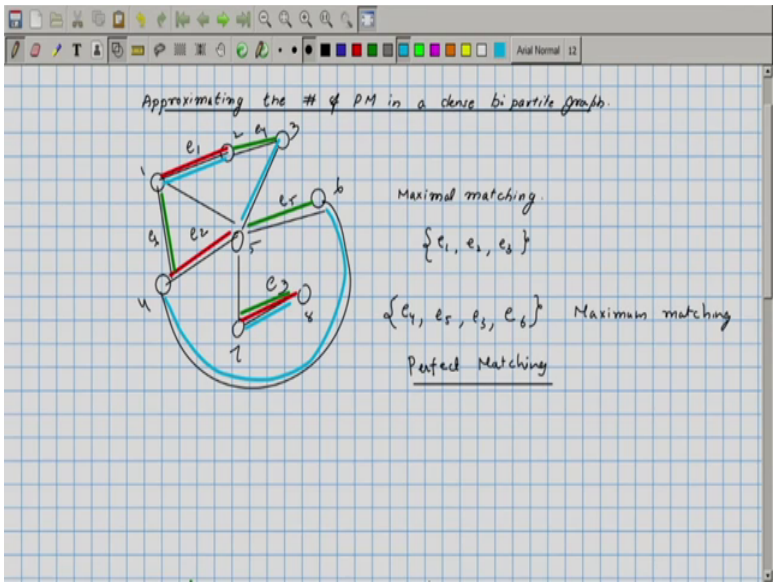
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 32
Perfect Matching – II

The problem that we are interested in today and the next couple of lectures is approximating the number of Perfect Matchings in a dense bipartite graph.

(Refer Slide Time: 00:37)



So, each of these terms, bipartite dense and approximating in some sense presents some of the limitations that we have in counting the number of perfect matchings in an arbitrary graph, ok. So, in order to overcome that we are restricting it to bipartite and instead of exactly counting, we are approximating.

So, what is the problem? So, suppose we are given a general graph ok. So, this is a graph on 8 vertices ok. Now the red edges that I have drawn here, there is a matching. If you look at these edges, they do not share any vertices; e 1, e 2 and e 3. They connect distinct pair of vertices. Now there are two vertices which are not connected and there is no way we can connect. So, this is an example of what is called as a maximal magic. And different number the vertices 1, 2, 3, 4, 5, 6, 7, 8 then, e 1, e 2, e 3 is a matching and it is a maximal matching. In the sense, if you add any other edge of the graph to this set, then that is going to share a vertex with one of the existing edges.

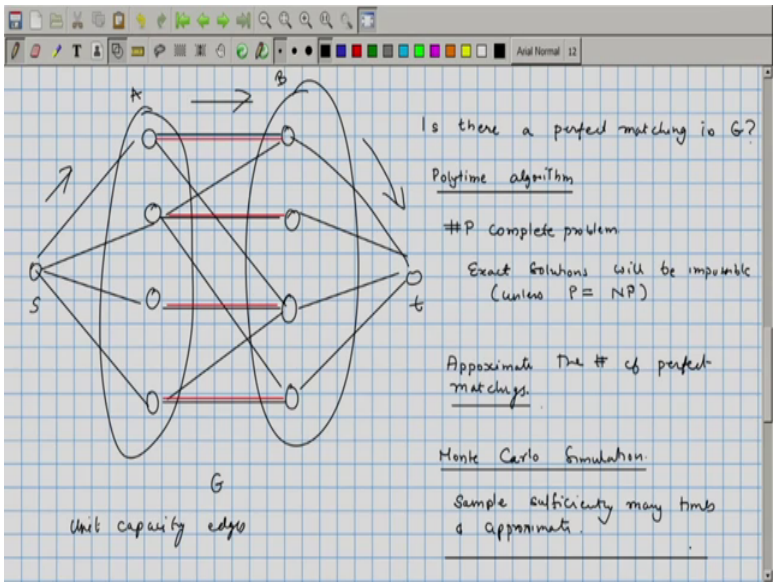
So, this is a maximal matching, but of course, there are other matchings which are larger in size. This is the matching of size 3; we could have the following matching. If we had taken the green edges, then we get what is called as a maximum matching and in this case, it is also a perfect matching ok. So, if I number these vertices or named these vertices as e 4, e 5 and the e 6; e 4, e 5, e 3, e 6 forms a maximum matching.

This is a matching which cannot be further extended. It is also a perfect matching. It matches every, you can think of it as each vertex is matched to another unique vertex by means of an edge; and it match this is a matching which matches all the vertices. So, since there are only 8 vertices in this graph, you cannot have a matching of size more than 4 and here you found one. So, this is a maximum matching and it is a perfect matching. It matches every vertex.

So, the general problem that we are interested in this given a graph can be find if is there a perfect matching. Our algorithms which will do this, we also want to count the number of perfect matchings that are there in this graph. So, the green edges denote one particular matching that is a perfect matching. If we had taken let us say these edges inside the blue edges that is another example of perfect matching. So, there are at least 2 perfect matchings are there more what is the number of perfect matchings in this graph. That is a question that we want to solve. It is generally a difficult question counting the number.

So, we will restrict our attention to bipartite graphs ok. So let us say we are given a bipartite graph.

(Refer Slide Time: 05:15)



So this is an example of a bipartite graph. So, each sides there are 8 vertices in this graph and you can divide the graph into 2 parts. Let us say A and B says that all the edges are from a to b or an or between a and b. This is an undirected bipartite graph. So, between the vertices in A; there is no word, no edge and between vertices in B also there are no edges. So, are there perfect matchings in this particular graph of course, there is if we look at these edges this is a perfect matching. There could be others as well ok.

So, probably there are others, but um. So, you need to count the number of perfect matchings in a given bipartite graph. Now first, we can ask ourselves this question. Is there a perfect matching in G? Could be any arbitrary graph; arbitrary bipartite graph. Now this question has a polynomial time algorithm, simple algorithm. We can just reformulate, it as a network flow problem. So let us say, there we add 2 vertices S and t and then connect all the vertices from A to S and all the vertices from B to t ok. So, this becomes a network and we can imagine that the vertices are all unit weight, unit capacity edges and the direction is from S to A, A to B and B to t.

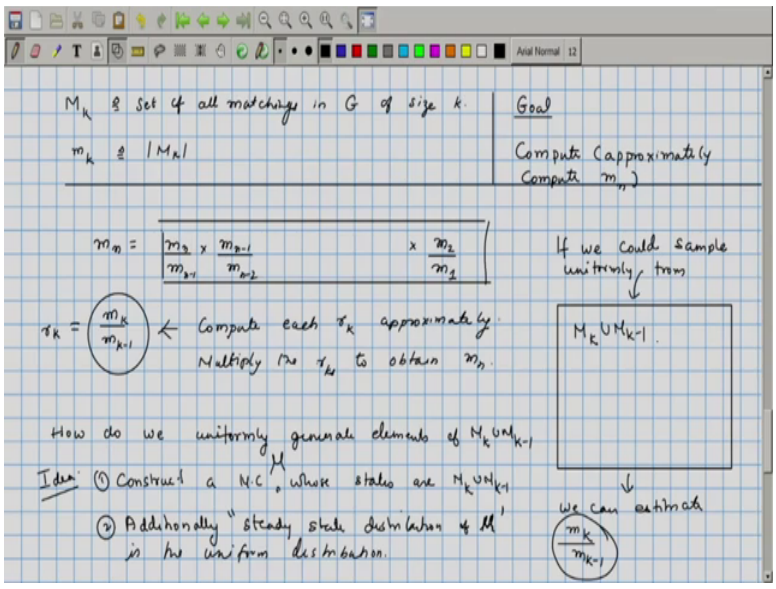
So, we can convert the input bipartite graph into a capacitated network and we can assign unit weight capacity to each edge and then we can ask what is the maximum flow from S to t. The maximum flow is equal to the number of vertices in A and it should be equal to the number of vertices in B, then we can say that that will be a perfect matching. In fact, the flow would even help us find their perfect matching ok. So, there is a polynomial

time algorithm. So, we might expect that since there is a polynomial time algorithm B even counting the number of such may be easy, but it is known that this is a sharp P complete problem.

So we do not expect that this bipartite matching problem can be solved exactly. So exact solutions will be impossible. So exact solutions which work in polynomial time will be impossible unless P is equal to NP. So under reasonably robust assumptions, we can say that this problem cannot have a polynomial time solution. So we look at can we approximate the number of number of perfect matchings. So how do we do this? So, one thing that we will use is what is called as a Monte Carlo simulation which essentially is sample sufficiently many times and approximate and whatever; that means,.

So, the idea is construct a sample space whose size we know. Then we keep on generating samples of perfect matchings ok. So, keep on continuously sampling and we check how many of them are perfect matchings ok. So, the sample space will consist of all the perfect matchings and if we and there are other elements as well. So, by continuously sampling, we can estimate the number of perfect matchings. The, I mean the ratio of the number of perfect matchings to the total elements in the set. That is the basic idea behind Monte Carlo simulation, but we will do a little more sophisticated sampling ok.

(Refer Slide Time: 11:03)



So, we need to have some basic terminology. So, let us say M k denotes the set of all matchings in G of size k and small m k will denote the size of the set. How many such machines are there? What we are interested in is compute our goal is to compute or approximately compute small m n ok. So, we can write. So, our algorithm is based on the following observation m n can be written as m n divided by m n minus 1 into m n minus 1 divided by m n minus 2 into m 2 divided by m 1. So, this formula that, we will use to evaluate m n.

Now, each of the terms is of the form m k by m k minus 1. Now how do we; so if you can evaluate each such term. So, let us call this as equal to r k. Compute each r k approximately and then multiply the r k's to obtain m n ok. So, that is the outline of the algorithm, but how do you compute each r k approximately? Ok. So, what does each r k? It is a ratio of the number of matchings of size k to the number of matchings of size k minus 1.

So, suppose we had a bag or a box containing all elements of m k union m k minus 1 ok. So, suppose we can sample, if we could sample uniformly from the set then, we can estimate m k divided by m k minus 1 ok. But how do we uniformly sample from this particular sample space ? So, one way is to construct a Markov chain. So, so how do we uniformly generate elements of m k union m k minus 1? So the idea is, construct a Markov chain whose states are m k union mk plus 1 ok.

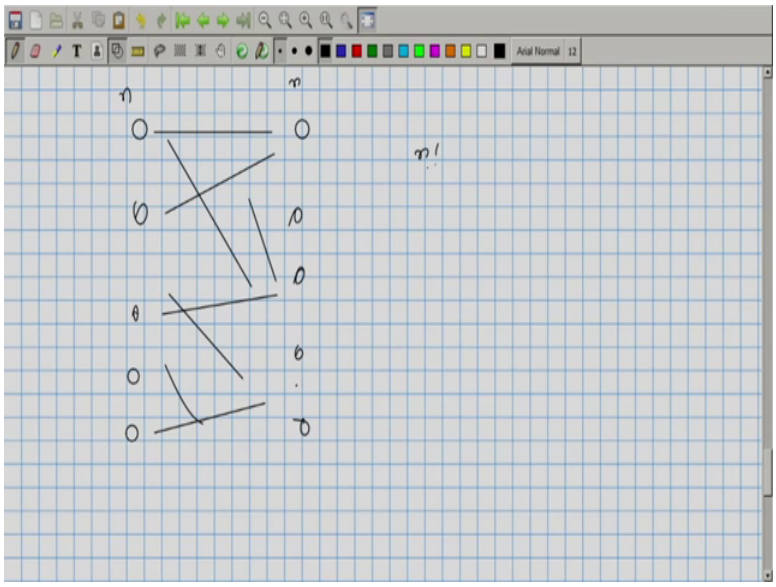
And then, I mean additionally, we will require that the steady state distribution of let us say m is the uniform distribution. This is the key thing that we will use. So, we will generate the Markov chain whose states are each a matching either of size m k or of size m k minus 1 ok.

Now if you have a Markov chain whose states are these, if you run the Markov chain or a similarly the Markov chain for enough number of times we need to bother about what is enough number of times. The Markov chain, if it is a suitably constructed Markov chain, it will approach its steady state or stationary distribution. Now once it reaches its stationary distribution, the individual states will appear with probability corresponding to the steady state distribution. So if the steady state distribution was a uniform distribution, then we can say that the states will appear with uniform probability.

So, if we could construct such a thing then it means we could sampled m uniformly from m k n m k minus 1 and from that we can estimate the ratio. There could be error in that, and when we multiply many such terms each of which which have an error we will have to work out we will have to do the algebra and show that these errors do not compound to give some unacceptable amount of error. So, all those things needs to be done, but at an idea level this is what is happening. We are going to construct a Markov chain whose states are going to be the matchings of size either k or k minus 1. We will do this for all k varying from let us say 2 to n.

Now, what do we mean by constructing a Markov chain? It means, a Markov chain you can think of it as a graph, but this graph could be very large. We want all our computations to be done in polynomial time.

(Refer Slide Time: 17:37)

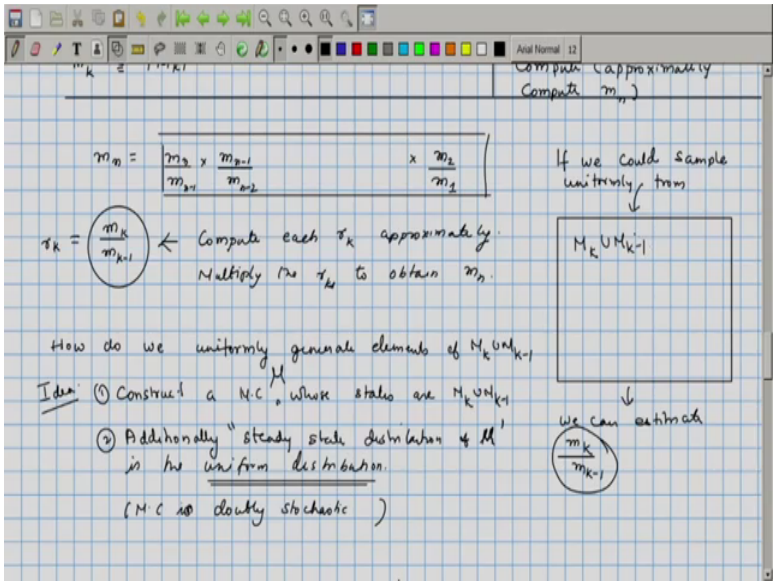


This graph for example, if our initial let us say; bipartite graph was a very large and very dense graph with all mean let us say if it was a complete bipartite graph ok. There are n vertices on either sides; each n factorial permutations. So correspondingly, every permutation you have a matching. So, the number of matchings of size n is very large and we cannot really construct that graph and try to uniformly sample because that construction itself is going to take that graph. The intermediary Markov chain that you construct is going to be huge object. So, it defeats the whole purpose.

So here, when we say we construct a Markov chain, it the the graph behind it could be large, but we could locally compute. Which means if you are mean if you think of the Markov chain as a random walk on a finite graph, the finite graph could be exponentially large. When you are at a particular state you can compute the neighboring states and you can choose one of them with equal probability ok. If you could construct a Markov chain such that, locally you can simulate the Markov chain, then that is good enough. You do not have to compute the whole Markov chain and keep it with you ok. Instead, we need to have the ability to simulate the random walk on the Markov chain. That amounts to having the ability to locally compute the next step for each at each stage ok. So, how do we construct such Markov chain?

Now one thing that we will I mean in order to ensure uniform distribution this thing is fairly easy because all you have to do is ensure that the Markov chain is doubly stochastic ok.

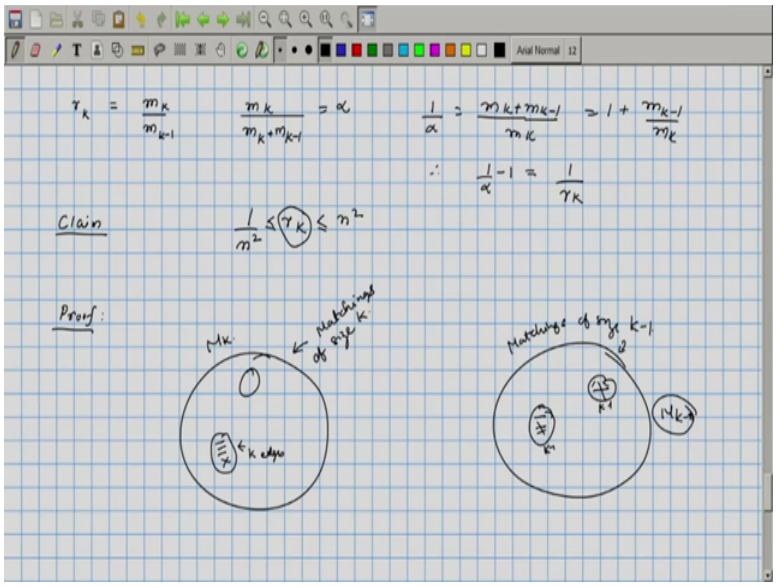
(Refer Slide Time: 19:27)



If you ensure that it is doubly stochastic then automatically the only steady state distribution would be the uniform distribution ok. So, we will use that later on. So, I mean, what we will bother about is constructing the Markov chain. By construction, it will be a symmetric Markov chain. The transition probability matrix will be symmetric and when it is symmetric the columns and the rows both sum to 1 because column, the rows anyway sum to 1. By symmetry, the columns will also sum to 1. So, these kinds of

matrices are called as doubly stochastic matrices and for doubly stochastic matrices, under reasonable assumptions we can say that the only steady state distribution will be the uniform distribution. So we will try to construct a random work which is which is symmetric ok.

(Refer Slide Time: 20:41)



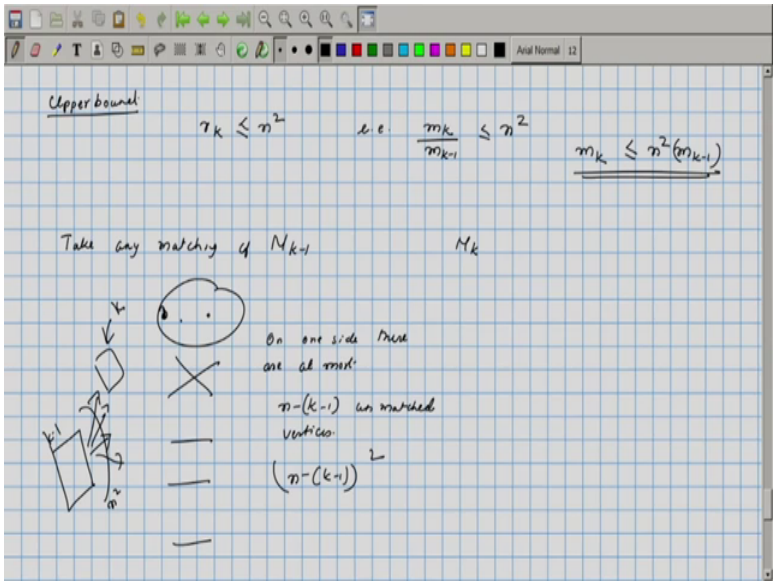
Now, how do we in our objective is to compute r k which is equal to m k by m k minus 1? Ok now, this is sufficient if we compute let us say, m k by m k plus m k minus 1 because I mean. So it is not equal. So, if we compute this quantity from that we can infer what is a m k by, so this is alpha then, 1 by alpha is equal to m k by by m k. This is equal to 1 plus m k minus 1 by m k. So 1 by alpha minus 1 is equal to 1 by r k ok.

Now, in order to determine these things, we can again do Monte Carlo simulation. When the reason why we focused on these r k is I mean, if we could uniformly generate matchings of size k and k minus 1 then, the ratios could be generated using Monte Carlo simulations, but if we wanted to do the Monte Carlo the method then, these r k; I mean m k and m k minus 1 should have I mean they should be reasonably close to each other ok. So, so claim. So we will try and prove this claim r k lies between n square and 1 by n square ok. So, if what is the use of this claim? Now if this claim is true then, our Monte Carlo method that is generate enough number of samples and then compute the ratios of the samples which belong to m k and the samples which belong to m k minus 1 that gives r k ok. So that, I mean that gives an estimate for r k.

So, in order to ensure that certain algorithm gives good bounds I mean, we require that r k is reasonably mean it is not very small or very large ok. So, we will show that r k will lie between n square and 1 by n square. How this helps us, we will see later ok. Now look at m k. These are matchings of size k. Now how can we associate these things with matchings of size k minus 1. That is, this is our m k minus 1. So, each of this is a matching. So let us say some collection of edges and the number of such things are k. So, k edges are there and each of these things would contain k minus 1 edges ok.

So what we will do is, we will show that every matching in m k minus 1 can be obtained by I mean from a matching of m k in a certain way and we will also show that every matching of m k can be obtained from an m k minus 1 match in a certain way. Ok and we will count the total number of ways in which we will do that and that will be our proof. So, r k less than n square we will call that as the upper bound.

(Refer Slide Time: 24:59)



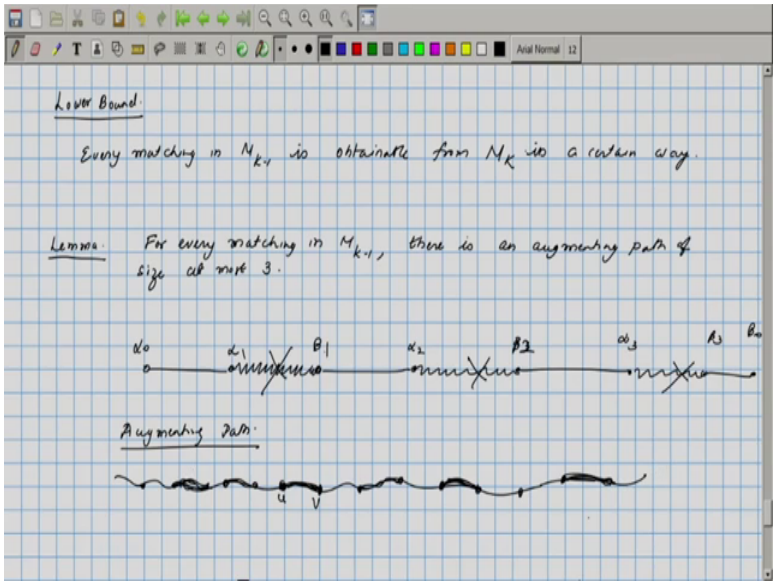
So, let us first try and see the upper bound. So, upper bound says r k is less than or equal to n square ok. So, that is m k divided by m k minus 1 is less than n square. So, take any matching of of m k minus 1 ok. How do you extend it to m k?

So, there are k minus 1 edges. There are n minus k vertices. So let us suppose each side contained n is the number of the maximum size of the matching. So n minus k minus 1 edges have already been matched and the unmatched vertices they are going to be 2 times n minus k minus 1. So, if you look at just one side, there are; so on one side there

are at most n minus k minus 1 unmatched vertices. Now each of those vertices could be matched to one of the n minus k minus 1 vertex on the other side. So in all, each I mean let us say if you take a specific element of m k. So let us say this is one particular matching, this matching could be extended to let us say n minus k minus 1 whole square other matchings ok. These this is a matching of size k, this is of size k minus 1.

So, total number of extensions possible is at most n minus k minus 1 whole square which is n square. So, each matching from m k minus 1; capital m k minus 1 may be extended to another matching of size k. Now in fact, every matching in m k, if you remove an edge from it you are going to get a matching in m k minus 1. So, every matching is obtainable in that form and therefore, if you look at each individual matching in m k minus 1 and try all possible extensions for all of them, we will get the complete collection m k ok. So, we know that m k is going to be less than or equal to n square times m k minus 1 ok. So, this is basically the upper bound. How do we do the lower bound ok?

(Refer Slide Time: 27:53)



So we need to claim that every matching in m k minus 1 is obtainable from m k in a certain way and we will count all those possible ways, and that will give us the bound for m k minus 1 ok. So first of all, let us prove following lemma for every matching in m k minus 1, there is an augmenting path of size at most 3 ok. So typically, when you take a particular matching and if you want to make it a bigger matching, what we will do is find an augmenting path ok. So let us understand what is an augmenting path?

So, let us say this was a matching edge and just drawing it by wiggly lines ok. So, let us say this connects alpha and beta and let us say alpha 1 beta 1 ok. Then there was another matching edge between alpha 2 and beta 2 and there was another matching edge between alpha alpha 3 and beta 3. Now suppose, these had the property that I mean by virtue of this being matching, these edges had to be disconnected. But if these vertices say alpha 0 and beta 0 and these are vertices such that these edges were present in the graph ok. If the original graph contained alpha 0 alpha 1 edge, beta 1 alpha 1 edge, beta 2 alpha 3 edge and beta 3 beta 0 edge then, we could throw out these edges from our matching and include the other edges.

So, we do not want any matching edge start either at alpha 0 or beta 0 ok. So, such a path in the original graph is called as an augmenting path ok. So, we can just simply think of an augmenting path as a path in the original graph. So let us say, this is a path in the original graph such that, if you look at vertices which are adjacent to each other ok. So, this is u v and if this is an edge ,this is a matching edge then the next edge is going to be a edge which I mean; the next edge is not going to be present in the matching. So, an augmenting path is a path such that if you look at the alternate edges, they are all matching edges. Clearly by definition of matching, there cannot be adjacent edges which are matching edge, but they could have been let us say separated by length more than 1 and if you take an arbitrary path maybe this is an edge and this is a matching edge, but these two are not present ok. This could happen in an arbitrary path, but that we will not call it as an augmenting path.

So an augmenting path is a path in which the edges alternate between matching edges and unmatched edges. Further, we require the starting edge and the ending edge to be non-matching edge. If you have such a path, what you can do is you can I mean if you have such a maximal path it means further this cannot be extended further. I mean there are no, I mean if you look at any outgoing edge there are no matching edges appearing there ok. So, that is called as an augmenting path. If you flip suitably then the augmenting path will increase the matching size by 1 ok. So this lemma states that, for every matching in m k there is an augmenting path of size at most 3. We will prove this lemma.