Parameterized Algorithms
Prof. Neeldhara Misra, Saket Saurabh
Department of Computer Science and Engineering
Indian Institute of Technology-Gandhinagar

Lecture-20
Derandomization

Welcome to the final module of the 4th week in parameterized algorithms. So, this is going to be a very brief discussion about derandomization. Given that all week we have been talking about randomized methods the title derandomization can feel a bit off topic. But really this is just to satisfy any curiosity that you might have about whether the randomized algorithms that we have discussed have deterministic variants.

And in particular is there a process by which the random experiment can be replaced with some sort of a deterministic procedure that does the same job. Now it turns out that there is no magical way of doing this automatically and in particular doing it in a way that gives you some sort of a promise of efficiency, if not the same efficiency at least something that is close, that is not clear.

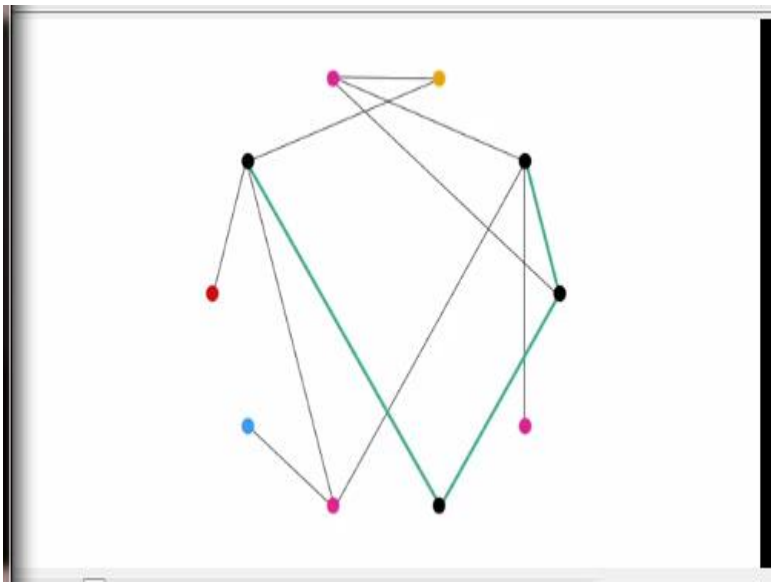
But many randomized algorithms do lend themselves to natural derandomization procedures. And I will just use colour coding as a running example to introduce you to the kind of methods that exist to perform the randomization. So, I will just be introducing this one combinatorial object called the hash family and we will see how that can be used to derandomize.

We will not really get into the construction of these hash families and in the interest of time I will also not be introducing the other analogous combinatorial objects that can be used for some of the other methods that we have seen, like chromatic coding or random separation. So, this is just going to be a very focused and a very brief discussion, if you really interested in more then you could look up the derandomization section in the textbook and you will find a few more details there.

But even the textbook presentation does not actually get into the construction of most of these combinatorial objects that you are going to need to perform the derandomization. So, if you are

really interested in those, there are some pointers on the course website, so you can check them out. But for now let us just get an idea of how this business of derandomization works.

(Refer Slide Time: 02:28)



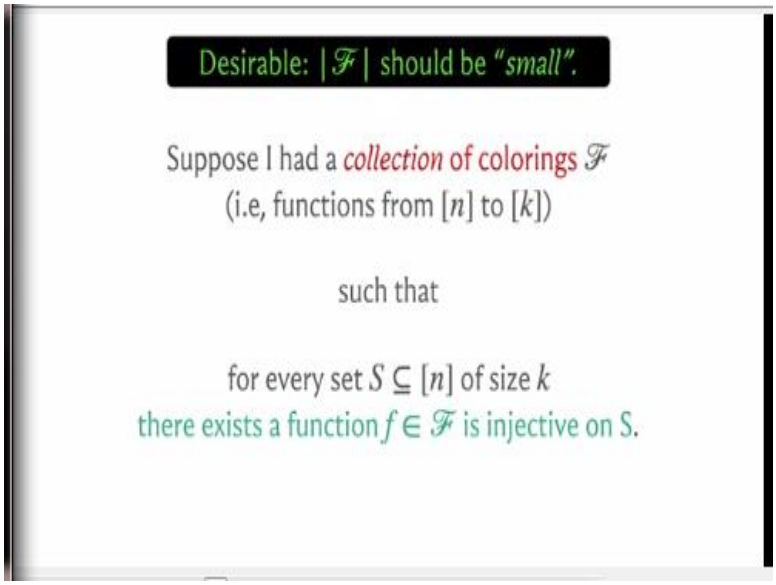
So, let us go back to the good event in the k path problem where the random experiment that was employed by the colour coding algorithm involved colouring every vertex of the graph, uniformly at random with one of k colours. And of course these experiments were independent across all vertices. So, notice that the good event for us was that if we focused our attention on the k vertices that are involved in some path, we wanted these vertices to get all distinct colours.

So, how do we replace this random engine with something deterministic? Well, one natural thing to try is to simply loop over all possible colourings, that would account for every possible eventuality of this random experiment and it would do the job. But notice that the problem with this is that it is way too expensive, so we would definitely want to avoid this. But at the same time instead of working with a one shot random colouring or even with multiple random colourings.

We want to work with a carefully chosen collection of colourings which has 2 key properties; one of course is that the collection should not be too large; it should be a manageably sized collection. And the second thing is that, well, we do not know where the path is going to be; it could be any subset of k vertices. Therefore what we want is that for every subset of k vertices

there must be some colouring in our collection that takes ownership of that subset. It says, look if this was the subset which form the path then I am a colouring which will colour these vertices with distinct colours.

(Refer Slide Time: 04:15)



So, let us write all that down, just to recap. So, what we want is some collection of colourings which we are going to formalize as functions from 1 to n to 1 to k, where k denotes the size of your palette, in this case it is also the size of the solution that you are looking for. And n is of course the number of vertices in the graph. So, you can think of 1 to n as representing the vertex set of G.

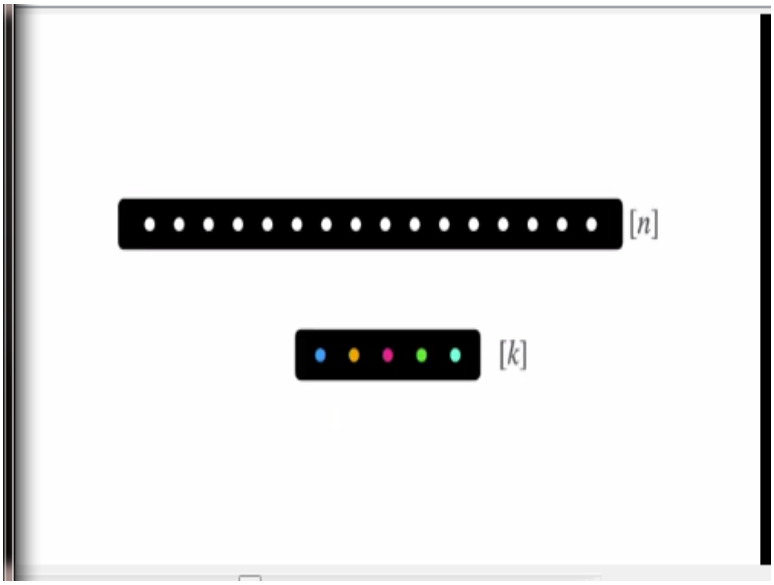
So, what you want is a collection of functions from 1 to n to 1 to k which has the property that for any subset of 1 to n of size k. You want that there is some function there could possibly be more than one function but there should be at least one which is such that it is injective, if you were to restrict it to these k vertices. Of course since you restricting to k vertices and you promised that it is injective and the range also has size k, it will in fact end up being bijective.

But the reason I have used the term injective is to really emphasize that colours do not repeat when you focus on this subset of k vertices. So, the crucial thing of course was the first thing that we mentioned which is that we want this collection of colourings to be small. Because you can

imagine that you can come up with such a collection quite easily by simply going through every subset of size k and just coming up with a function that is specialized to that subset.

So, you could say, look, I am going to colour this subset injectively and do something arbitrary on the rest. And this would be a perfectly valid collection but it would have size and choose k which is still not good enough for us. So, well, we will not get into the construction of efficient collections that have this property. But let me at least just give you some intuition for why you hope to do better than this very naive approach that I just described.

(Refer Slide Time: 06:14)



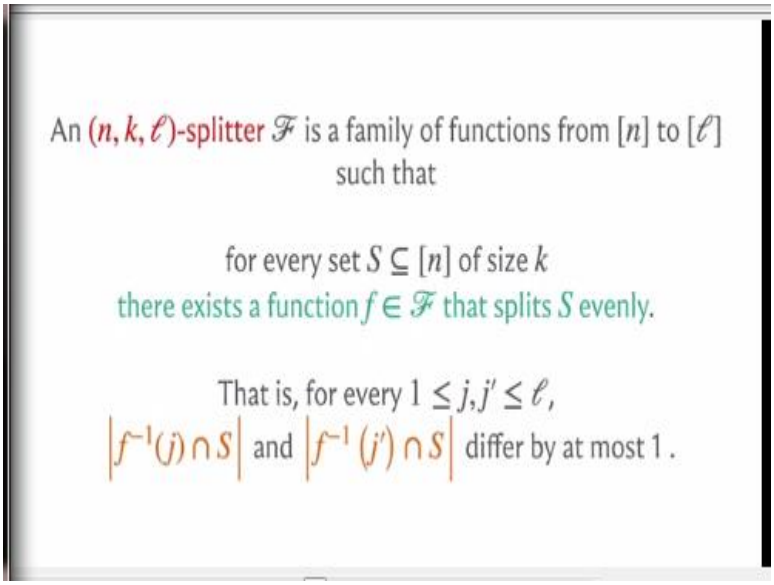
So, let us say that we have this abstract version of the domain and the range. The domain being 1 through n and the range being 1 through k. Now let us just look at some function from n to k denoted by just the way that these elements are coloured, that tells you which element they map to. So, if you look at just this one function, then notice that there are a lot of subsets that are mapped injectively.

So, you could pick any one of the blue elements and then pick any one of the yellow ones either one of the 2 pink ones and one of the 3 green ones and one of the three light blue ones. So, any combinations of these elements I think there are like some 288 of these, so that is 4 times 4 times 2 times 3 times 3. So, whatever that works out too, that many subsets that are coloured injectively by just this one function.

So, you hope that you can choose your functions cleverly, now I am sure you can come up with functions that also terrible for the goal that you have in mind. They do not really colour a substantial number of subsets injectively at all. So, for instance if there was a function that mapped all elements to the same element in the range, then this function is no good and you would not want to include it in your solutions.

So, you can carefully craft your collection of functions in such a way that it somehow does take care of, you want every function to do a lot of work for you. And that is kind of at least a very, very high level intuition for why you even hope for something like this to be possible. To actually craft such a collection, it turns out that you go vias a more general notion called splitters.

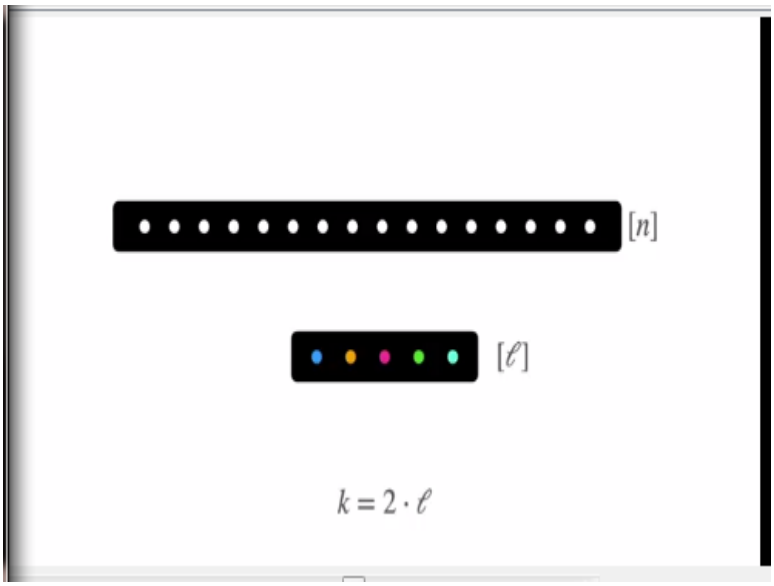
(Refer Slide Time: 08:00)



And it turns out that this is just a useful generalization to work with and it allows you to construct the kind of collections that we need by just working through it in h 2 steps in a certain way. Now I am not going to really elaborate on this here, but I am going to introduce the definition of a splitter, because I believe this is also something that comes up in the assignment, so let us just go through the definition.

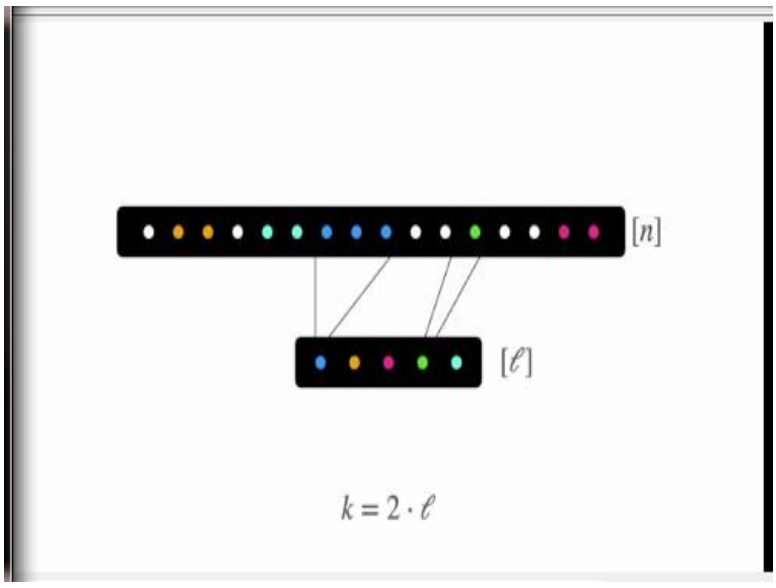
So, here we have a slightly more general situation where we are looking at functions from 1 to n to 1 to l. And what you want is subsets of size k to be split evenly. Now of course if l is at least k then this question of splitting evenly just ends up being the same as the function being injective.

(Refer Slide Time: 08:54)



But let us say that l is smaller than k then the question of splitting would be a little more general. So, for instance let us say that we have l being 5 as before and let us say k is 2 times l. So, what does it mean that a function splits a subset of size to time cell?

(Refer Slide Time: 09:12)



Let us focus on this subset, so the subset is all the coloured elements, I believe there are 10 of them here. And I want you to pause here and think about whether this function has been

projected on these 10 elements, does this function actually split this set? So, would you consider this to be a valid, a good function for this particular subset? Well, if you think about it and just look at the pre image of the blue element and the green element.

You will see that there are 3 elements that are mapped to the blue element and just one that is mapped to the green element. But notice that from the definition previously what we wanted was the pre image of any pair of elements from 1 to l to differ by the sizes of the pre images we wanted them to differ by at most 1. And therefore this particular example, this is not really a valid split.

(Refer Slide Time: 10:19)

So, it turns out that you can construct the kind of collections that we were looking for which by the way are called n, k perfect hash families. That is what you need to de-randomize the colour coding algorithm; they are really just n, k, k splitters if you think about it. And it turns out that one way to construct a n, k, k splitter is to compose a n, k, k square splitter with a k square k, k splitter.

So, notice that a n, k, k square splitter is a collection of functions that map 1 to n to 1 to k squared. And a k squared k, k splitter is a collection of functions that maps the domain 1 to k squared to 1 to k. So, if you were to compose these functions then you will end up with functions

that map 1 to n to 1 to k and it is just useful to do it in these 2 steps. Now I will not be getting into the details of how this composition is done, but I will state the result for you.

(Refer Slide Time: 11:17)

So, what you get? Once you go through this process is n, k perfect hash family of size e to the k k to the order log k times log n. Now if you notice what we had previously was a random experiment and the cost of boosting the success probability was e to the k. So, we wanted to repeat our experiment e to the k times to get our error probabilities down to a constant.

Now instead of doing that you kind of replace that whole process with just this one giant for loop over all the functions in your n, k perfect hash family. So, it is going to take you e to the k, k to the order log k n log n time upfront to construct this family. Once the family is available, then it is going to be the cost of the dynamic programming which if you remember was 2 to the k times n or so.

So, it is going to be that multiplied by the number of functions that you are working with and that is e to the k, k to the order log k times log n. So, it is really a k to the order log k times log n additional multiplicative expense to get rid of the randomization. And to have the peace of mind that you will always get the right answer if you used this combinatorial construct. And as I said there are analogous combinatorial objects that you can use to derandomize algorithms based on chromatic coding as well as random separation.

And I would really encourage you to look up section 5.6 in the textbook, if you are interested in learning about what those objects are? But with this very brief discussion I just wanted to give you an overview of the way in which you hope for some of these randomized algorithms h to be derandomized. So, I hope you enjoyed this little tour through randomized methods for coming up with FPT algorithms. And I think the overall theme has been that the algorithms themselves are typically really easy to explain and easy to implement they are elegant in their descriptions.

But they usually involve interesting and non-trivial analysis to show that they actually work out the way that you hope for them to work out at the end. So, that brings us to a wrap in terms of our explorations for week 4, I hope that we can continue this conversation over at the discord community as well as the mailing lists. Especially if you are watching this during an active run of the course, thanks very much for watching and we will see you next week.