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

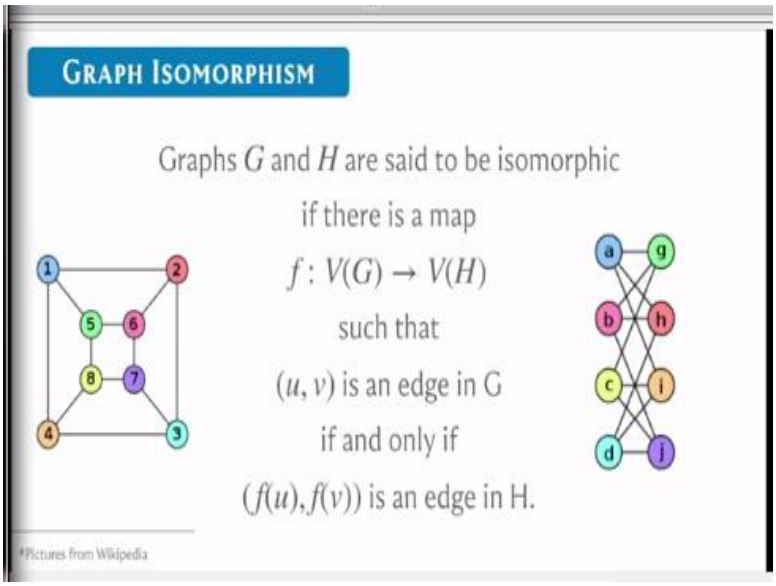
Lecture-19
Random Separation and Subgraph Isomorphism

So, welcome to the 4th module of the 4th week. We are still discussing randomized methods for the design of FPT algorithms. And in this week we will introduce a new technique called random separation. And we will try to understand this technique through a problem called subgraph isomorphism, which is again a problem that you are probably meeting for the first time. You can see a star against the name of the problem in the slides which might make you think about whether there are some terms and conditions that apply.

And this is indeed the case, we will not really be solving subgraph isomorphism in it is full generality. But we will be solving it for a special class of graphs and specifically these will be graphs that have bounded maximum degree. For reasons that will hopefully become clear as we go along. Now you have already seen techniques like colour coding and chromatic coding. So, in spirit random separation is not very different, the fundamental idea is still to use some concept of colouring or partitioning and in particular a random partition of the objects in your instance.

In a way that hopefully cleverly makes your solution pop out whenever it works out well. But the specific mechanics of the way in which we execute this partitioning is going to be of a very different flavour compared to what you have already seen before. So, with this background in place let us get started, we are going to begin by defining the subgraph isomorphism problem. But to actually talk about subgraph isomorphism we first need to understand the notion of isomorphism which is going to be a very fundamental concept throughout this lecture.

(Refer Slide Time: 01:39)



So, let us make sure that we get this right. So, 2 graphs are said to be isomorphic when roughly speaking they are essentially the same graph. But how do we capture this idea of 2 graphs being essentially the same? For instance if you look at the 2 graphs on your screen right now, they are in fact the same graph but they look so different the way they have been drawn out. That at first sight it may not be obvious to you that these are in fact the same graphs. Even if you write them out in adjacency matrix form they may not actually match cell by cell because you may have ordered the vertices in such a way that the rows and columns do not match.

But the idea is that there is a way of rearranging the rows and columns of one of these adjacency matrices, so that it exactly tallies with the adjacency matrix of the other graph. Another way to say this is that you are essentially working with the same graph up to relabeling. So, let us formalize it in this way. Suppose you are given 2 graphs G and H, we say that they are isomorphic if you can basically map the vertices of one graph to the vertices of the other.

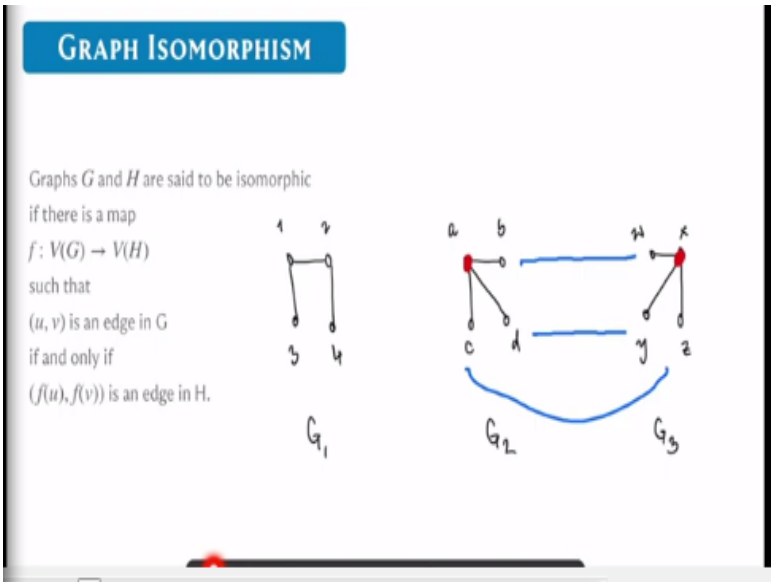
In a way that this map is a bijection, so you do not have any awkwardness like 2 vertices mapping to the same vertex or some vertex in the target graph not having a pre image at all. So, it is a clean bijection every vertex in G gets mapped to exactly one vertex in H and every vertex in H has a vertex that is being mapped to it. But of course just coming out with a bijection is not enough because that would work for any 2 graphs which have the same number of vertices.

You want this bijection to actually preserve the structure of the graph and what do we mean by that? What we mean by that is that if you have an edge in G, then it gets mapped also to an edge in H. So, for example if u and v are a pair of vertices which are adjacent in G then f of u and f of v should also be adjacent in H. Similarly if a pair of vertices are non-adjacent in G then they should not suddenly become adjacent in H when you look at where they have been mapped.

So, in other words u v is an edge of G if and only if f of u, f of v is an edge of H that is when we say that the graphs G and H are isomorphic. You can think of this as just a way of relabeling the graphs as you can see in this picture which by the way is from wikipedia. So, if you want to find out more there is a link in the description which leads to the wikipedia page and that has a lot more context and background for the notion of isomorphism.

And also the computational problem of graph isomorphism discussion about which is well beyond the scope of this lecture. So, for now it is enough if you just get comfortable with this definition, so given 2 graphs when do we say that they are isomorphic? Again roughly speaking what you want to keep at the back of your mind is that these graphs are exactly copies of each other. So, that is the intuition that you want to carry with you but remember that you should be also able to tally this intuition with the formal definition.

(Refer Slide Time: 04:58)



Just to get some practice let us look at an example. So, here you can see 3 graphs that have been drawn out, 2 of which are isomorphic to each other and the 3rd one is the odd one out. So, take a moment here, just to see if you can identify which is the pair of isomorphic graphs. Visually it should be very visible but ideally it will be great if you can come up with an actual mapping that witnesses the isomorphism.

And perhaps even a reason for why the graph that is not isomorphic to the other 2, in fact cannot be isomorphic to the other 2. So, there is no map that can witness it is isomorphism. So, take a minute here and come back when you are ready. Hopefully, you had a chance to pause there for a minute. Now if you look at G 2 and G 3 they do look pretty much the same, they practically look like mirror images of each other.

So, if you just quickly guessed that they are the 2 isomorphic graphs then you would be quite right. But it is also easy to actually formalize this with a mapping, so for instance you can map the vertex a to the vertex x. By the way notice that this is first because these are the only 2 vertices of degree 3 in both of these graphs. But for the remaining vertices pretty much anyway that you map them should work, so for example we can map b to w, d to y and c to z.

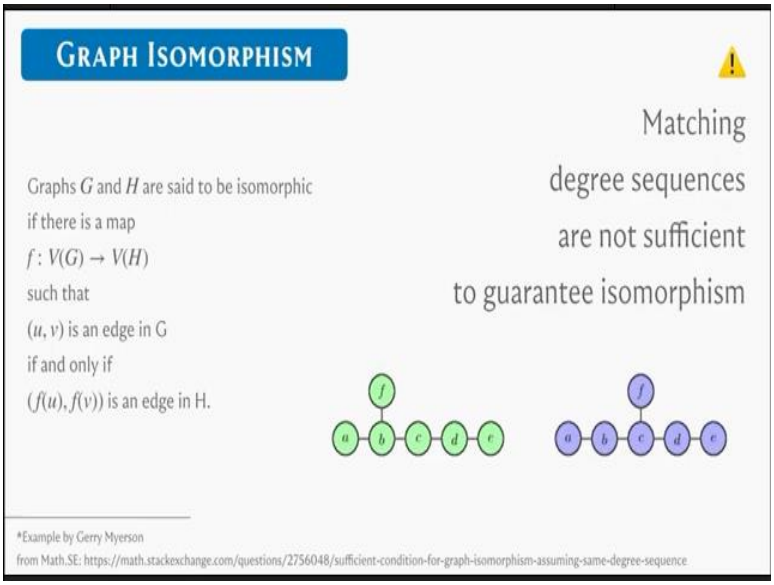
And I did that here because that was just the most convenient thing to draw but really any other permutation of these 3 vertices would work out just the same. Now on the other hand let us look at G 1, notice that G 1 does not have any vertices of degree 3. And based on this you can convince yourself that you will not be able to find isomorphism from G 1 to either G 2 or G 3. In general if you have 2 graphs G and H and let us say you are mapping a vertex x and G to a vertex u in H.

Notice that the degrees of x and u must be the same otherwise your isomorphism is not going to work out. In particular what this means is that if you wrote down the degree sequences of G and H, by the way what is a degree sequence? It is just the list of all vertex degrees listed in let us say ascending order. So, if you wrote down the degree sequences of G and H they must be exactly the same for you to have any hope that they are isomorphic. So, that is not happening between say G 1 and G 2 here, so we conclude that G 1 is the odd one out.

Now a natural question might be that whenever you say that 2 graphs are not isomorphic then is this reason can you always spot a pair of vertices whose degrees do not match up in the degree sequence? It turns out that the answer to this question is no, so the degree sequences matching is a necessary condition for isomorphism but it is not necessarily sufficient. So, if you are seeing this fact for the first time it is a cute puzzle to just come up with an example of a pair of graphs that have the exact same degree sequence.

But you can see that they or you can show hopefully that they are not isomorphic. So, in the next slide I will show you such an example, so this is a spoiler alert if you want to pause this would be a good time to do it.

(Refer Slide Time: 08:23)



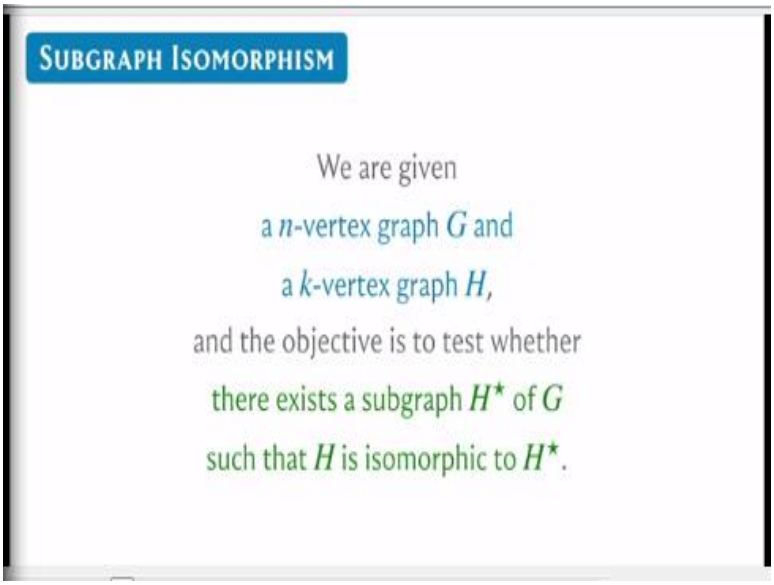
So, here is an example of 2 graphs that have the exact same degree sequence but you can see that because they again have unique vertices of degree 3 which would be b in the green graph and c in the purple graph. Any isomorphism must map b to c and their neighbors in some fashion but then by a case analysis you can see that you will not be able to extend this map to an isomorphism.

So, the take away message is that if your degree sequences do not match then your graphs are certainly non-isomorphic. But if your degree sequences do match then you may still have more

work to do. Of course graph isomorphism is one of the most fundamental and one of the most well studied computational problems, the question of determining whether G is isomorphic to H or not.

And there is been some very exciting progress on this problem in recent times although we still cannot squarely place it in p we do know much more than we knew say a few years back. And once again let me point you to the links in the description if you are curious about this problem and you want to find out more. In the meantime let us move on to the problem that we want to discuss in this lecture which is of course the sub graph isomorphism problem.

(Refer Slide Time: 09:39)

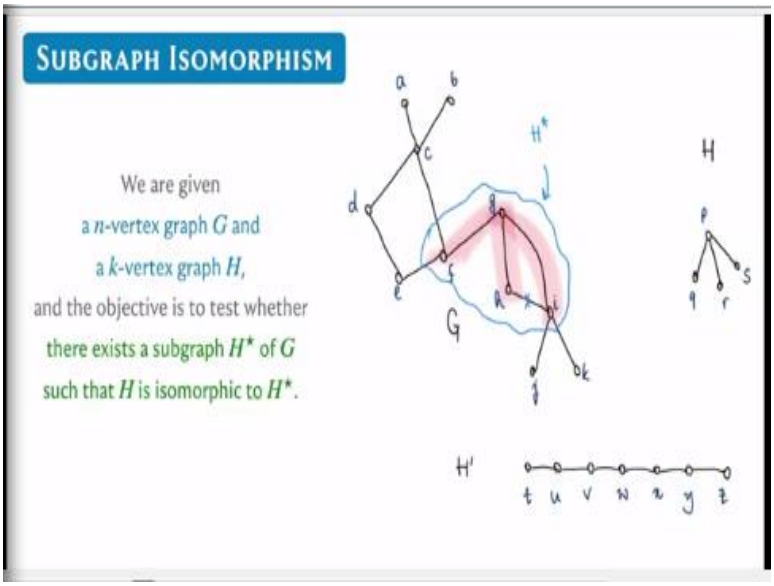


So, here we are still given 2 graphs G and H but now I want you to imagine G as being this big graph on n vertices and we will often refer to it as the host graph. And I want you to think of H as being this relatively smaller graph on just k vertices. And this is something that we will often refer to as a pattern graph and the goal is to find a copy of the pattern in the host. Now of course my use of words like big and small here, completely loose and informal but it is just to help you build up some intuition.

So, I think of G as being this big forest in which you are just trying to find a specific tree which is the graph H. And this might remind you of some of the discussions we have had in colour coding, it has this flavour of finding a small object in a large space. So, that is what the subgraph

isomorphism problem is about. So, we are trying to see if G has a subgraph H star by the way this does not have to be an induced subgraph, any subgraph is fine. But you want to know if G has a subgraph let us call it H star and what we want is that H star is isomorphic to H that is the goal.

(Refer Slide Time: 10:57)



Again as usual let us just try to get comfortable with the help of an example. So, you can see here that we have the graph G and the graph edge is just a star with 3 leaves and these highlighted edges show you a copy of H in G. And notice that as we said earlier this does not have to be an induced subgraph. So, this edge here can simply be deleted if you want to be left with something that is an exact copy of H.

Now if you want some practice for yourself you want to pause here and just see if you can find a copy of H prime which is a path and it is a path on I believe 7 vertices. So, you want to check if G has a copy of H prime feel free to take a look right now. Some of this might remind you of colour coding and you might be wondering, well, cannot we just use the same techniques?

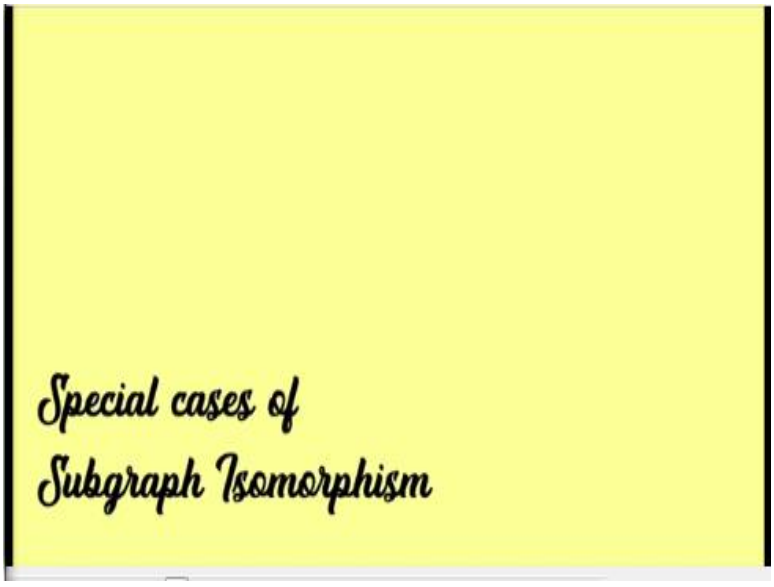
So, cannot we? For example randomly colour the graph with k colours and just try to find a copy of any arbitrary graph edge in this colourful graph, hoping that it gets coloured in a colourful manner. Now it is true that the probability that the object that you are looking for will get

properly coloured is going to be the same as the probabilities that we had worked out when the object was specifically a path but the challenging bit comes after this.

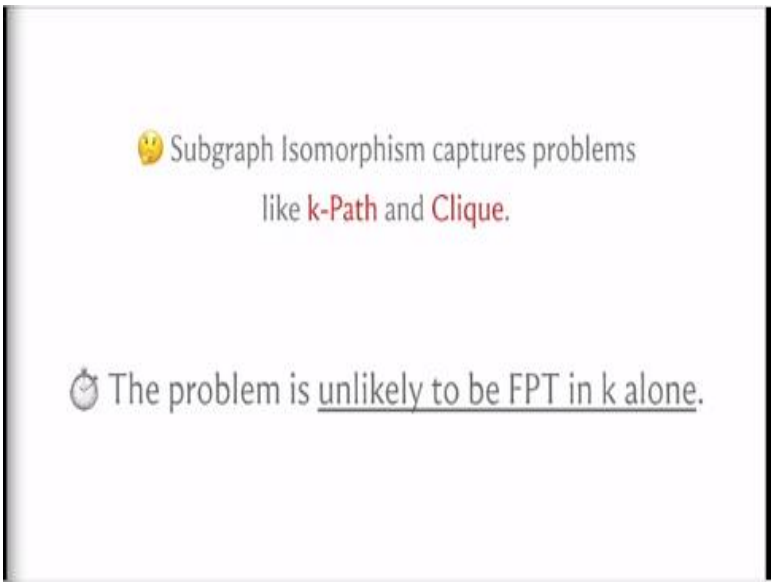
How do you find the object that you are looking for? If you go back and think about what we did in the case of a path, we had 2 strategies in one we could organize the colour classes based on the sequence in which they appeared in the path. And then we could do some sort of a BFS after some appropriate simplifications. And the other idea that we had was to use some dynamic programming.

But if your graph edge is some completely arbitrary structure then it is not really clear how you would make progress. In fact even back then we did say that this technique that we have developed will work for structures that go beyond a path. But with the caveat that those structures still have to have some leverage, something that you can exploit to apply the kind of dynamic programming strategy that we were discussing. So, it will not work for any arbitrary graph edge and we believe also with some good reason.

(Refer Slide Time: 13:15)



(Refer Slide Time: 13:17)

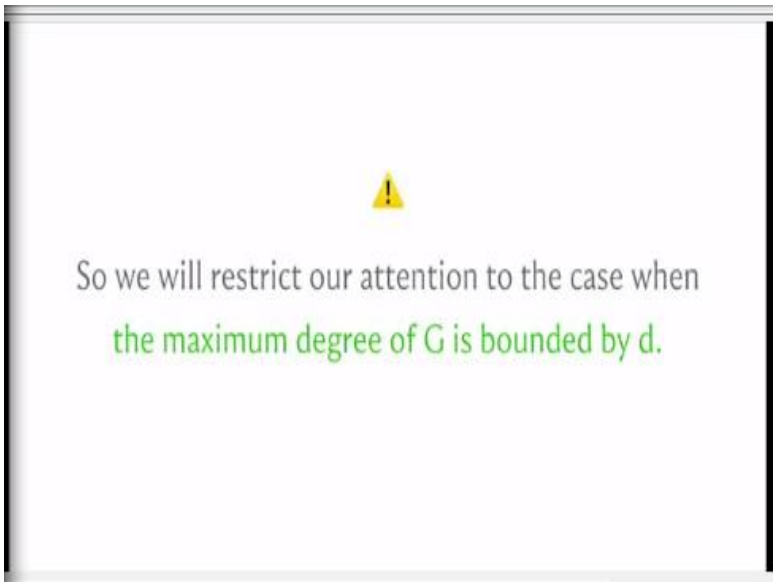


So, let me say that the problem of finding a clique on k vertices is a special case of subgraph isomorphism. This is the case when the graph H which is your pattern graph is simply a complete graph. Now for reasons that will become clearer only much later in the course, it turns out that we do not really expect to have an algorithm which is FPT and k for the problem of finding a clique on at least k vertices.

But if you were to parameterize by the size of the pattern graph and subgraph isomorphism that is precisely a problem that you will solve on the way the problem of finding a k clique is subsumed by the problem of subgraph isomorphism parameterized by the size of the pattern graph. So, now this might sound like a bit mysterious at this stage, I am asking you to take it on faith that this problem is super general. And we do not really expect to be able to solve it in FPT time.

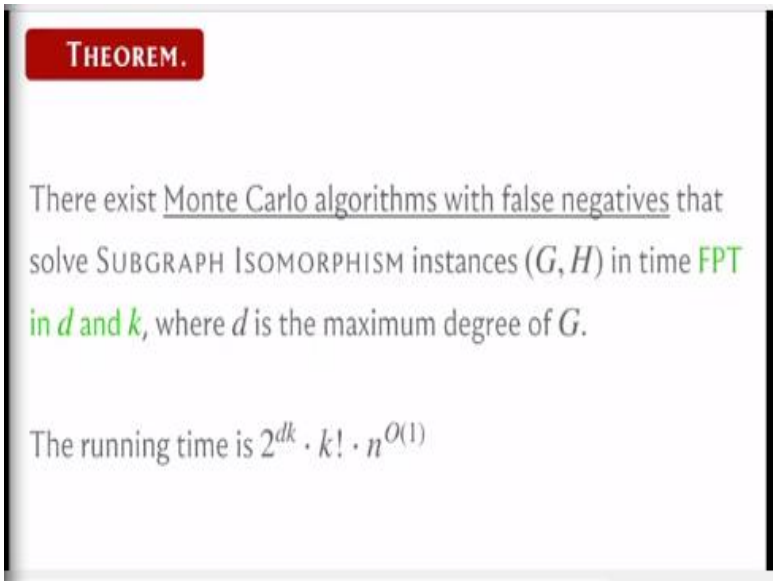
As I said we will revisit the problem of finding a clique in the last week of this course and then you might jog your memory and hopefully come back to this point about why we should adjust our expectations with regards to what is it that we can achieve for this very general problem of subgraph isomorphism o, it seems like we got a bit stuck right at the outset but remember we made some opening remarks about how terms and conditions apply and so on. So, when you are stuck in this fashion, well, one thing that you can do is to give yourself a little more wiggle room, a little more power.

(Refer Slide Time: 14:57)



And the way we are going to do that here is by just allowing ourselves to work with a slightly bigger parameter. So, the parameter that we will be working with is going to be k which is the size of the pattern graph plus d which is the maximum degree of the host graph, the maximum degree of G. In particular we will show the following.

(Refer Slide Time: 15:14)

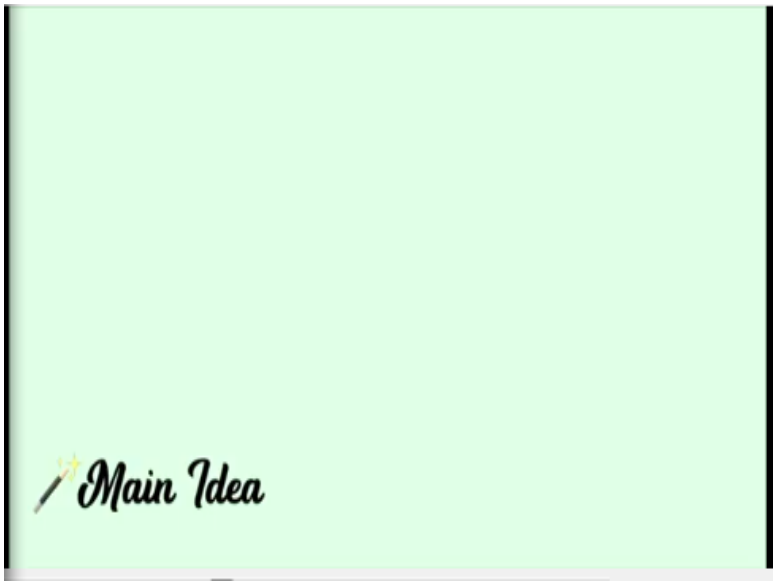


As usual we will want to come up with a randomized algorithm that has one sided error and specifically we will only report false negatives and the running time of this algorithm is going to be FPT in d and k, the specific form of the running time is going to be 2 to the dk times k factorial with the usual polynomial overhead. But even if you do not remember the specific form

your major takeaway from here is that the running time is FPT in the combined parameter d and k where once again d is the maximum degree of the host graph G.

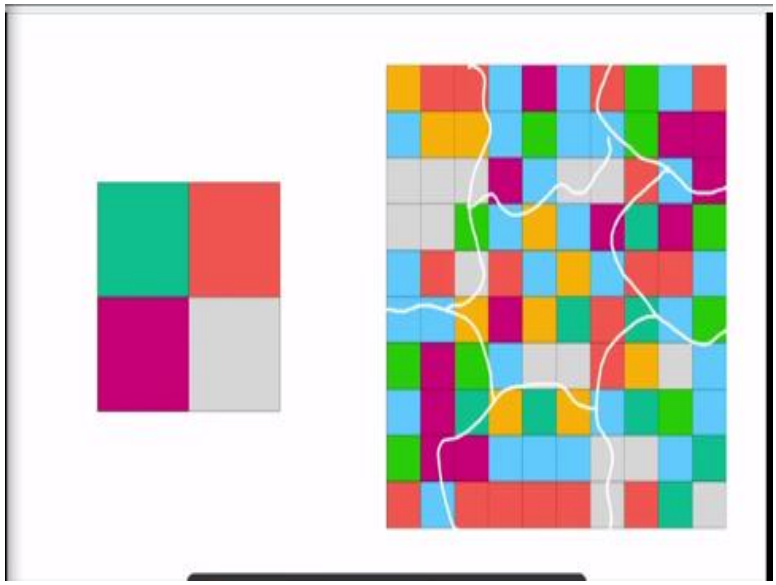
Also for reasons that are unknown to us right now but that we can take as a G 1. Let us also just remember that we do not expect subgraph isomorphism to be FPT parameterized by k alone, where k is the size of the pattern graph. So, now that we know what we are looking for.

(Refer Slide Time: 16:08)



Let us actually turn to a discussion of the algorithm.

(Refer Slide Time: 16:11)



So, before we get started though let me give you a slightly physical analogy, it is of course not perfect, so do not take it too seriously but I hope that it is a helpful mental model. So, imagine that somebody has challenged you to this puzzle where on the one hand you have a large sheet of paper which is essentially a very colourful grid. And on the other hand you have this small pattern and you have been asked to figure out if this pattern occurs in this grid or not.

Now normally at every point that I give you a concrete example like this I ask you to pause and figure out what is going on. But here I think it is really a waste of time, so do not bother. As a matter of fact I also do not know if this pattern quite appears in this grid or not but that is not so relevant here. The important thing is that in principle if you really had to find this pattern then you would probably attack the grid from left to right, top to bottom go through the whole thing systematically.

And check for every 2 by 2 sub grid that you can find if the colours match with the pattern that you are looking for. So, this would be an analogy for what a brute force approach would look like. And let us say you decide this is too boring for you, so you just set the puzzle aside and say you will come back to it later. But when you do come back to it let us say that you find that somebody was just having some fun and they have torn up this piece of paper into small scraps.

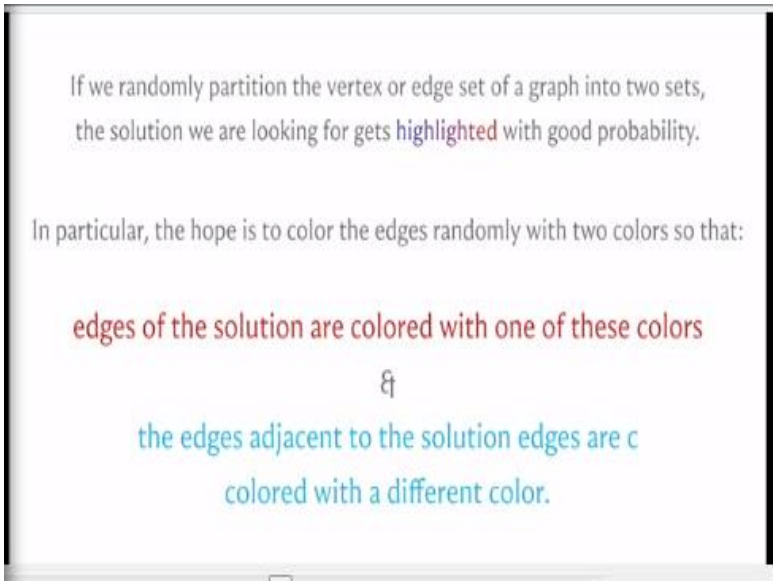
So, all you have when you come back to it is these small shreds of paper from the original puzzle. But let us say that you get very, very lucky and somehow it happens that exactly this pattern is one of the shreds. You do not know which one of these scraps of paper has the pattern. But suppose somebody tells you that you know instead of your big piece of paper here is a box of scraps and one of these scraps is the thing that you are looking for.

If that were to happen of course if you just randomly cut the paper into small pieces, I do not know what the chances are that you will get this lucky but we will come back to that later. Suppose it did happen, right, then would you be a happy person, I would suggest that yes you should be happy. Because now what you can do is start going through these pieces of paper and if a piece of paper is not the exact size of the thing that you are looking for you can simply discard it.

Because remember you have been promised that the pattern is exactly one of the pieces of paper. So, you can simply disregard any piece of paper that does not match the size of the pattern and all those shreds of paper which look like 2 by 2 subgrids, you can just hold on to those and just go through them one by one and see if they match the pattern. So, your life becomes somewhat easier compared to the process of going through the entire large grid.

Because hopefully what will happen is that you are able to throw away a lot of stuff and the stuff that remains it just becomes very easy to check. Again this is by no means a perfect analogy but it is kind of close visually to what we are going to be doing with our graph, so I thought I will give you this warm up story before we actually get to the algorithm.

(Refer Slide Time: 19:29)



So, what does our algorithm actually do? It is going to try and randomly partition the edges of the graph into 2 parts you can think of this as colouring the edges with 2 colours, let us say red and blue. And our hope is going to be that every edge that belongs to a solution, so remember that a solution is simply an isomorphic copy of the pattern graph H which sits inside G as a subgraph, so we will continue calling this H star.

So, what we want is that every edge of H star if it exists should be coloured with one colour, let us say that colour is red. And every edge that is incident to this solution H star which is to say

every edge which has one end point in H star and the other end point outside of H star, we want all such edges to get coloured blue.

(Refer Slide Time: 20:23)

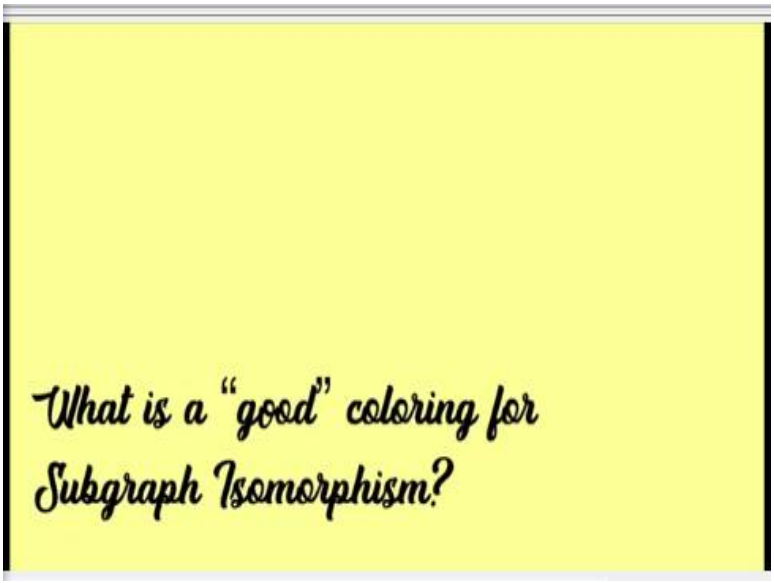
So, in some sense you can imagine that these edges which are incident on H star the kind of form a protective fence around the solution. And just thinking ahead a little bit to compare this with the pieces of paper analogy. Let us say somebody gives you this big graph G where the edges are coloured with 2 colours red and blue. You know that if this colouring was a good colouring in the sense that we just described it, it has these properties.

Then we know that the blue edges are useless for us, so they do not really belong to H star, so we can just delete all of them. Now when you delete all the blue edges the graph falls apart into some connected components. And now what we know is that one of these connected components must exactly be H star. And we will make this more precise in a bit but this is like saying that somebody has managed to tear up your graph into some pieces with the guarantee that one of the pieces is in fact exactly what you are looking for.

So, once again just like we did with the piece of paper you can disregard all pieces that do not have exactly k vertices, all components that are bigger or smaller than k will not be relevant to us. And every component that does have exactly k vertices, well there we simply need to check if

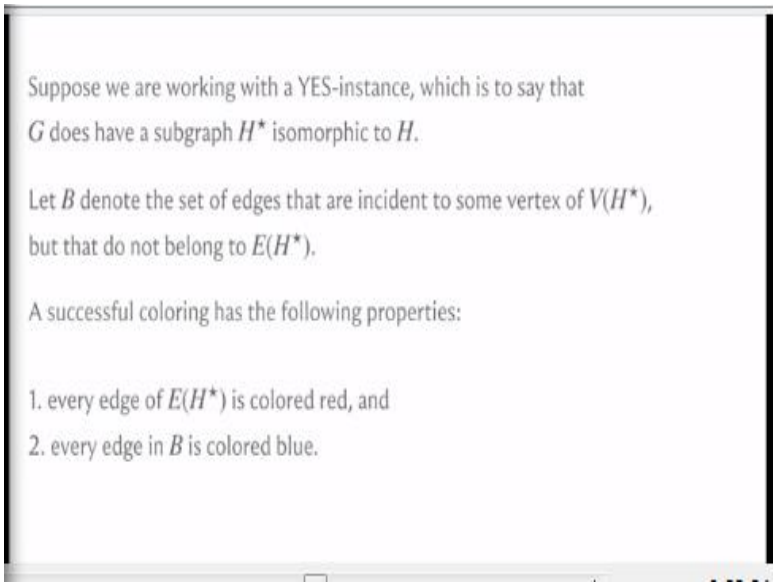
it is really a copy of H or not and that is where the k factorial in the running time was coming from.

(Refer Slide Time: 21:48)



So, let us go over all of this once again but just a little bit more properly and slowly.

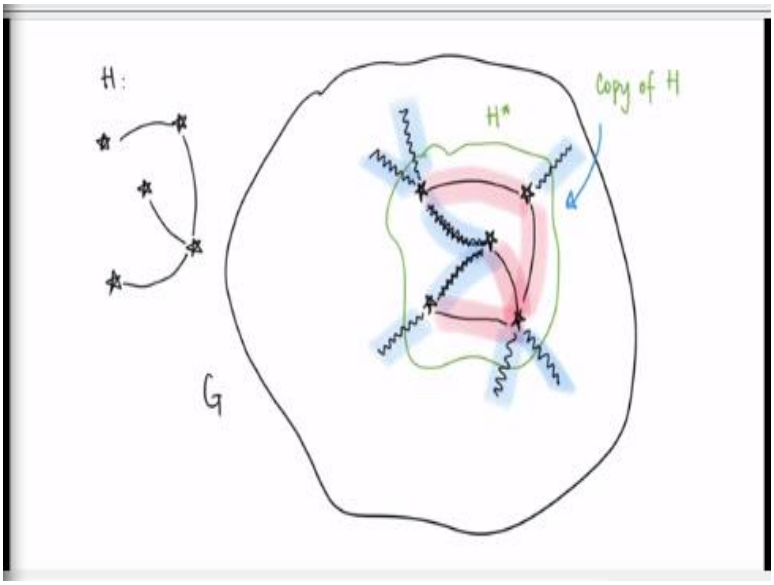
(Refer Slide Time: 21:52)



So, first let us look at the formal definition of a good colouring. So, suppose we are working with a YES instance, so we do have a copy of H in G which will call at star. And let B denote the set of edges which are these cross edges, so they have one end point in H star and the other endpoint outside of H star. So, now for us a good colouring is one which colours every edge inside at star

both end points in H star every such edge gets coloured red and every edge that is in B gets coloured blue.

(Refer Slide Time: 22:25)

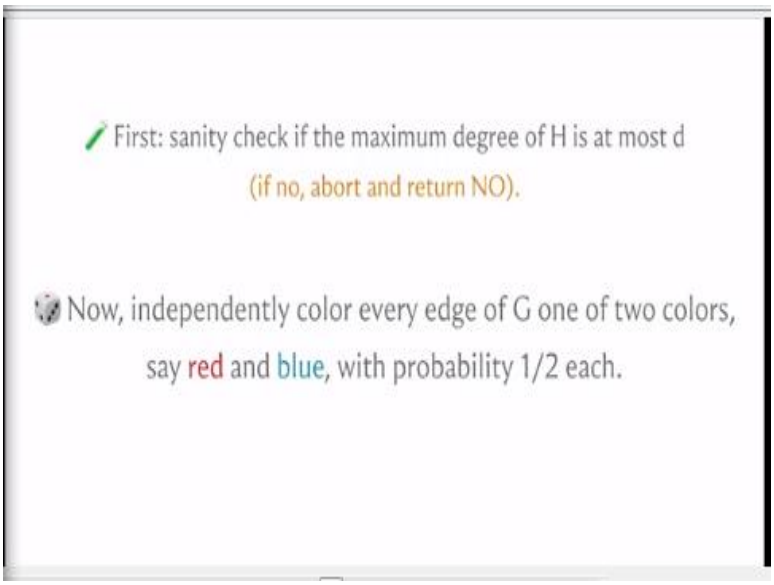


So, here is an example of what this would look like for a specific H and a specific copy of H inside G. So, now that we understand what we expect from a good colouring.

(Refer Slide Time: 22:38)

Let us talk about the random experiment that we hope will generate such a good colouring with a decent probability. It turns out that for this problem a random experiment is really simple.

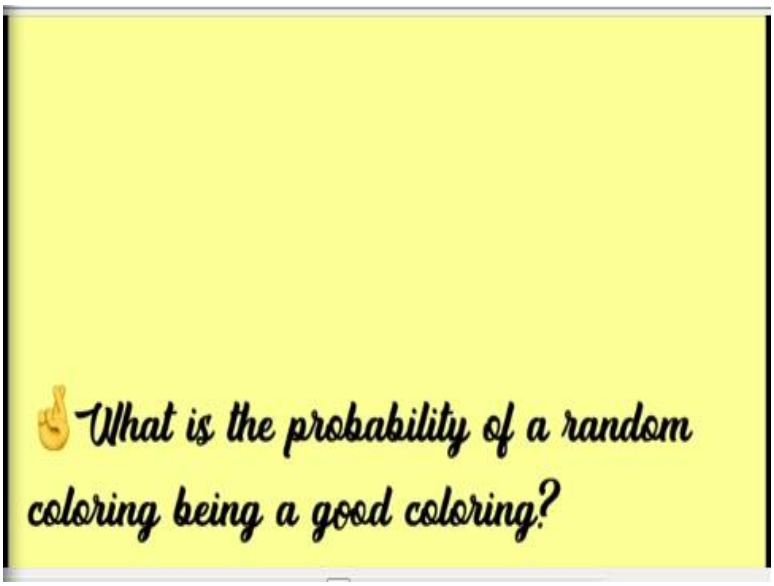
(Refer Slide Time: 22:51)



And to begin with we are just going to start with a straightforward sanity check; we do not have any fancy preprocessing or anything like that. But we simply check if G has bounded maximum degree or not. So, remember that we are parameterizing by the degree of the host graph G, so if G has some unusually high degree vertices more formally if G has a vertex whose degree exceeds our threshold d.

Then at that point we simply return NO, we have no obligation to solve this problem anymore. But once you pass the sanity check and you are working with an interesting instance, then we are going to essentially randomly colour every edge with one of the 2 colours red and blue uniformly at random. And these experiments are going to be independent across edges that is it. That is the entire random experiment it involves m coin flips where m is the number of edges.

(Refer Slide Time: 23:48)



And now the next natural question that we want to address is what is the probability that this random colouring is actually a good colouring for us and in this calculation that we will see the relevance of the host graph having bounded maximum degree. We will see how that helps us out.

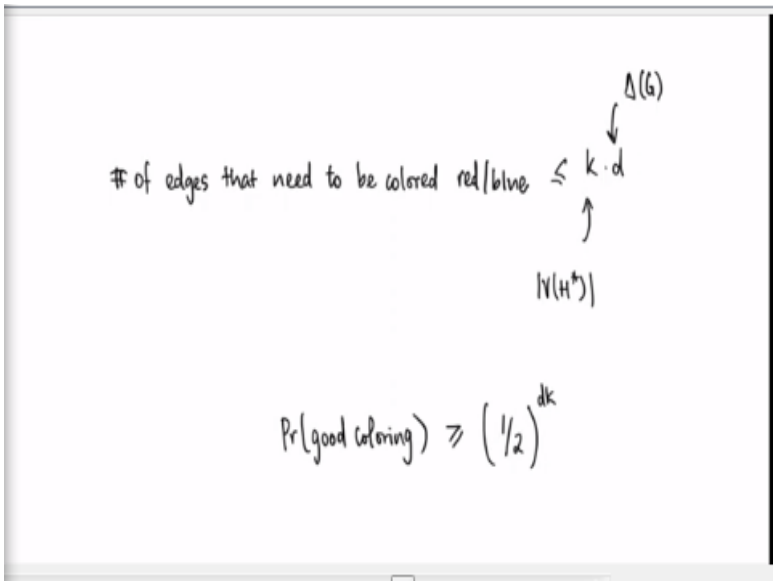
(Refer Slide Time: 24:05)

So, for this colouring to be a good colouring let us think about what are the edges that we should focus on. Remember that something should happen only if we are working with a YES instance. If you are working with a NO instance then we are anyway pretty safe because we are never going to manifest a H star out of thin air if it does not exist. So, then we are anyway going to say no.

But whenever there is a YES instance we want a certain subset of edges to be coloured in a very specific way. So, how many edges are there? What is the scope of the number of edges on which we are really keeping our fingers crossed and hoping that the random colouring does the right thing? If you need to just pause for a moment here and really think about this because this is going to determine the analysis of the success probability.

And it is not very hard to figure out, remember if you need just go back to the definition of what we needed from a good colouring? We needed the edges of H star to be coloured say red and we needed the edges of B which have one end point in H star and the other end point out of H star, we needed those to be coloured red. So, basically what is the probability with which this happens to understand this first you need to understand how many edges are even involved? So, please think about that and come back when you are ready.

(Refer Slide Time: 25:29)



So, notice that the number of edges that we really, care about is bounded by k times d. Why is that? Well, notice that H star has exactly k vertices and the number of edges we are worried about whether their edges with which land both of their endpoints within it is star or they have one endpoint in and one out. They are all accounted for if we just sum the degrees of all the vertices in H star.

In fact if we sum the degrees of all the vertices in H star, we actually over count potentially the number of edges we care about. Because those edges which have both of their endpoints in H star will get counted twice. But that does not really matter we are looking for an upper bound, so an overestimate is just a safe upper bound. And further the sum of all the degrees of course is bounded by the maximum degree times the number of vertices in H star.

The maximum degree of course is guaranteed to be at most d and the vertices in H star, the number of them is k, so that is why we have this bound here. And notice that each of this k d many edges at most k d many edges gets the right colour with probability half independently. So, the overall probability that the colouring does the right thing, the random colouring does the right thing on the subset of edges that we actually care about is at least 1 over 2 to the dk.

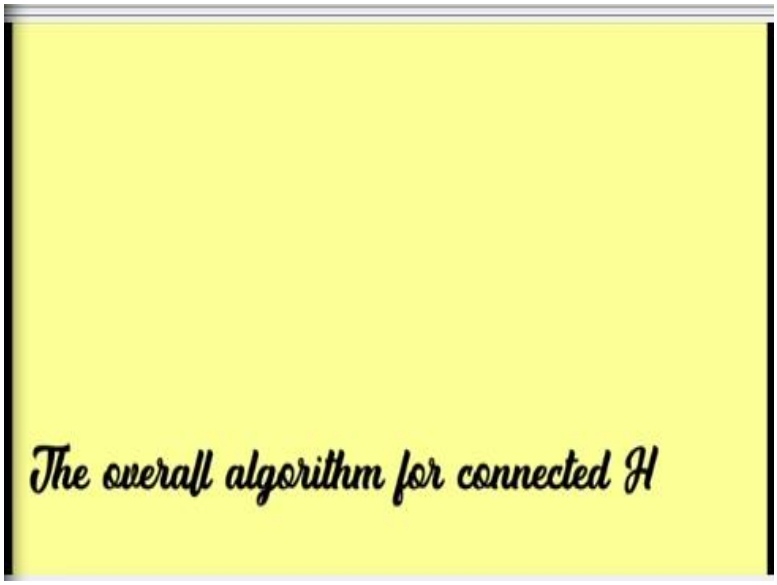
Of course the colouring does various things outside of this set of edges that we care about, but just like we have seen in the other lectures with colour coding and chromatic coding and so on, that does not really affect our success probability. So, whatever happens outside this is really the event that we care about. So, you could write out the probability more comprehensively but you will end up at the same lower bound on the success probability.

So, as usual you know from just staring at this expression that by applying our usual boosting argument by spending time proportional to 2 to the dk you can drive the success probability up to a constant, so you know we are pretty happy at this point. Now if you have some memory of the running time that we were claiming earlier it was 2 to the dk times k factorial times poly n.

So, we seem to have already accounted for the 2 to the dk term in that expression. And you might now be wondering where did the k factorial come from? Well, that is going to show up when we do the work that is required to actually find the pattern once we have a good colouring on our hands. So, remember so far we have just gotten to the point where we can be reasonably assured that if there is a copy of H and G, then there is a good colouring, the random colouring had the property that we wanted.

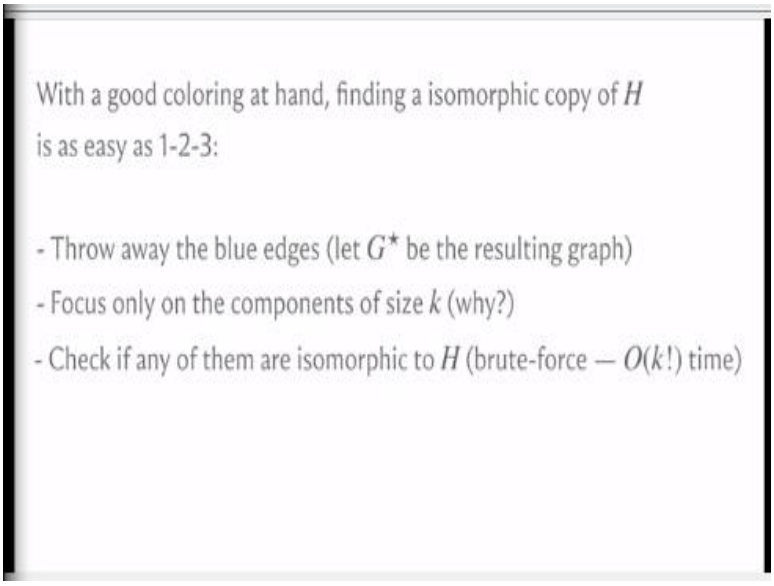
But moving on we have to now figure out that if we are given a graph and let us say even a promise that the edges have been coloured in such a way that it is a good colouring as you defined it a few minutes ago. Then can you actually find a copy of H assuming that it exists? So, this is the part that we need to discuss next.

(Refer Slide Time: 28:41)



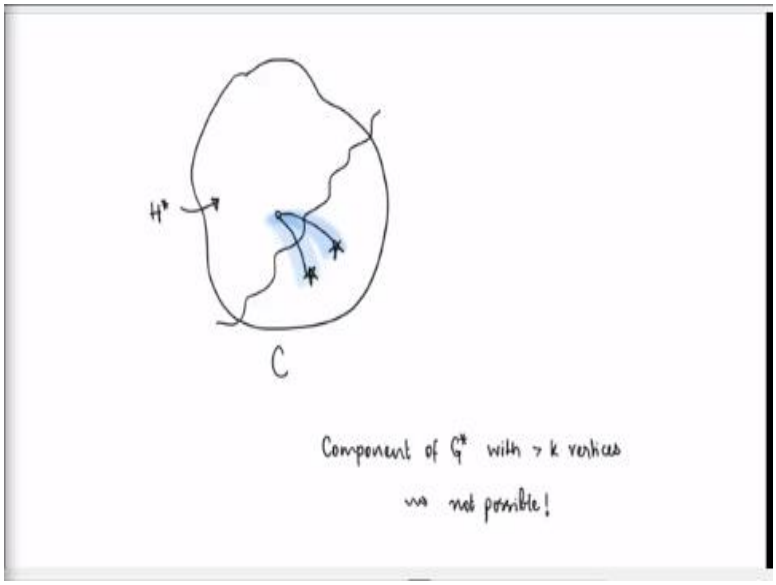
So, let us assume that the pattern H that we are looking for is connected and let us try to figure this out for this case. If you remember the physical analogy this is the part where we actually use the random colouring to tear up our paper into small shreds and then examine the shreds and try to match it with the pattern. So, let us actually do this.

(Refer Slide Time: 29:03)



So, given a good colouring it turns out that life is quite easy, so to begin with you can just throw away the blue edges, this is like tearing up the paper. And why can we throw away the blue edges? Well, we know that the blue edges never participate in the solution in H star, so we might as well delete them. But the crucial thing happens next which is that after deleting the blue edges when you look at the connected components that the graph disintegrates into. Then if H star did exist it is going to be exactly one of these components. So, in particular this means that we can only focus on components that have k vertices.

(Refer Slide Time: 29:43)



So, in particular just to make this really explicit, suppose you have a component that has let us say more than k vertices or you could also think about components that have less than k vertices and that is going to be a similar impossibility. But let us say that you have a component on more than k vertices. And is it possible that H star is hiding somewhere in here? Well, suppose H star is sitting inside a component that has more than k vertices.

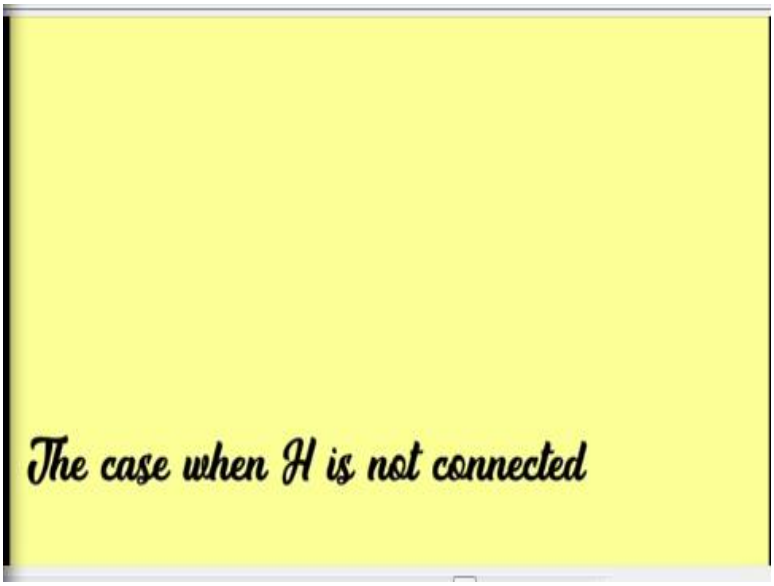
Because this is a connected component if you look at H star and if you look at the rest of the component there must be an edge that crosses this cut. But this edge by our assumption must have been a blue edge because remember we are working with a good colouring. So, any edge that has one of it is endpoints in H star and one outside it must have been coloured blue. But in our very first step we have deleted all the blue edges, so these edges should not have been here in the first place, so that is a contradiction.

And that is one way of formalizing this idea but at this point it should be fairly intuitive that your H star is going to be exactly one of the components once you have deleted all the blue edges. So, at this point you are practically done, right, you can simply shortlist all those connected components that have exactly k vertices. And for each one of them you can test if it is an isomorphic copy of H or not, this time simply by brute force.

Because your components are small, you can afford to try all possible ways of mapping the vertices of H to the vertices of the component that you are working with. That is essentially k factorial possibilities to explore and for each of those maps it is going to be a little bit of a polynomial overhead to verify if that map constitutes a valid isomorphism or not. So, this is k factorial poly n per component but again there are only at most n components, that is a very loose bound.

By the way since every component has k vertices the worst case you are dealing with perhaps something like n by k components. But in any case it is just overall a polynomial amount of work on top of k factorial which is really the standout dominant term in the running time for whatever we have done so far. So, hopefully this part is clear, what happens when H is a connected pattern? Now what happens if H is not a connected pattern?

(Refer Slide Time: 32:09)

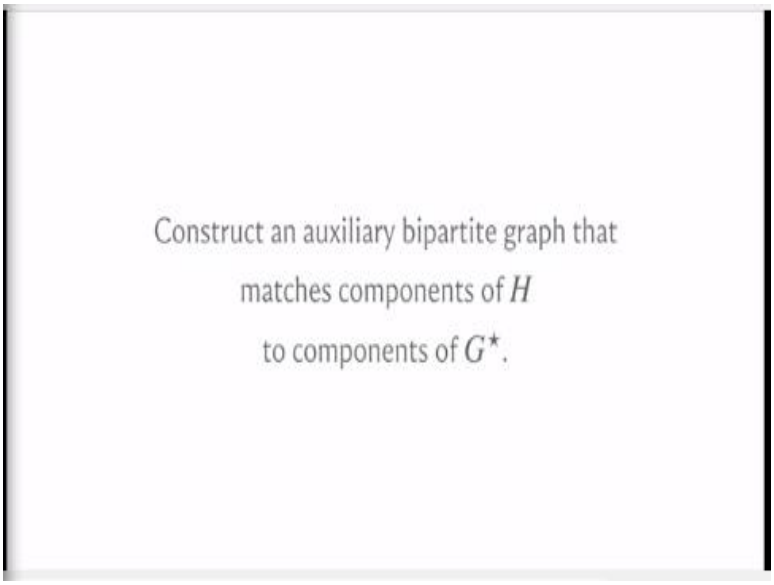


But let us say **it** itself is a bunch of smaller connected components. Knowing what you know by now you might want to take a pause and see if you can puzzle this out for yourself; it is not very difficult at all. It just needs a little bit of imagination for how you will deal with mapping these different components to the components that you have at your disposal from G once the blue edges are gone. So, feel free to take a pause here and again come back once you are ready.

So, as usual it really helps to just think about some small examples first. So, what is the smallest non trivial case of H being a pattern which is not connected? So, let us imagine that H is just 2 connected components, let us say with k 1 and k 2 vertices in them with k 1 + k 2 being equal to k. So, now probably the components that have exactly k vertices are no longer so interesting. But instead it is the components that have k 1 and k 2 vertices in G - B that are potentially things that the components of H can be mapped to.

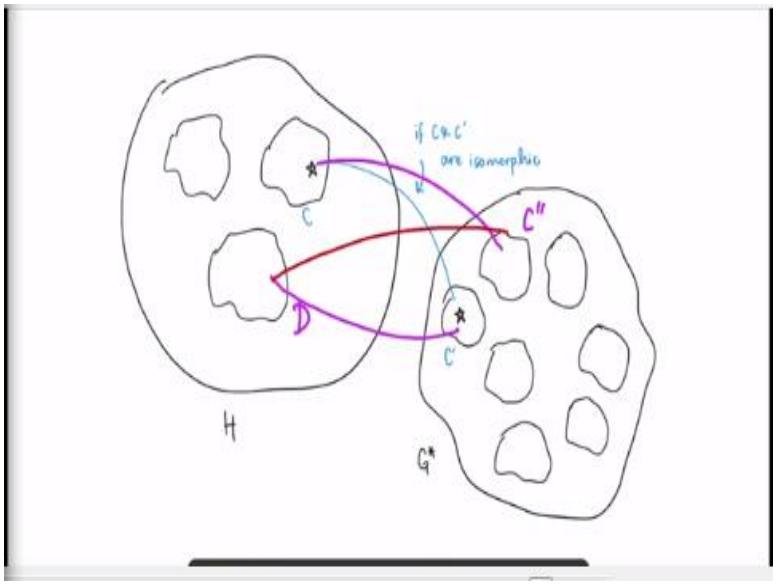
So, we want to experiment with trying to match the components that have k 1 vertices within them, with the first component in H the one that has k 1 vertices. And similarly those components of G - B that have k 2 vertices in them become candidates to be matched with that component in H which has k 2 vertices. So, to kind of generalize this 3rd process and to formalize the idea here.

(Refer Slide Time: 33:45)



Let us actually construct an auxiliary bipartite graph which is designed to match the components of edge to the components of G star. So, already I think the fact that I have used the word match so many times may have triggered the thought of probably literally using a matching algorithm. And if that occurred to you then that is great because that is exactly what we are going to do.

(Refer Slide Time: 34:11)



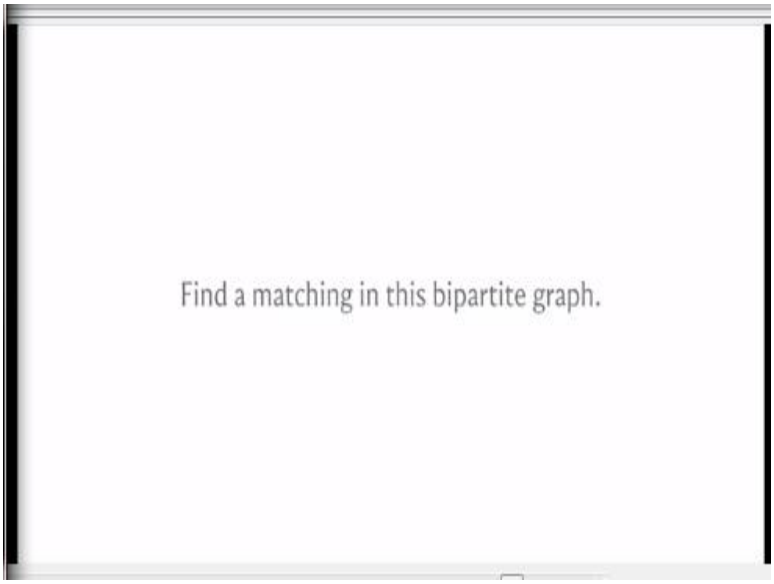
So, this bipartite graph is going to have of course 2 groups of vertices. On the one hand we will have vertices that represent the components of H and on the other hand we will have vertices that represent the components of G. So, we will actually I should not really be saying G, this is going to be G with the blue edges removed, so let us denote that by G star and so this is your auxiliary bipartite graph.

And when do we have an edge between a vertex representing a component of H and a vertex that represents a component of G star? Well, that is very natural, we are going to have that edge precisely when these 2 components are isomorphic copies of each other. So, this is something that again you can test by brute force. So, now you will be applying your k factorial subroutine as many times as there are potential matches to be carried out between the components of H and the components of G star.

So, if your bipartite graph has let us say p vertices on the left and q vertices on the right, then you will be running your k factorial subroutine at most p times q many times. Now once you have

built up this graph all you have to do is check if there is a matching that saturates all the vertices that represent the components of H. If you have such a matching then you are in business otherwise there is no isomorphic copy of H in G.

(Refer Slide Time: 35:40)



So, notice that everything that we have discussed so far for the last leg of this journey essentially boils down to finding a matching in this bipartite graph. And specifically you want to of course find a maximum matching and check if it saturates the side which represents the components of H. Now of course to find this maximum matching you can deploy your favorite maximum matching algorithm on bipartite graphs you could use flows, you could use augmenting baths, you could use whatever you like.

But it turns out that this specific bipartite graph here has a very special structure which makes your job even easier. So, if you think about it a bit you will perhaps discover that this bipartite graph actually has components that are complete bipartite subgraphs. And the reason for that is essentially the transitivity of isomorphism. So, for example let us imagine that the component C here has edges to C prime and C double prime and let us say the component D has an edge to C prime.

Notice that it is not possible for D to not be adjacent to C double prime because if D is isomorphic to C prime and C prime is the same as C and C is the same as C double prime. Then

of course these things are essentially all the same, so D must also be isomorphic to C double prime as well. So, you can essentially build this argument out further and formally demonstrate that every component of this bipartite graph is in fact a complete bipartite graph.

And then essentially the check for whether there is a matching just boils down to comparing the sizes of the left and right sides of these components, it is a very simple thing to go ahead and check. So, from an implementation point of view it again becomes a really simple algorithm. So, with that we have come to the end of our exploration of subgraph isomorphism.

(Refer Slide Time: 37:34)

We discussed a randomized algorithm with one sided error as usual with only false negatives which figured out if G has a copy of a pattern H. The running time of this algorithm this is something that is important to remember is FPT in k + D, where k is the size of the pattern graph and D is the maximum degree of the host graph in which you are looking for this pattern. So, with that we come to an end of our discussion of random separation and subgraph isomorphism.

And in fact with this we even wrap up our discussion of randomized methods in parameterized algorithms at large. These were all the algorithms that we wanted to discuss this week. So, by now we have seen a very simple randomized algorithm for FVS that just involved picking a random edge and a random endpoint after some pre processing. We have talked about colour coding and chromatic coding which are by now classic techniques in terms of finding patterns or

finding objects and really making them stand out by randomly partitioning objects in the graph in clever ways.

And with random separation we again looked at a technique that was very much in that spirit but it was done in a slightly different way. So, I hope you have enjoyed this little tour of some really cool ideas, I think a common theme for all of these algorithms is that they are really elegant to describe, they are simple to implement and the analysis is usually non trivial and interesting as to why the success probabilities work out the way that they do.

Now the one last thing that we want to talk about very briefly is the issue of derandomization. So, if that tiny, error probability is really nagging you, if you are feeling paranoid about it or maybe you are thinking about critical applications where we really cannot afford to go wrong. Then you want to stay tuned for the final lecture our last discussion for this week which will be about the randomization. So, we will see you then, thanks for watching as always and bye for now.