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

Lecture-17
Colour Coding for Long Path

Welcome back to the second module of the 4th week in parameterized algorithms. So, in this week we have been exploring how we can leverage the power of randomness to come up with interesting FPT algorithms for a wide variety of problems and in this particular video we will be considering the long path problem or the longest path problem if you prefer the optimization version.

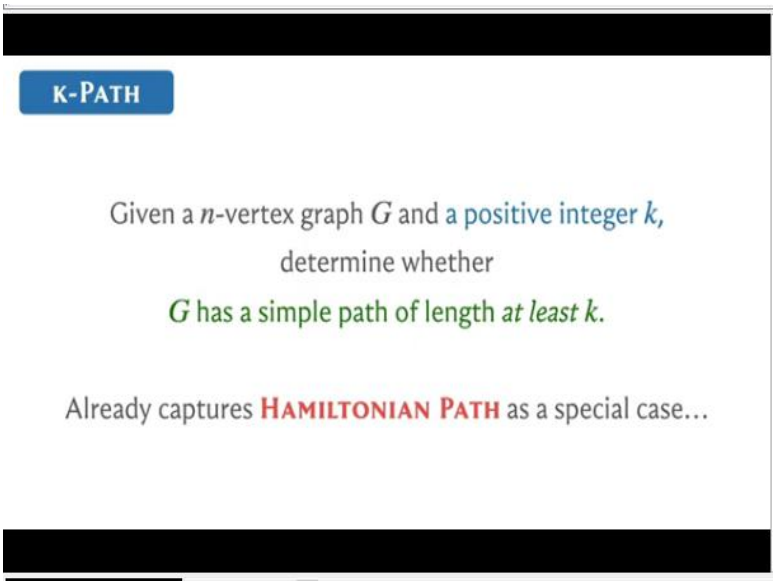
And in a previous undergraduate algorithms course you may have seen the short path problem or the problem of finding some shortest path between a pair of vertices, so this might remind you of that in some sense this is the exact opposite optimization objective. So, you really end up looking for parts that are as long and drawn out as they can be instead of looking for ones that are as short and quick as they can possibly be.

Now unlike the shortest path counterpart the long path problem does turn out to be NP complete which is why it is meaningful to look for FPT algorithms for them and we will be working with the standard parameter here which would be the size of the path which we think of as the solution and the parameter therefore will just be the solution size. Now the technique that we will introduce to solve this problem is called colour coding.

And it turns out that it is this really elegant and simple technique which is also extremely powerful and it has evolved to become a really popular technique both in theory and practice since it was first introduced in about 1995. So, with all that said let us get started with the definition of the problem and then we will talk about the overall random experiment and as usual we will try to understand what is our concept of a good event.

And then we will try to justify why the good event is actually good for us and then we will come back and analyze the probability that the good event actually occurs and with that we will be able to wrap up our analysis. So, let us begin by formally defining the long path problem.

(Refer Slide Time: 02:21)



So, here we are given as input a graph G and a positive integer k, where G we will assume as always is a simple and undirected graph on n-vertices. Although the problem definition carries over very naturally to directed graphs as well our focus in this discussion will be on our usual default setting which is that G is undirected. Now the question we are interested in is whether G has a simple path of length at least k and there are a couple of keywords there that are worth emphasizing and clarifying.

The first one is the notion of length, so for us the length of a path is just going to be the number of vertices on the path. So, we are really looking for a path on k vertices and k vertices or more and the other adjective here is simple. So, by a simple path we mean a path where vertices do not repeat. Now depending on which book you are reading some people would just not use the adjective simple.

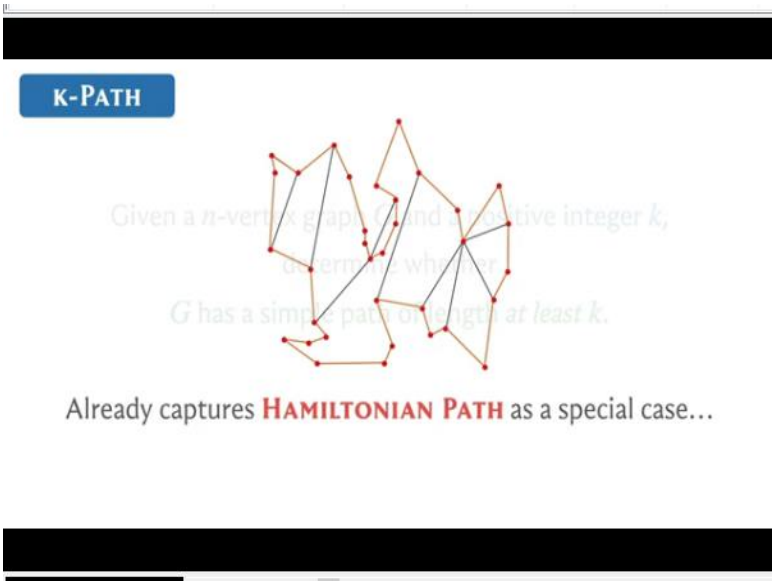
And would just use the word path to refer to a sequence of distinct vertices where every pair of consecutive vertices has an edge between them and will use the term walk to refer to the possibility that vertices can repeat or they do not all have to be distinct whereas in other sources a path would by default allow for repeated vertices and you use the adjective simple to emphasize that repetitions are not allowed.

So, in any case here we just want to be quite clear that we do not allow for repetitions. If you do allow for repetitions then the things that you are looking for let us call them walks are somehow much more tractable computationally and we will be revisiting this notion

sometime later in this course. So, I will not get into it very much here. Just keep in mind that what you are looking for is a sequence of k distinct vertices such that every pair of consecutive vertices has an edge between them.

Notice that it does not have to be an induced path so you could have other edges going around here, but at least these edges must be present. Now notice that k path generalizes the Hamiltonian path problem which essentially asks if there is a path that visits every vertex of the graph exactly once.

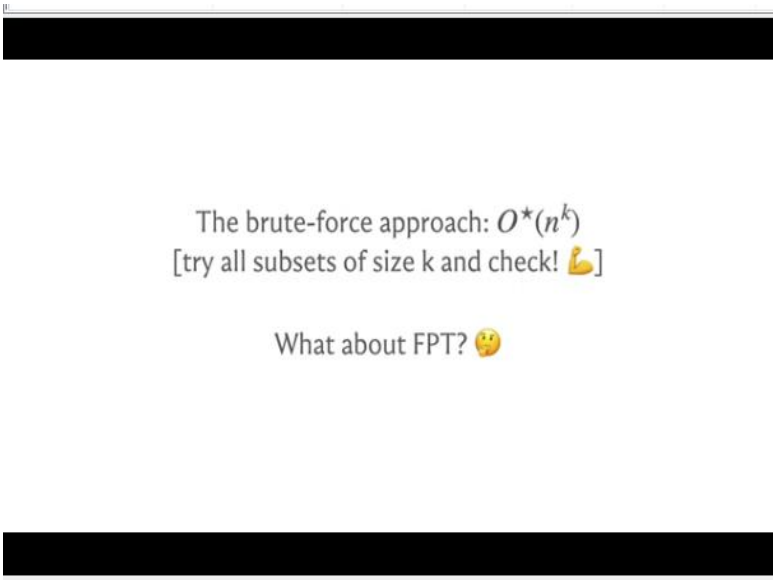
(Refer Slide Time: 04:50)



And that is essentially k path with k set to n. So, I am sorry that in this diagram what is highlighted is a Hamiltonian cycle and not a Hamiltonian path, but you get the picture. Hamiltonian path is a classically well-known NP -complete problem and because k-path generalizes Hamiltonian path, it is also NP -complete. So, from the parameterized perspective a natural parameter here is the standard parameter which is k the size of the solution that we are looking for.

And I would encourage you to pause here for a moment and think about the question of whether you can come with some sort of an algorithm for long path keeping k in mind as the parameter. What would be the complexity of let us say a very naive first cut brute force approach, give that a thought and come back when you are ready.

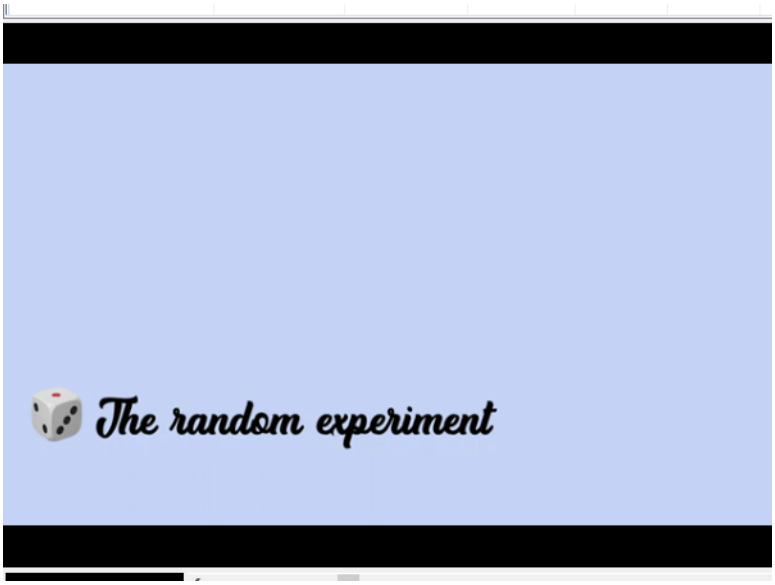
(Refer Slide Time: 05:52)



So, if you tried the brute force approach you would probably have to go over all possible subsets of size k as candidate solutions and for each subset that you considered you would probably have to try all possible ways that you can sequence that subset and check if that forms a valid path or not. Now this is going to be at least end to the k time just for enumerating all the subsets and now you already know that this is not looking like a FPT running time at all.

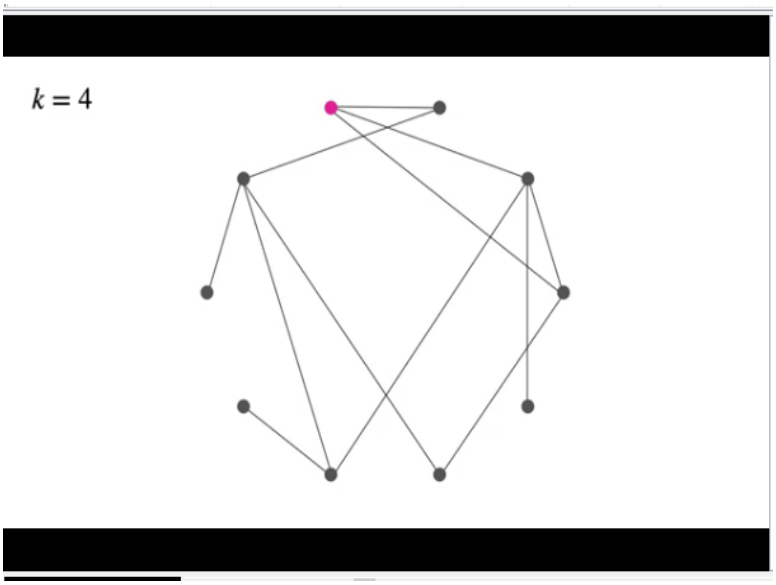
So, we really want to ask ourselves if we can come up with a FPT algorithm and what we are going to describe next is a randomized FPT algorithm that has in fact a single exponential running time and let us explore how this works.

(Refer Slide Time: 06:38)



So, I will begin by describing the random experiment that we are going to perform. So, as usual let me try to describe this with the help of an example.

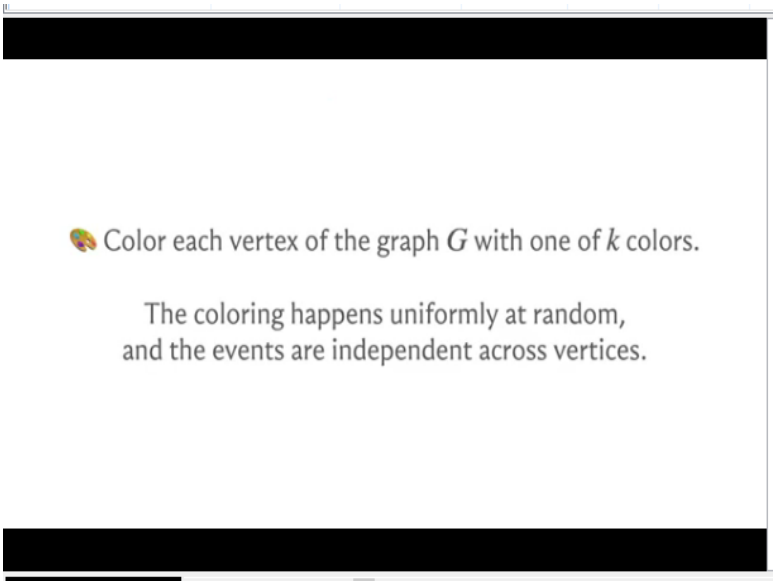
(Refer Slide Time: 06:46)



Let us say this is our graph G and suppose we are looking for a simple path on 4 vertices, the random experiment that we are going to perform is the following. What we will do is we will go to every vertex in the graph and try to assign a label to it uniformly at random where the labels can be thought of as being numbers between 1 and k. So, in this case the numbers will be from 1 to 4.

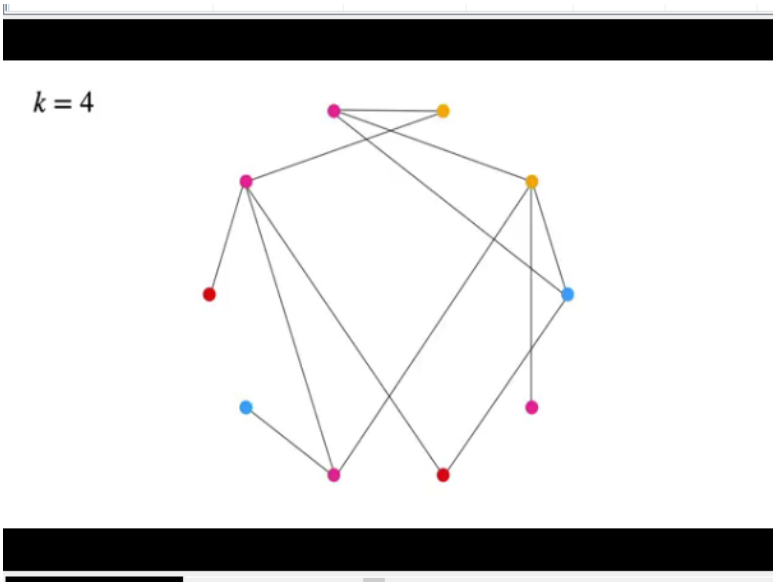
Now this is called colour coding because it is useful to visualize this labeling process some sort of a colouring process. So, you can visualize every vertex is getting one of k distinct colours, so you have a palette which has k colours and what you are doing is colouring each vertex with one of these k colours. Continuing with the example let us say the palette we are working with has the colours red, yellow, blue and pink.

(Refer Slide Time: 07:46)



What we want to do is essentially colour each vertex with one of these colours uniformly at random and notice that these events happen independently for each vertex. There is no correlation between any subset of vertices, it does not depend on the structure of the graph, you just go to each vertex and you perform this labeling exercise or the colouring exercise completely independently.

(Refer Slide Time: 08:12)

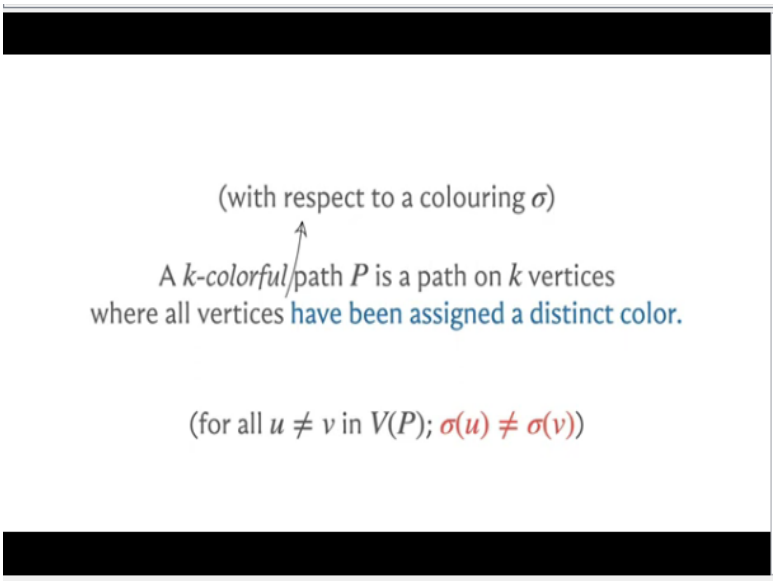


So, at the end of the experiment your graph will end up looking quite colourful, every vertex will have a label or a colour between 1 and k and now the natural question is what is the good event?

(Refer Slide Time: 08:25)

What are we hoping will happen as a result of this experiment? Notice that we are still working with these one-sided error algorithms, so we really want to cross our fingers when we are working with YES instances. So, we are hoping for something specific to happen when the graph does have a path of length k.

(Refer Slide Time: 08:48)



And intuitively what we are hoping for is that this labeling helps our path somehow pop out and if you imagine the labeling as partitioning the graph into k different parts the hope is that if there is a solution, if there is a path on k vertices then it somehow gets spliced across these parts and thinking in terms of colours the hope is that the path gets colourfully coloured which is to say that colours do not repeat on the path.

So, formally we say that a path p is colourful with respect to a colouring sigma if no pair of vertices on this path have been assigned the same colour. So, every colour is used exactly once on this path. So, if we go back to our example and we look at the random colouring you will see that this is indeed a good colouring for us because there is a path on 4 vertices and it has been colourfully coloured.

So, hopefully you are able to identify both the path as well as the fact that we did end up with a lucky colourful colouring. The next question that should come to your mind now is well all this is well and good but how does a colourful colouring help? So, suppose I give you a coloured graph with the promise that if there is a path at all there is a path which is also colourful then can you use this information to actually find path relatively quickly and here by quickly we just mean using FPT time.

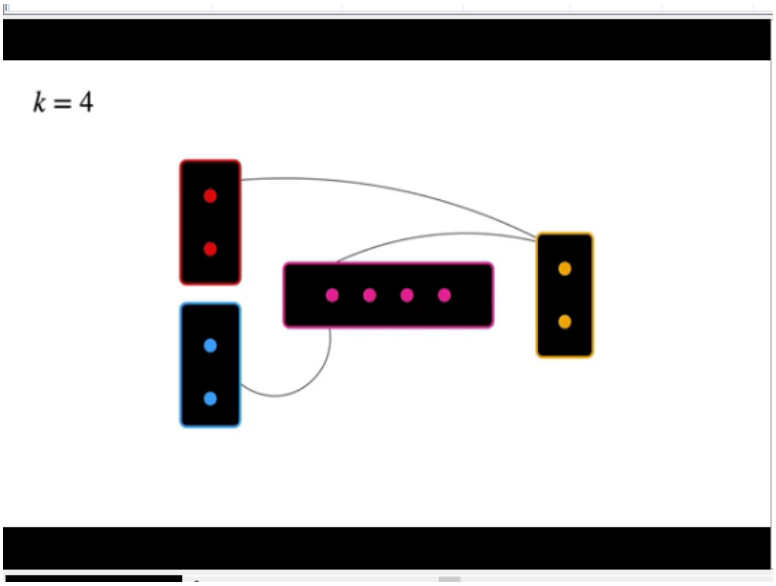
This is a good place to pause and think about whether you can come up with an algorithm if you are in the good situation. Just assume for now that the good situation does manifest, we will come back and discuss the probability that the good event actually occurs. So, for now just take it as a give it a thought and come back once you are ready. Hopefully you had a chance to think about this.

(Refer Slide Time: 10:59)

Notice that the answer to this question will justify our concept of a good event. So, if we are able to identify colourful parts quickly then it makes sense that we would like a random colouring to colour a solution in a colourful way. So, let us go back to the good situation and just stare at what that looks like and instead of tracing out a colourful path let us just try to

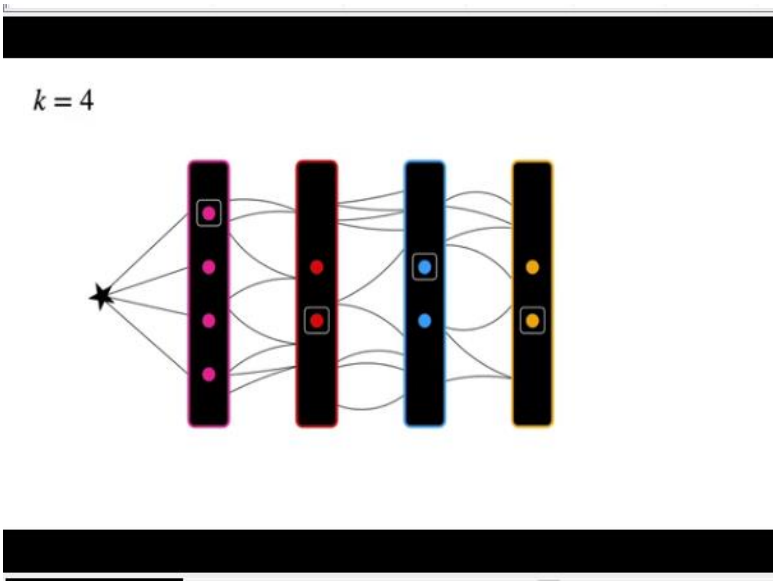
look at this situation from a different perspective. Remember we said that you could think of a k colouring as a partition of the graph into k parts, so let us try to think of this in terms of the partition into these colour classes.

(Refer Slide Time: 11:35)



So, what we know at this point is that there is a path that visits each colour class exactly once, but what we do not know for example is the sequence in which these colours appear on the colourful path. So, for instance it could be that the path begins at red, goes to pink and then yellow and then blue. But it could also be that it begins at pink goes to yellow then blue and then red or it could even be that it starts with blue then pink then yellow and then red; we do not really know this sequence. So, what if we did know this sequence? So, what if this was given to us.

(Refer Slide Time: 12:16)



Let us say somebody told us that look this colourful path actually starts with a pink vertex then it visits a red vertex then it picks up a blue vertex and it finally ends at a yellow vertex, is this information useful? Because notice that if you can actually exploit this information then this information is not very expensive to guess. Notice that there are only k factorial ways in which the colours can be ordered.

So, we can sort of guess the sequence in which the colours appear on a hypothetical colourful k path. So, let us fix an explicit guess and try to figure out what would we do with this additional information. So, if you did not have it figured out when you took the pause previously this is a good time to pause again and see if with this extra information now can you come up with an algorithm to detect a colourful k path if it exists.

Notice that in this picture there are many edges that are completely irrelevant, so see if you can delete some edges; clean up the graph and use an elementary graph algorithm to check if there is a path of length k or not that is colourful. Come back once you are ready. So, once again hopefully you had a chance to puzzle this out a little bit, notice that once you fix the sequence of colours on the solution that you are looking for a lot of edges do become useless to us.

And that we know that they will never feature in this solution. In this particular example we know that the path consists only of edges whose endpoints have the following colour combinations pink and red, red and blue and blue and yellow. So, any edge whose end points have a different combination of colours say edges that have both of their end points within the same part or edges whose end points are for example pink and blue or yellow and red or yellow and pink.

These are all edges that are not relevant to us at all. So, we can simply erase these from the graph, after that we are just left with the edges that go between the consecutive parts of this partition. From here what can we do? Well we want to know if we can go from the leftmost part to the rightmost one with through a path. So, one way to do this is to for example orient all of these edges from left to right.

And add an artificial global vertex to the first part of the partition, also orient these edges from the global vertex to the first part and now you have this nice simple directed graph and

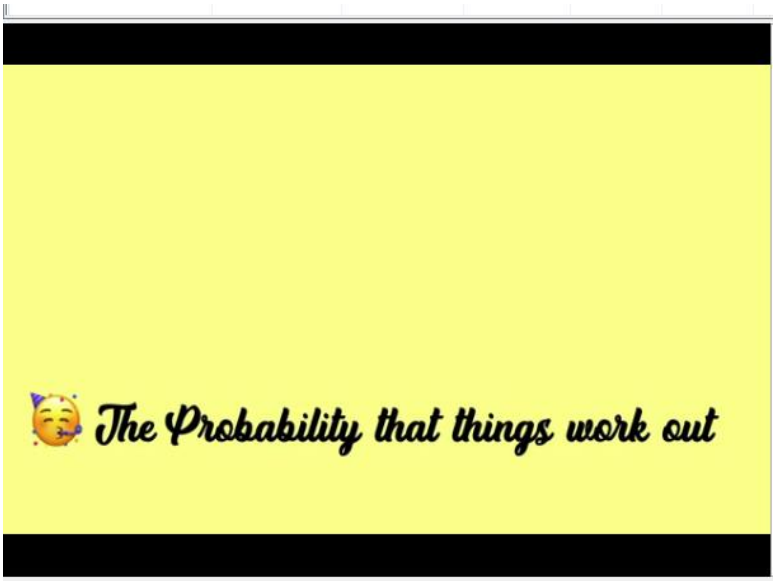
you could start a BFS from this artificially introduced global vertex. So, why is it useful to do a BFS starting from the source vertex? Well notice that if there is a colourful part of the sort that we were looking for then this colourful path is also a shortest path from the newly added global vertex to the last vertex on that path itself which in this case happens to be a yellow vertex.

The reason this is the shortest path is practically by design by the way in which we have eliminated all the other edges in the graph. So, in particular all the vertices at distance 1 from the newly added source vertex or the global vertex are exactly the vertices in the first colour class in this case that is all the pink vertices. The vertices which are at distance 2 are all the red vertices and those at distance 3 are the blue vertices and so on and so forth.

So, we just running a BFS will essentially capture exactly what we needed to capture. So, if the last colour class is reachable from the source vertex then the path that you obtain from that is a colourful path of length 4 by definition and if it is not reachable then well you could not have had a colourful path of length 4 in this setting, because such a path would have been detected by the BFS. So, I hope that that makes sense and by paying this expense of k factorial up front what we finally have is a nice polynomial time algorithm for detecting a colourful path in the setting of a lucky event.

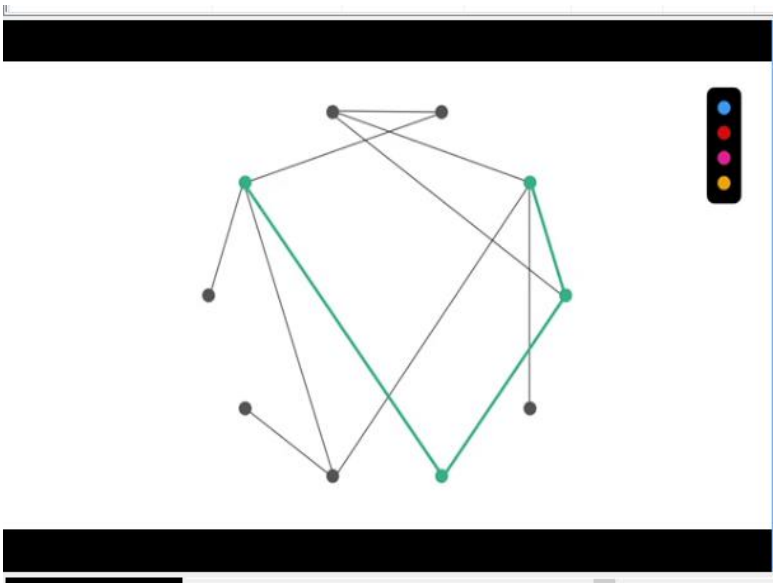
So, if we had a good situation from the random experiment then we have a k factorial times poly n procedure to figure out if a path of length k exists or not. We simply take all the colour classes, line them up in all possible k factorial ways we do these simplifications that involve deleting some edges, orienting some edges and adding an artificial source, run a BFS and we are pretty much done. So, that is a nice simple randomized algorithm for detecting the presence of a k-path.

(Refer Slide Time: 17:31)



The only thing that remains to figure out is well how often do we get lucky, do we get lucky with a decent enough probability that our boosting trick will guarantee us an overall FPT algorithm. So, let us talk about the analysis of the probability that a good event actually happens.

(Refer Slide Time: 17:50)



So, let us recall the concept of a good event. We call that we have been saying that a random colouring is good for us if it colours the vertices of a path on k vertices, if it exists of course distinctly with all different colours. So, in the running example that we have been working with k has been 4, so we have been working with a palette of 4 colours blue, red, pink and yellow and what we want is that if there is a path on 4 vertices then it should get each of these colours.

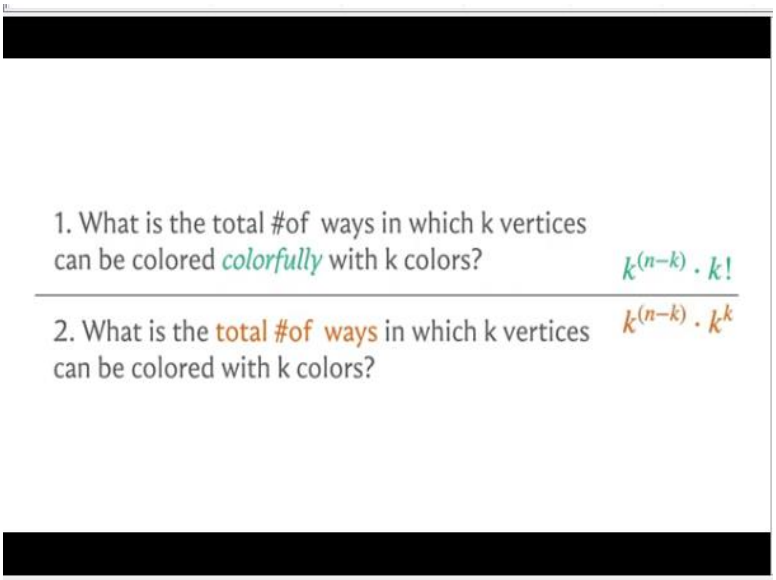
The 4 vertices on this part should be coloured with each of these colours exactly once. Now how do we analyze the probability that a random colouring does have this property? By the way let me just get one small thing out of the way; notice that if a graph is a YES instance it may actually have multiple witnesses to the fact that it is a YES instance. In other words there may be several paths of length at least k.

Even in this example it is not very hard to find other paths on 4 vertices. However, in the spirit of doing a worst case analysis we will just focus on the probability that an arbitrary but fixed path in the graph is coloured colourfully. This is sufficient because in a YES instance you are going to have at least one path. So, we might as well fixate on any one of them, it does not matter which one.

But just fix your attention on one path and compute the probability that that gets colourfully coloured. If that is a decent probability; then in the real world when your graph actually has multiple solutions the probability that any one of them gets colourfully coloured will be if anything even better. So, it is enough to just do this and notice that in the worst case if your graph happens to be such that it has only one solution, this calculation will also account for these kind of extreme cases.

So, with that out of the way let us just consider what is the probability that this arbitrary but fixed path on k vertices gets coloured in a colourful way? Now the way we calculate probabilities is fairly standard; we look at the number of events in the sample space that work out in our favour that are good events for us and then we look at all possible things that could happen good and bad put together.

(Refer Slide Time: 20:20)



So, we really need to answer these 2 questions. The first is what is the number of ways in which these k vertices can be colourfully coloured with k colours? Those would account for the good events and to account for all situations we need to understand what are all possible colourings; what is the number of all possible ways in which these vertices could be coloured? Now the ratio between these 2 would be the probability of the good event.

Now please take a moment here and try to answer both of these questions which are not very difficult if you have pen and paper handy, please just try to work through it, both of these are fairly straightforward counting questions, do come back once you have had a go at it. First let us consider the number of ways in which these k vertices can be colourfully coloured. Since we have k vertices and k colours and what we really care about is the fact that every vertex should get a distinct colour what we are really looking at is a permutation of these k colours.

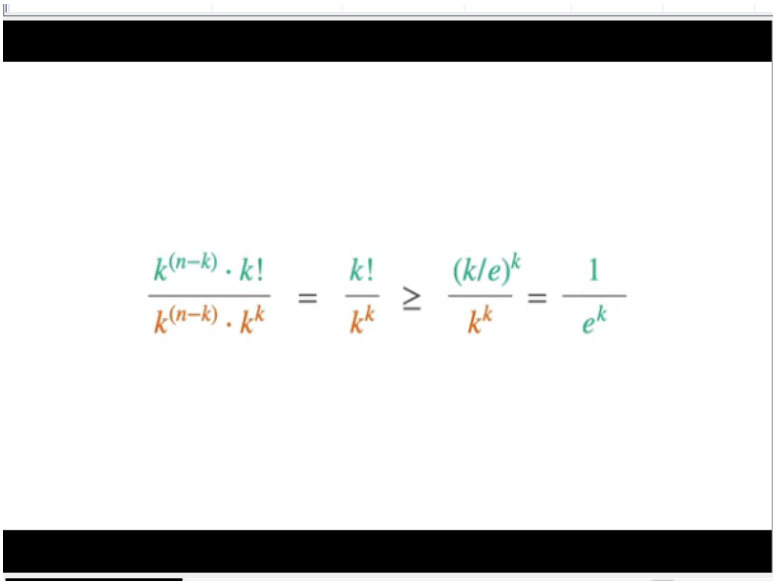
So, for these k vertices there are k factorial goods scenarios and for each of these good scenarios the remaining n - k vertices can get coloured in whatever fashion we do not really care. So, for each of the k factorial permutation colourings on these k vertices we can extend these colourings to colourings of the complete set of n vertices by colouring the remaining vertices in any way we like.

Now what are the number of ways in which we can colour the remaining vertices, there are n - k vertices that remain and since we do not care about how they are coloured there are no restrictions, they can be coloured in k to the n - k different ways. Each of these vertices have

k choices of labels and all of them are valid. So, the total count is k to the n minus k times k factorial for the good event.

Now for collecting the total number of events that is really just looking at all possible colourings, so that is going to be just count all possible colourings with no constraint. So, that is going to be k to the n but just to make it easy for us to simplify I am going to break that down as k to the n - k times k to the k.

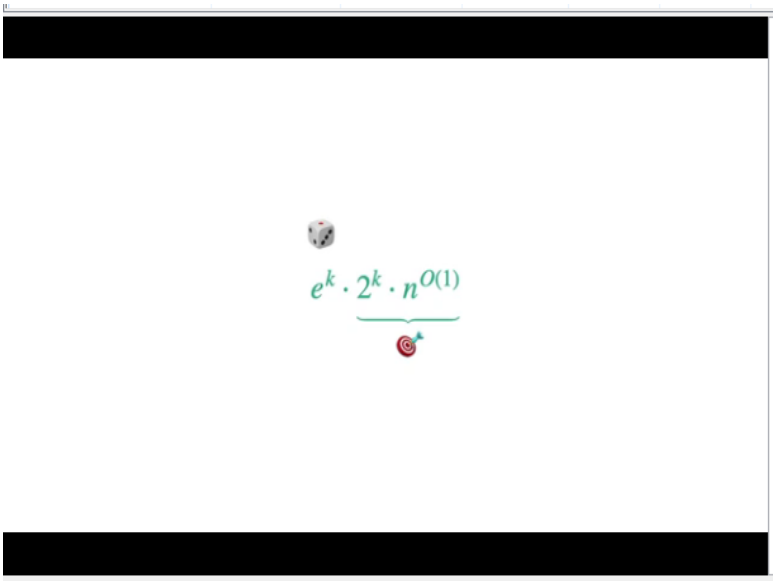
(Refer Slide Time: 22:46)



So, we see that the k to the n - k term cancels out and what we are really left with is k factorial over k to the k. Now it turns out that k factorial can be lower bounded by k over e to the k, this is a standard inequality and it is quite helpful in this case, because now you see that the k to the k cancels out and we are left with 1 over e to the k. Notice that if you apply the boosting arguments that we have discussed in the previous module you can get this up to a constant by simply repeating the random experiment e to the k many times. So, now let us put everything together and take a look at the overall running time.

(Refer Slide Time: 23:25)

(Refer Slide Time: 23:28)



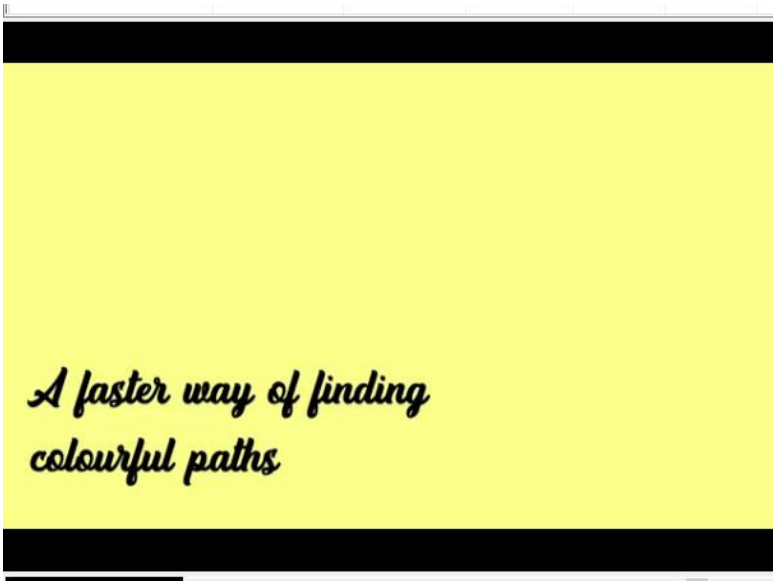
So, what we have is e to the k coming from the boosting part of the argument, the repetition of the random experiment and we had this k factorial amount of work that we did when we guessed an ordering of the colour classes and then we did a little bit of cleanup simplifying the graph a bit adding a source vertex and then running a BFS. So, all of that was a polynomial overhead.

So, right now we have a running time that looks like e to the k times k factorial which is a pure function of k multiplied by a polynomial in n. So, this is very much a randomized FPT algorithm for the problem of finding a path of length at least k. So, we are pretty much done but a question that you can always ask yourself at this stage of the discussion is the question of whether we can do better in particular can this running time be improved.

And notice that as far as this algorithm is concerned there are 2 major components to the running time. The first is e to the k which comes from repeating the random experiment, so that we are confident about the success probability, this is the boosting part. The second component is the k factorial poly n which is the work that we needed to do to find a colourful path under the assumption that the good event actually panned out.

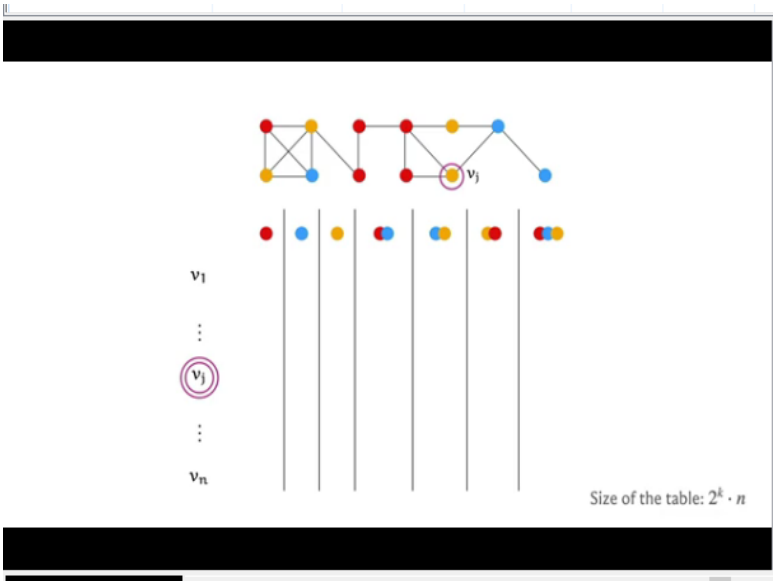
Now it is the second part that we can actually replace with a different approach which is slightly more sophisticated and gives us a better running time in particular we can improve the k factorial to 2 to the k using dynamic programming over subsets of colours instead of trying out all possible sequences.

(Refer Slide Time: 25:13)



So, let us go ahead and discuss what that could look like so that we get this improvement in our running time.

(Refer Slide Time: 25:19)

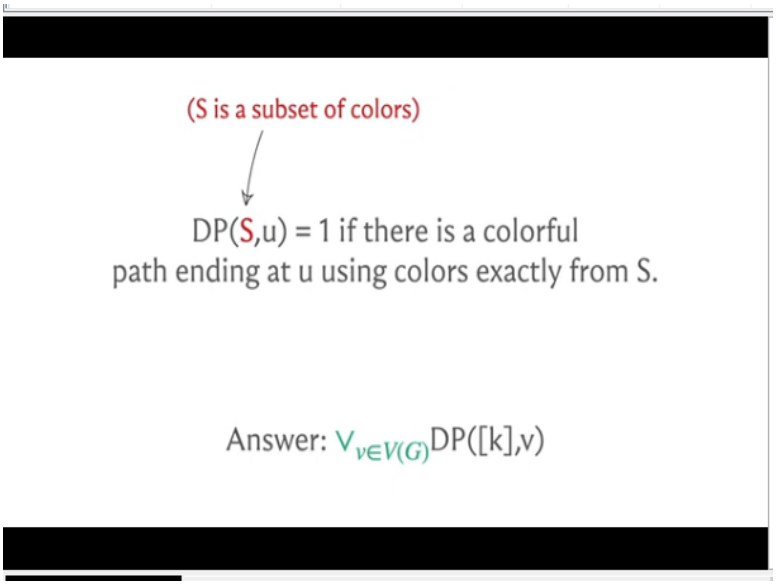


So, remember that the input at this point is a colourful graph well I should not use the term colourful it is a coloured graph, every vertex has been assigned one of k different colours in this example we are just going with k equals 3 and the 3 colours are blue, red and yellow. And the question is if this graph has a path on k colours that is colourful which is to say that every colour is used exactly once.

So, what is the DP table look like? Well its rows are going to be indexed by the vertices of the graph V 1 through V n and the columns are going to be indexed by the subsets of colours. So, there are 2 to the k columns, each column corresponds to a different subset of colours and you could just organize them in increasing order of size and when you are looking at subsets that have the same size it does not really matter how you order them.

So, notice that the size of this DP table is 2 to the k times n, so that looks promising; I mean at least if we know how to fill out this table efficiently and if we can give it some meaningful semantics then perhaps we are on the right path. So, what are the semantics of this DP table, what do we want to populate it with and when do we populate it one way or the other?

(Refer Slide Time: 26:40)



Well so let us consider the entry corresponding to the subset S and the vertex u. So, this is the row corresponding to the vertex u and the column corresponding to the subset S, we are discussing that entry. So, this is going to be a table with boolean entries, so the entries will be 1 or 0 or true or false you can pick whatever terminology you like and the semantics is that the value is 1 if there is a colourful path ending at u that uses colours exactly from S.

So, here our notion of colourful is just that colours do not repeat. So, you have some path that ends at u using colours exactly from S which is to say that every colour in S is used exactly once that is the sense in which we want the path to be colourful. Notice that this is fairly intuitive in the sense that if you are used to thinking about DP as an approach that builds on partial solutions.

This is the natural projection of the notion of a colourful path of length k to sort of smaller parts. So, the natural projection is that your colourful but with respect to a smaller subset of colours. So, is this a useful semantics to work with in the sense that if we do manage to fill out the table with the semantics will we get to our answer. This is a good place to pause and think about whether the answer to your original question which is does G have a colourful path on k vertices.

Can you find that answer from this table if the table was correctly populated with these semantics? So, hopefully you had a chance to think about that. Notice that the ultimate thing that you are looking for which is a colourful path on k vertices is essentially S being equal to 1 to k, S is the set of all colours, it is the subset corresponding to the universe the full set of

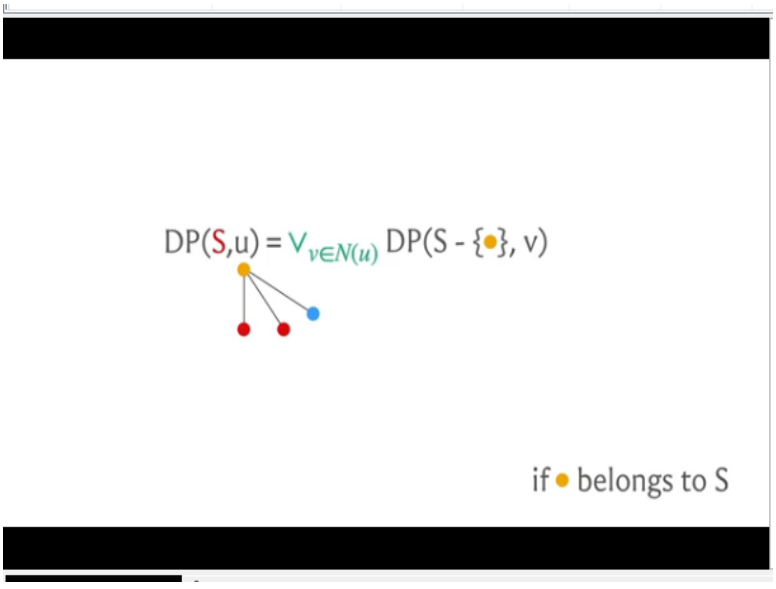
colours and that if a colourful path exists well it must end at some vertex; we do not know which vertex.

So, we will simply try all possible vertices, all possible choices of the last vertex on the colourful path that we are looking for. If there is a colourful path then at least one of these DB table entries must be one and if there is no colourful path well of course then all of these entries must be false. So, by just doing an r over these n entries. So, essentially taking a boolean r over the last column in the table you get to your final answer.

So, clearly these semantics are useful if we manage to populate the DP table with respect to these semantics then we would be done. So, the only thing left to discuss is the recurrence. Once again at this point I would strongly recommend that you pause and think about what the recurrence would look like, how would you populate this, I will in fact not discuss the base cases because they are completely straightforward and hopefully you will be able to figure those out for yourself.

In case you have any questions please do ask at the forums and we will be happy to clarify further; you could also look at the notes on the course website where there are more details, but for now let us just focus on the recurrence and I hope that you are able to pause and think about this for a bit and come back when you are ready.

(Refer Slide Time: 30:05)



So, let us take a look at a generic entry DP S, u and just to make this a little bit concrete let me assume that u itself is coloured yellow, it could be any colour, it is going to be some

colour. So, of course the first thing we want to ensure is that this colour belongs to us because we are looking for a colourful path which is colourful with respect to S which ends at u. So, the path itself includes u.

So, if this colour does not even belong to S then all bits are off we just set this entry to false and move along. But if this colour does belong to S then well, then we can focus on the rest of the path and notice that the penultimate entry on the path, the path ends at u what was the previous vertex? Well the previous vertex must have been some neighbour of u. So, well we do not know which neighbour.

So, we are going to try all possible neighbours and we are going to look up the DP table entry corresponding to well parts ending at those neighbours, we try out each neighbour as a candidate for where the sub path should have ended and of course we will have to remove the colour of u from the set S because we have already used this colour we are not allowed to use it again. You just have to go ahead and look up these DP table entries.

And if any one of them is one then what does that mean? Well that means that you do have a colourful path using all colours in S except for the one that was already used on u and such a path in fact ends at a neighbour of u. So, you know that this path can in fact be extended to a path that ends at u and uses all colours in S exactly once. So, if the right hand side of this equation evaluates to 1 we know that the left-hand side must evaluate to 1.

On the other hand if the left-hand side is 1 which is to say that you do have such a path then we well just look at what neighbour of u did that path end just before it actually ended at u, that sub path of length size of S - 1 will be a witness for some entry on the right hand side to be triggered to true. So, therefore we see that this equation makes sense this recurrence holds and of course you do have this top level sanity check that the colour of you must belong to S.

So, if the colour of u does not belong to S then it is anyway false; otherwise the value of the DP table entry is completely determined by the entries that have been shown to you on the right hand side of this equation. So, I hope that this recurrence makes sense; you can see that computing any entry of this DP table is just a polynomial amount of work. In fact it is just proportional to the degree of the vertex u that is the number of table lookups that you need to do after a simple sanity check.

And therefore the overall running time is simply the size of the table which was 2 to the k times n times a polynomial overhead. So, basically that is the improvement that we claimed and that is the improvement that we now have modular description of the base cases. So, of course the base cases are very important, your algorithm will not run without it, but they are also fairly easy to fix up. So, I am going to leave that as an exercise, again with the invitation that you should feel free to reach out or just leave a comment in case anything about the base cases is not clear.

So, I hope that made sense, this is a really fun technique and as I said it is applicable well beyond long path and I think the assignment questions should give you some practice on how this can apply to other patterns as well. So, do have fun with that and I will see you in the next video with a slightly different version of the colour coding technique. Thanks for watching and bye for now.