there is only one vertex that can destroy it.

So, what we are going to do is we are going to remove this vertex from the graph and reduce k by 1, remember to reduce k by 1 to account for the fact that v is going to be a part of our solution. Now, the second situation is what if you have multiple edges between a pair of vertices and in particular, you have more than 2 edges between a pair of vertices. Let me say here that if you have a pair of edges between 2 vertices that is going to mean that there is a cycle of length 2 in this graph.

But if there are 3 edges between 2 vertices, it still means that there is a cycle of length 2 in this graph between these 2 vertices, it does not really mean anything more. So, it would be quite legitimate to reduce the multiplicity of this edge to 2, because that just carries the same amount of information. Now, remember that the equivalence of these 2 instances needs to be established formally, but I am going to skip that in the interest of time. But it is a couple of sentences in both directions.

So, please do write that out to convince yourself that nothing has been damaged here. Remember that k remains the same here. There is no need to change the budget. What if you have a vertex whose degree is 1, remember that for vertex cover. For instance, if you see a vertex of degree 1, you might want to immediately choose its neighbour. But of course, we are now talking about feedback vertex set.

So, what we are going to do is slightly different. For the reason that degree 1 vertices never participate in cycles, we can get rid of these vertices without changing k. So, remember that we will never participate in an optimal or a minimal FVS, because any FVS that contains a degree 1 vertex is also FVS without this degree 1 vertex. So, there is really no reason to leave this vertex in this graph, you might as well deleted.

Next step, what do you do if you see a vertex of degree 2? Well, this is going to be a little less straightforward than vertices of degree 1. But nonetheless, there is something that you can do. But I will say that of all the scenarios I have presented so far, this is the one that requires the most thought. So, please feel free to take that pause and think about what would you do in this situation?

So, you have probably realised that every cycle that passes through a degree 2 vertex; also passes through both of its neighbours. It is almost as if these 2 neighbours are guards, ensuring that they witness every cycle that goes through the vertex u. So, in function, whatever you can do; either v or w can do it as well. So, it seems like this is a dispensable vertex. So, what if we just dispose of it?

Now, that might be slightly problematic, because if you are directly delete a degree to vertex, you might actually killed cycles that it participates in. And in the new graph, although v and w would have been perfectly capable of killing these cycles, they simply do not know that the cycles are there. So, we need to do something to preserve the memory of the cycles that you did participate in. And to do that well, one thing that you can do is add an edge between the vertices v and w.

And remember that you want to add this edge on top of potentially an edge that already existed before. Think about why that is the case. And also do try to formalise the argument for why these 2 instances are in fact equivalent. In this case, this will take little bit of work, but it is not difficult at all. So, please do that and convince yourself that this idea works. This process is sometimes called short circuiting.

Short circuiting the vertex, you are kind of bypassing you and you are letting its neighbours do the work for you instead. So, that is going to be our last reduction rule here. And let us just summarise what we have achieved with the application of these reduction rules. So, just to summarise, we had 4 reduction rules where we took care of vertices with self-loops, edges of high multiplicity and vertices of degree 1 and degree 2.

So, if you have a graph where none of these reduction rules apply, then you are in a situation where the graph has no loops. It has only single and double edges and it has minimum degree at least 3. So, that is the short of a graph that we might as well be working with because if this is not the case, then you can bring in one of these reduction rules into the same. So, now, the natural question in your mind must be how does all this help with branching.

So, to address this, let us finally talk about the branch same strategy. I mean, wisdom want to keep these properties in the back of our mind. And I promise we will soon see why they are useful. But what is the branching strategy going to be? Well, we have already said that we are not going to be branching on cycles. We are going to be trying something different. And that turns out to be high degree vertices. We want to somehow involve high degree vertices in our branching strategy.

Now, this is reasonably intuitive, because high degree vertices seem to get a lot of work done and it seems tempting to include them in our solution. But unlike vertex cover, where we could confidently say that if you have a vertex cover of size at most k, then it must include any vertex whose degree is greater than k. Feedback vertex sets do not have this obligation. And you can verify this by coming up with an example yourself.

But, for how long can feedback vertex sets avoid high degree vertices. Try coming up with an example where a small FVS avoids all the high degree vertices in an instance. And we want to see that for some choice of quantifications. This statement can actually be formalised before we get into the specifics. Third, let us try to get some high level intuition by staring at a picture.

So, here what is depicted is a small FVS that leaves behind this messy looking forest. What would happen? If we dulled the FVS that look, you are not allowed to pick up any of the high degree vertices in this graph. What would happen then? Well, all the high degree vertices would then be sitting in the forest. And they would lead to the existence of very many leaves. And notice that our graph is reduced with respect to these reduction rules from before.

And crucially, the minimum degree is at least 3, which means that all of these leaves actually have at least 2 more neighbours in the graph. But, since those neighbours do not belong to the forest, they must be in the feedback vertex set. Similarly, if you have degree 2 vertices in the forest, then they must have at least one other neighbour which must again belong to the FVS. What this means is that the more high degree vertices in the forest, the more edges that we have that how one of their endpoints in the forest and the other in the FVS.

So, in some sense, the forest is throwing a lot of edges at FVS. But now, the FVS is a small set of vertices, whose degree is now bounded in some way, because we said that this FVS is not allowed to pick up any of the high degree vertices. As a result, the capacity of the FVS to absorb these edges that are being thrown at it is also limited. And it should lead us to some short of a contradiction. So, that is the plan.

And this is at least the overall intuition for why we expect that at least some of the high degree vertices must actually belong to the FVS. Technically, at least one of the high degree vertices should belong to the FVS. Now, the question is: what is the quantification? How far do we need to go in the line-up of high degree vertices to be sure that at least something gets picked up by the FVS? For the purposes of branching, we would really like this threshold to be some function of k. And as it turns out, this is indeed the case.

So, let us now take a look at the main technical lemma that will drive the correctness of the branching algorithm. So, let us begin by ordering the vertices in descending order of degree, which is to say that the high degree vertices come first and so on. And let us also remember that our graph is reduced with respect to the reduction rules that we described earlier then the claim is the following.

So, if you have a feedback vertex set of size at most k in such a reduced graph, then this feedback vertex set must contain at least 1 vertex from the top 3k vertices in the list of vertices that have been shorted by their degree. Now, if this lemma were true, then hopefully you can see that the branching algorithm is very, very natural. What you can do is short the vertices according to their degree and you could just branch on the top 3k vertices.

And thanks to this lemma, this branching strategy would be completely exhaustive. And notice that because you are picking 1 vertex in your solution in every branch, as before, the depth is naturally bounded by k, but more nontrivially. The width of your branching, the number of instances that you generate at every stage is bounded by 3k, thanks to this lemma. If you were going to go and implement this algorithm, make sure that you are careful to ensure that G is always reduced.

In particular, this means that these reduction rules have to also be applied recursively. So, it might happen that when you pick a vertex to be included in your prospective solution. After you remove this vertex from the graph, you generate vertices there are degree 1, degree 2 and so on. So, make sure that these are cleaned up according to our reduction rules from before, before you tell into branching again, because the correctness of the branching really crucially relies on the fact that the graph is reduced.

So, I leave it to you as an exercise to work through the details of the complete branching algorithm defining the measure and formally arguing that the running time of the algorithm is 3k to the k with some polynomial overhead and that everything checks out. So, for the rest of this discussion, our focus will be completely on proving that this lemma is true.

So, we are going to do that by first proving an intermediate claim and then trying to put a few inequalities together. The whole thing is really driven by the intuition that we described before. But actually prove this claim with this specific threshold; we still need to do some calculations. So, let us get started. First, we will show a small intermediate claim, which is effectively a lower bound on the sum of the degrees of all the vertices in a feedback vertex set.

Well, I mean, the expression here is slightly more awkward than that it is the sum of numbers that are one less than the degree. And of course, we could have just rewritten that and pulled out a k on the other side. And that would have just been a little more convenient. But it turns out that this is the form that is useful for the computations that are coming up in a moment. So, we will live with this.

More importantly, let us just try and take a look at what this inequality is telling us visually? So, that we at least have some intuition for the direction of the inequality. And we know that it makes sense. It is what we should expect. So, here is the FVS of size k. And here is the rest of the graph. Now, the rest of the graph being a forest on n – k vertices can only accommodate n – k – 1 edges.

Notice that every other edge, that does not belong to the forest, is incident on x and can be proxied in some sense by the sum of the degrees of all the vertices in x. So, if we were to write down the following expressions, so let us say we have n – k – 1. And we also have the sum of the degrees of all the vertices in the feedback vertex set. Then notice that every edge here is getting counted at least once the edges in the forest get counted exactly once here.

The edges that have one endpoint in the forest and one in the FVS get counted exactly once when we sum over the degrees of all the vertices in x and every edge that has both of its endpoints inside x, well, they get counted twice. Let us say, you have an edge uv with both of its endpoints, u and v belonging to the FVS. Then when we think of degree of u, that edge gets counted once and when we think of the degree of v showing up in the summation, then that edge gets counted again.

So, essentially, every edge contributes at least one to the sum. But the sum, of course, is going to be potentially a little more than the number of edges in the graph, especially if the FVS is not empty in the sense that it has. It has some edges inside it. And so, certainly, this is lower bound for the total number of edges in the graph G. And now, with some appropriate rearrangement, so, we just pull in the minus k here.

So, what do you have is remember that the size of x is k. So, this really amounts to d of v – 1, which is promising that is kind of what we have on the left hand side there. And just bringing the remaining terms on the other side, we have exactly what we want. So, let us go back to the statement that we originally wanted to prove. Recall that we are working with a reduced graph and we are also organised the vertices in descending order of degree.

So, the high degree vertices come first. And in this setting, the claim was that any feedback vertex set of size at most k must choose one of the top 3 k vertices. And we will prove this by way of contradiction, which is to say that we will begin by assuming that the statement is false. And we will explore the implications of that and hopefully run into some problems, which will amount to approve of the statement itself.

So, let us begin by saying suppose not, suppose this was not true, then what is the scene? So, we have the vertices as before. And we have a feedback vertex set of size k that has somehow managed to evade the top 3k vertices completely. So, the intersection of x and v 3k is empty. And we are going to explore what happens if this is the situation. But let us just pause and think about what is it that we hope to contradict. There must be some short of a concrete assumption that breaks down as we explore the implications of the suppose not.

So, in terms of assumptions, we have that we are working with the reduced graph. And in particular that implies that the minimum degree is at least 3. So, what does this say about say the number of edges in the graph. You can imagine that if the minimum degree is high, then you are going to have lots of edges. So, let us just try to quantify that a little bit here. So let us begin by talking about the sum of the degrees of all the vertices in the graph.

We know that that is at least 3 times m from the minimum degree assumption. But we also know that the sum of the degrees of all the vertices in the graph is just twice the number of edges. So, we have that 2m is at least 3n. Or in other words, m is at least 3 by 2 n. So, this is an assumption that is fairly concrete one that we can perhaps hope to contradict. So, let us try. Let us start with this expression, which involves the sum of the degrees.

And let us just break that down into 2 parts. First, the top 3 k vertices and then the remaining vertices. Let us also recall the previous claim that we showed which had something to do with the sum of degrees, but in the context of the vertices of the FVS. So, let me begin by writing down the sum of dv – 1 for v and v 3k. So, recall that v 3k simply denotes the set of the top 3k vertices and that vertex order.

Now, I am going to say that this is at least 3 times as similar expression but taken over the feedback vertex set. Now, why is that? Well, notice that the feedback vertex set is really trailing the vertices of v 3k. All of the feedback vertex set sets beyond v 3k, little v 3k, in this case and so, you could break down this length of 3k vertices into 3 chance of k vertices each. And you could compare them vertex by vertex with the vertices of the feedback vertex set, which also is a set of size k.

And with that comparison, you are just going to reach this inequality. And the reason I wanted to write it in this form was to be able to apply this claim from before so, we have this is at least 3 times m – n + 1. All right, so, that is our first inequality. Let us consider the remaining vertices now, so, v n v – v 3k. And now, I am going to be able to say that this is at least v n x d of v – 1 for an even simpler reason than before. And this is because really x is a subset of v – v 3k.

So, literally, every term here is also here and maybe some more terms here. And since every term is non-negative, this works out fine. So, applying the same claim as we did before, we have that this is at least m – n + 1 as well. So, we have these 2 inequalities here. Let us combine them to obtain a third one, which is going to be about a sum over all the vertices and the expression is still going to be d of v – 1.

And now, we see that this is at least 4 times m – n + 1. So, let us break this down a little bit more. We have the sum of the degrees over all vertices again being 2 times m – n and that is going to be at least 4 times m – 4 times n + 4. Just rearranging the terms here, so, that it is easier to see the contradiction that we were hoping for. So, we have that 2 m is less than or equal to 3 n – 4.

And in particular, this implies that 2 m is less than 3 n. And that is problematic with respect to our assumption. So, that brings us to the contradiction that we were after. And this means that we are pretty much done. And with the proof of the main lemma, we now have the branching algorithm that we wanted. And that brings us to the end of this particular discussion **(Video End: 30:55)**. Thanks for watching and we will see you in the next module.