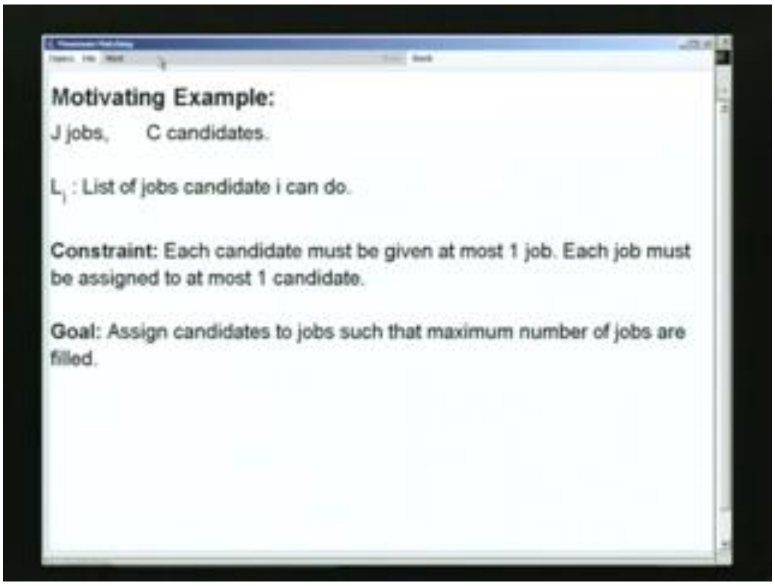
Design and Analysis of Algorithms
Prof. Abhiram Ranade
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 23
Bipartite Maximum Matching

Bipartite Graph Matching in the course on Design and Analysis of Algorithms. Let me begin with a motivating example.

(Refer Slide Time: 01:01)



Motivating Example:

J jobs, C candidates.

L_i : List of jobs candidate i can do.

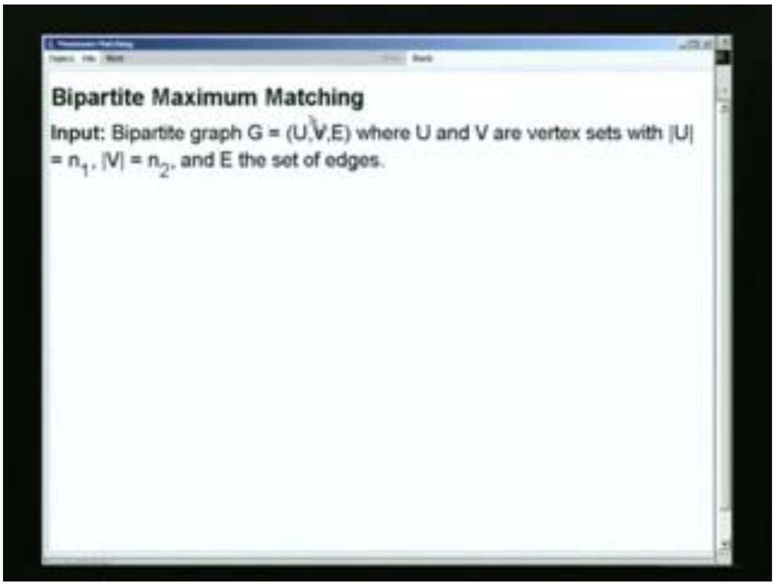
Constraint: Each candidate must be given at most 1 job. Each job must be assigned to at most 1 candidate.

Goal: Assign candidates to jobs such that maximum number of jobs are filled.

So, in this example we have some j jobs and some C candidates. The idea is that the each candidate we have given we have been given the list of jobs that they candidate can do. And we also have been given a constraint. The constraint says that, each candidate must be given at most one job. And also each job must be assigned to at most one candidate. The goal is our obvious goal.

Assign candidates to jobs such that, maximum number of jobs are filled. The bipartite maximum matching problem is exactly this, in more mathematical terms. So, I am going to first define this problem. And then, I will relate it to this job and candidate problem.

(Refer Slide Time: 01:59)

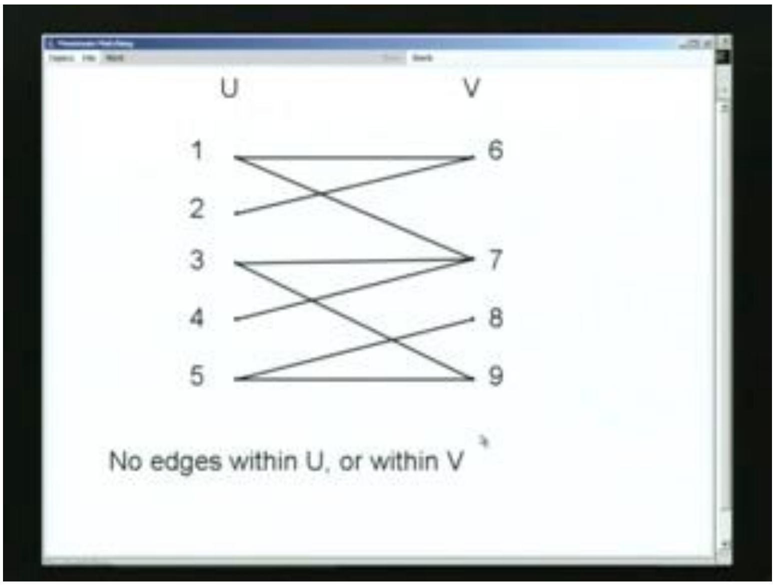


Bipartite Maximum Matching

Input: Bipartite graph G = (U,V,E) where U and V are vertex sets with |U| = n_1, |V| = n_2, and E the set of edges.

In this problem, our input is a bipartite graph. So, let us call it G. G is composed of two vertex sets U and V. And the cardinality of U says n 1. And the cardinality of V is n 2 and E is the set of edges. Let me remind you, what a bipartite graph is. A bipartite graph is simply the graph, in which the vertex set is in two parts U and V. And the edges only go between U and V.

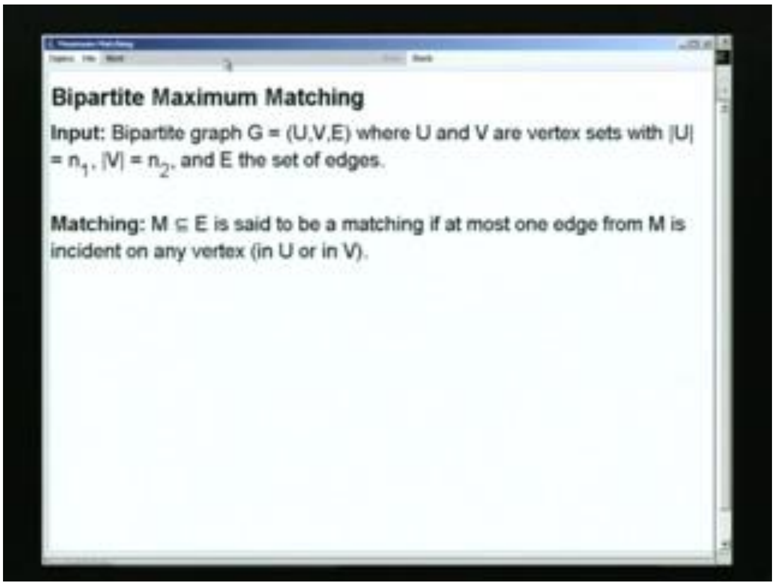
(Refer Slide Time: 02:31)



No edges within U, or within V

Here for example, is a bipartite graph. So, this forms this set of vertices forms U. This set of vertices forms V. And as you can see the edges, only go from some vertex in V to some vertex in U to some vertex in V. Specifically, there are no edges which connect our text in U to another vertex in U or our text is V to another vertex in V.

(Refer Slide Time: 03:00)



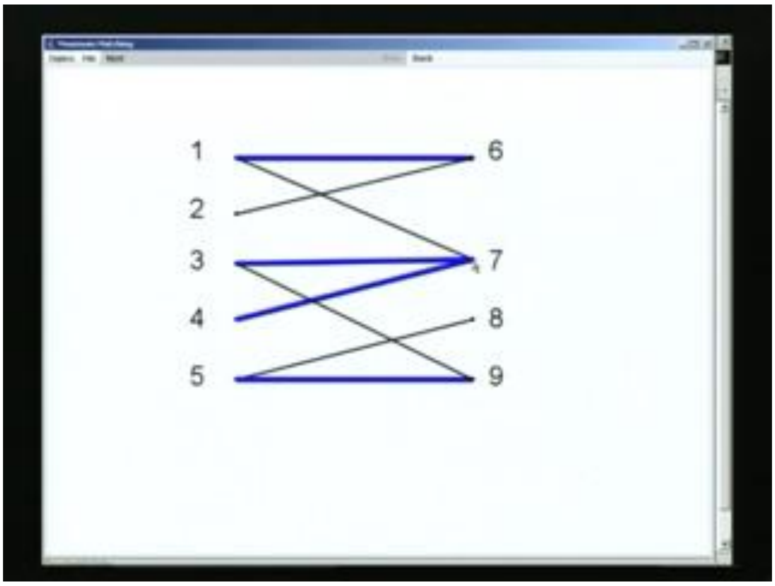
Bipartite Maximum Matching

Input: Bipartite graph G = (U,V,E) where U and V are vertex sets with $|U| = n_1$, $|V| = n_2$, and E the set of edges.

Matching: $M \subseteq E$ is said to be a matching if at most one edge from M is incident on any vertex (in U or in V).

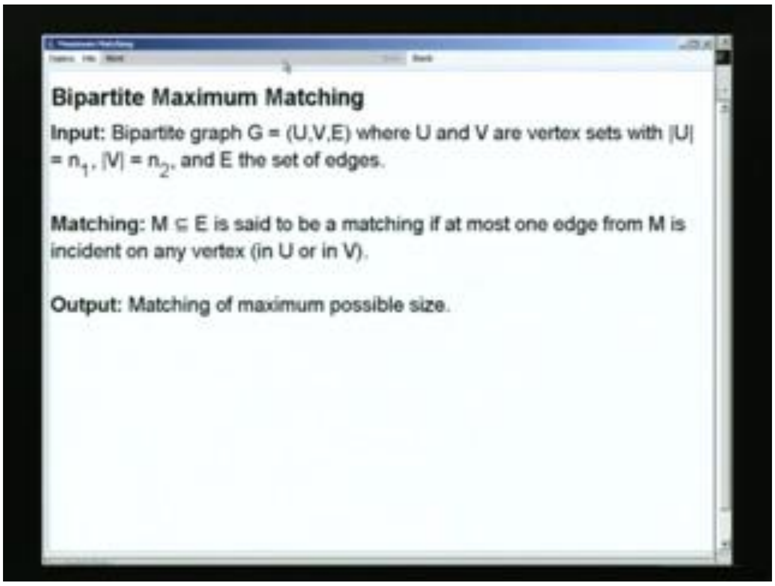
So, let me now define what a matching is. A matching is a subset of the edges such that, at most one edge is incident on any vertex either in U or in V.

(Refer Slide Time: 03:15)



So, in this graph for example, the set of blue edges would be a matching. If I add one more blue edge as here, this would not be a matching, because we would have a conflict at this vertex 7. So, here we would have been having two edges of the matching, of the so called matching being incident and that is not allowed.

(Refer Slide Time: 03:38)



Bipartite Maximum Matching

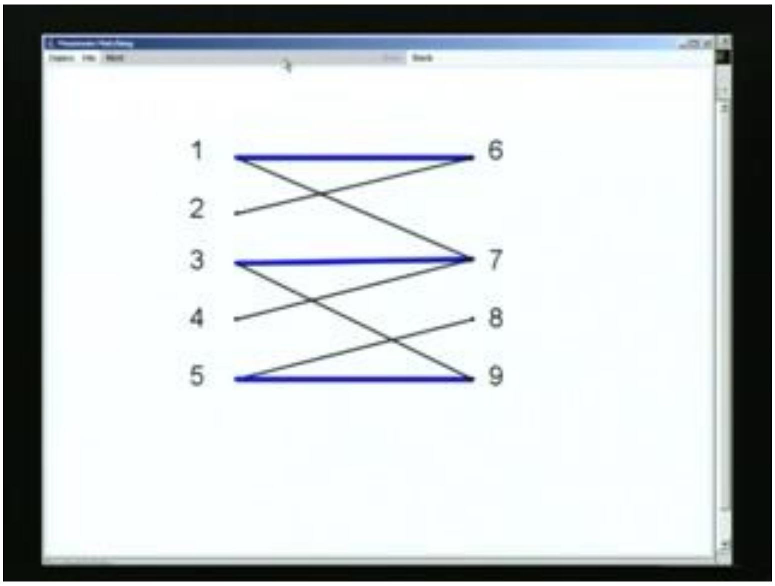
Input: Bipartite graph G = (U,V,E) where U and V are vertex sets with |U| = n_1, |V| = n_2, and E the set of edges.

Matching: M ⊆ E is said to be a matching if at most one edge from M is incident on any vertex (in U or in V).

Output: Matching of maximum possible size.

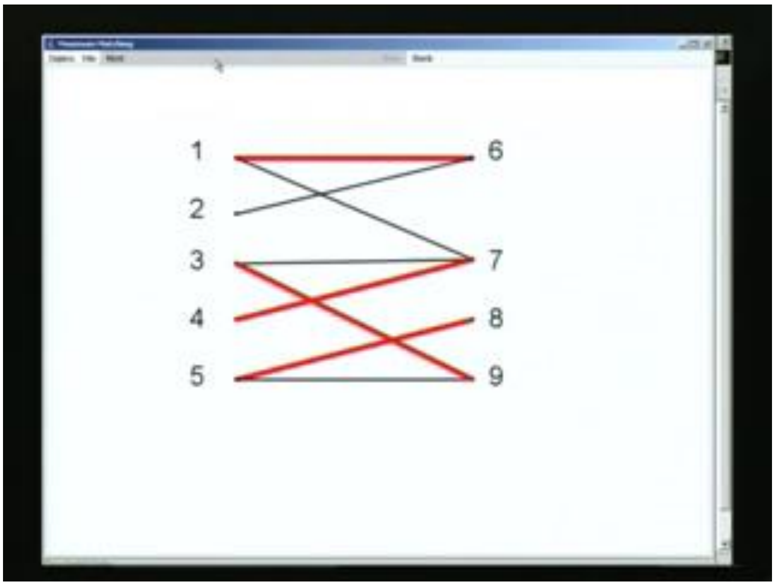
So, now let me tell you, what the goal in this problem is or what we require to output. So, we are required to output a matching of maximum possible size. A size of the matching is defined as the number of edges in it.

(Refer Slide Time: 03:54)



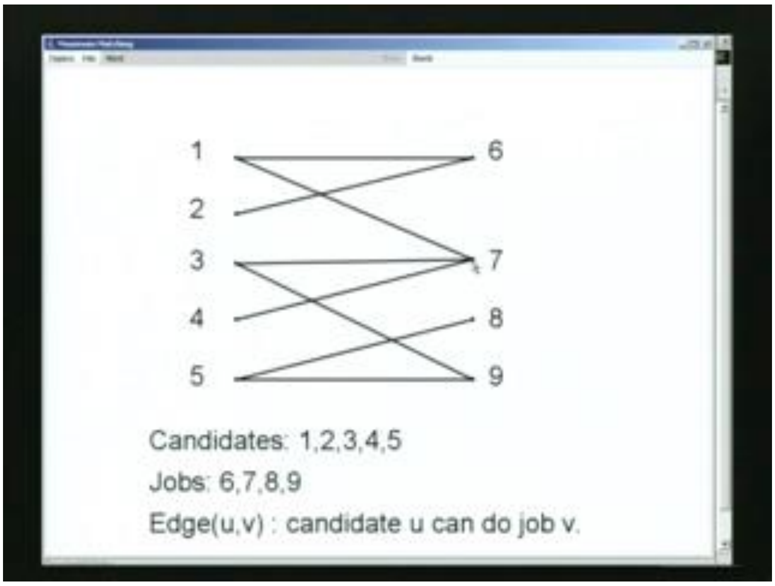
In this graph for example, we have this blue matching which consists of three edges. So, its size is 3. This however, is not the maximum sized matching.

(Refer Slide Time: 04:06)



This red matching for example is the maximum size matching. And in fact, it contains 1 2 and 3 and 4 edges. Maximum matchings are not unique, in this graph itself. For example, this is another maximum matching.

(Refer Slide Time: 04:25)



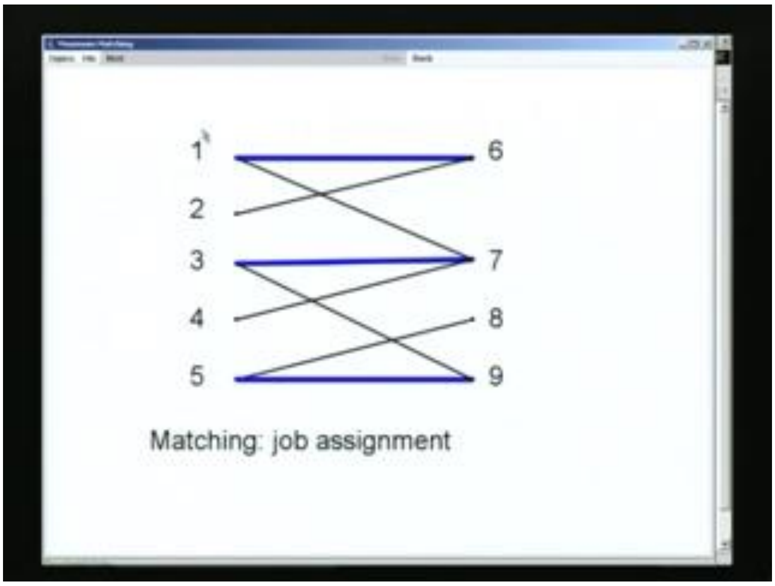
Candidates: 1,2,3,4,5

Jobs: 6,7,8,9

Edge(u,v) : candidate u can do job v.

Let me now relate, this matching problem to that of our candidates and jobs problem. So, this first set of vertices which we call U, can be thought of as the candidates. So, think of 1 2 3 4 5 as being the candidates. And the jobs are represented by this second set, which we call V. We draw an edge from a vertex over here to a vertex over here, if the corresponding candidate over here can do this job. So, these two edges for example, represent the fact that candidate 1 can do job 6 as well as job 7. So, in fact the list ally that I have mentioned will contain 6 and 7 in the list of candidate 1.

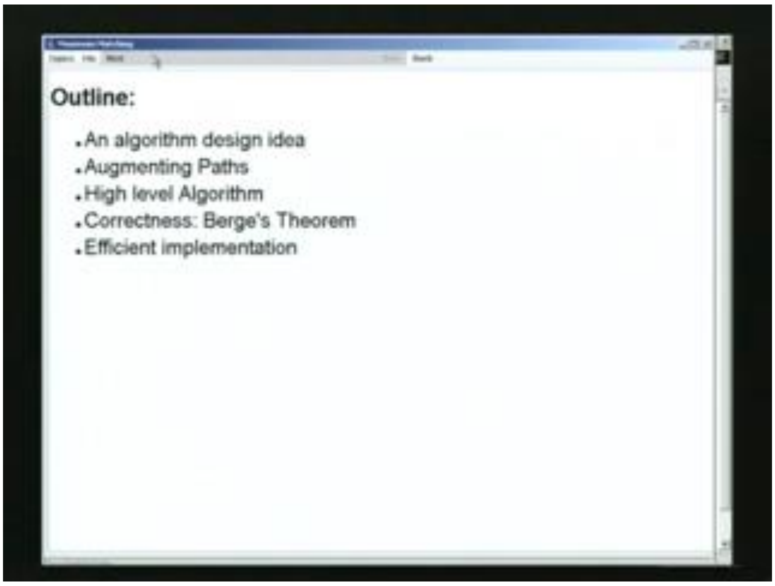
(Refer Slide Time: 05:21)



Matching: job assignment

A matching now is simply, an assignment of jobs to candidates. So, for example here is a matching which says. Let us assign to candidate 1 job 6 to candidate 3 job 7 to candidate 5 job 9. Of course, we would like to maximize the number of jobs assigned. So in fact, instead of this we should really be taking one of the red matchings that, we saw earlier.

(Refer Slide Time: 05:51)



Outline:

- An algorithm design idea
- Augmenting Paths
- High level Algorithm
- Correctness: Berge's Theorem
- Efficient implementation

Here is the outline of my lecture. So, I am going to begin with an algorithm design idea. And we will see few more ideas and refine them. Then, we will come to the notion of augmenting paths. This turns out to be a very important concept, in this whole matching area. That will understanding augmenting paths, will lead us to reasonably clean simple high level algorithm. It is correctness depends upon something called Berge's theorem. So, we will state and prove that theorem next. And then, finally we talk about how to efficiently implement this algorithm. And we will estimate its time.

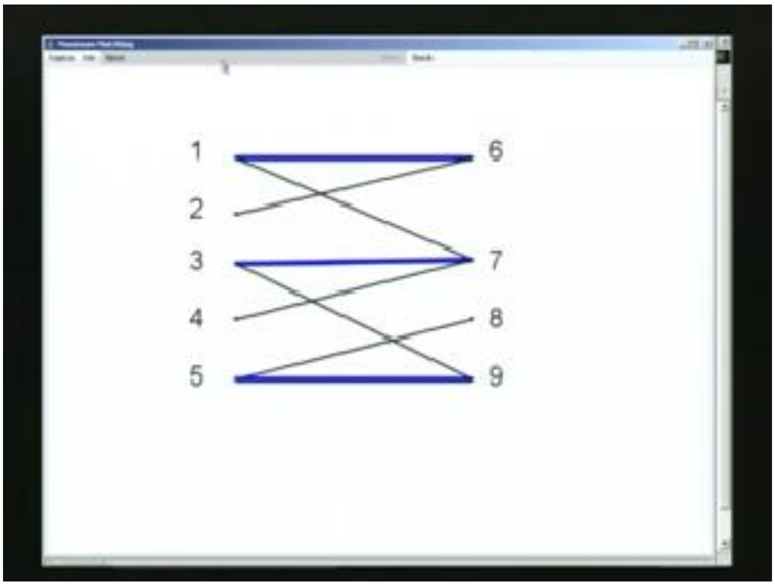
(Refer Slide Time: 06:35)

An idea:

Greedy: Keep on adding edges into M till no more edges can be added.

So, let us start with a really simple winded idea. How would you find the largest size matching? The moment you talk about larger size perhaps, the most natural thing to consider is the greedy idea. So, here is the strategy. So, we look at all the edges and maintain a set called M. And we keep on adding edges into M, till no more edges can be added. Of course, as we add edges into M you have to make sure that there are no conflicts or there are no two edges which ever are incident on the same vertex, which are placed in our set M.

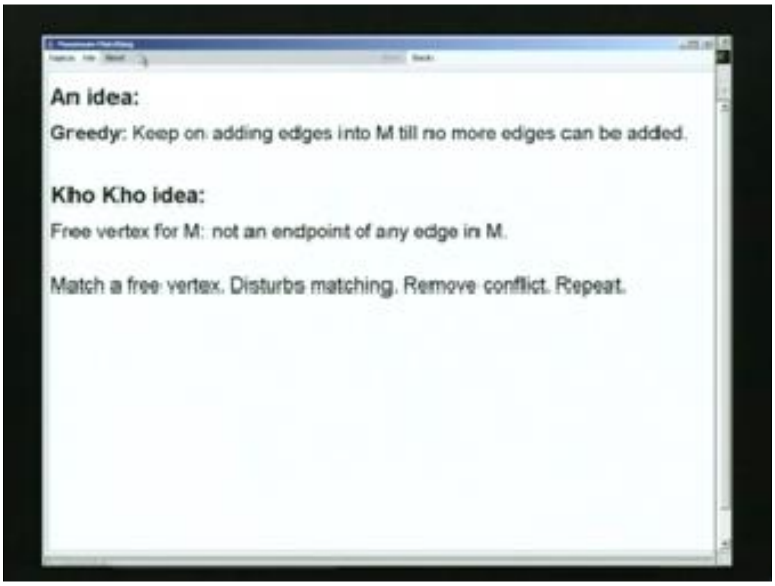
(Refer Slide Time: 07:19)



In this case for example, if we tried this idea what would happen? Well, we would start with this edge and add it no problem. Then maybe we will try to add an edge going out of 2, but this edge cannot be added because it would produce a conflict at 6. So, we take the next edge. So, we add that one. Then, we try that this edge over here this edge also cannot be added because, it will produce a conflict over here. Then, we try adding this edge.

That also cannot be added, because it would produce conflict over here. And then, we try to add an edge out of 5. And say for example, we end up adding this edge. After this, you can see that no more additions are possible. Have you reached the best possible matching? No, because we know that there exists a matching. In fact, any one of those red matchings which we saw earlier which have four edges in them rather than just the three edges, which we have found over here.

(Refer Slide Time: 08:24)



An idea:

Greedy: Keep on adding edges into M till no more edges can be added.

Kho Kho idea:

Free vertex for M: not an endpoint of any edge in M.

Match a free vertex. Disturbs matching. Remove conflict. Repeat.

So, we clearly need something better than the greedy idea. So, here is what I call the kho kho idea. So, if you have played this game of kho kho, you might remember that in that game there is a chaser who keeps running. But, who occasionally goes and knocks on the back of a person, who is sitting down. And then, that the person who was originally sitting down starts running and starts chasing the members of the opposing team.

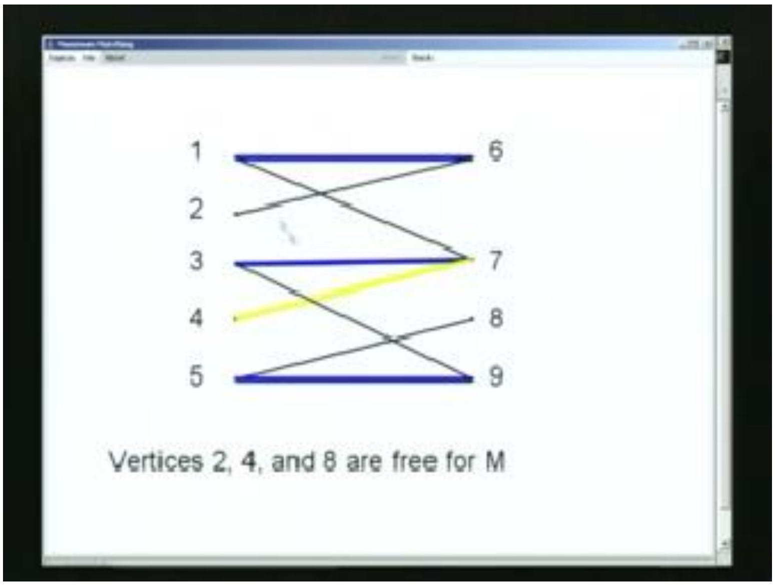
Until he or she again goes behind and knocks in the back of another person, who is sitting and who takes over. So, this is the idea that we are going to explore. So, I need a definition for that. The definition is that of a free vertex. So, given a matching M I will say that the vertex is free, if it is not an end point of any edge in that matching. So, we will take an example on this shortly. But, let me state that idea first. So, the idea is something like this.

So, suppose we have a free edge then, that indicates that maybe we can augment or we can increase the size of our matching. May be by throwing an edge by considering an edge out of that vertex, and may be trying to include that edge include that matching that we have found so far. This may succeed if it succeeds great then, we have a bigger matching. If it does not succeed, well how will it not succeed? It will not succeed because, may be it conflicts with another edge which is already present.

So, this is where the kho kho idea comes. So, this new edge that is present. The old edge that is present will get knocked up. And the new edge will sort of set in our matching.

But, now once we knock out an old edge, what will happen? Well, the other end point of that will become free. And now we will try to match that. And we will try to do on this, until we find that there is no conflict. And if we succeed then, maybe we have increased the size of our matching a little bit. So, let us try it out.

(Refer Slide Time: 10:33)

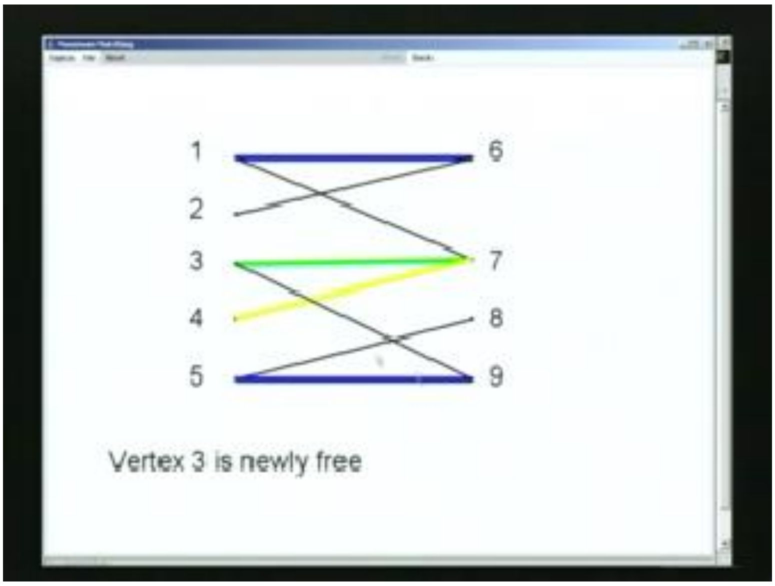


Vertices 2, 4, and 8 are free for M

So, the first thing to check is which are the free vertices? So, in this case vertex 2 vertex 4 and vertex 8 are free. Because, they do not contain incident on the any edge in the matching, edges in the matching are shown in blue over here. So, let us start with vertex 4. And let us try to see, what happens if we try to match it somewhere. Well, 4 can only be matched to 7. So, I have indicated that this is our candidate for inclusion into our old matching.

If we try to include this, what will happen? Well, that will have there will be two vertices two edges incident at 7. So, essentially we will have a conflict. Conflict just means that, there are two edges incident in a matching. We just want one edge to be incident or at most one edge to be incident. Well, if there is a conflict what do we do? Well, we are going to remove this edge.

(Refer Slide Time: 11:35)

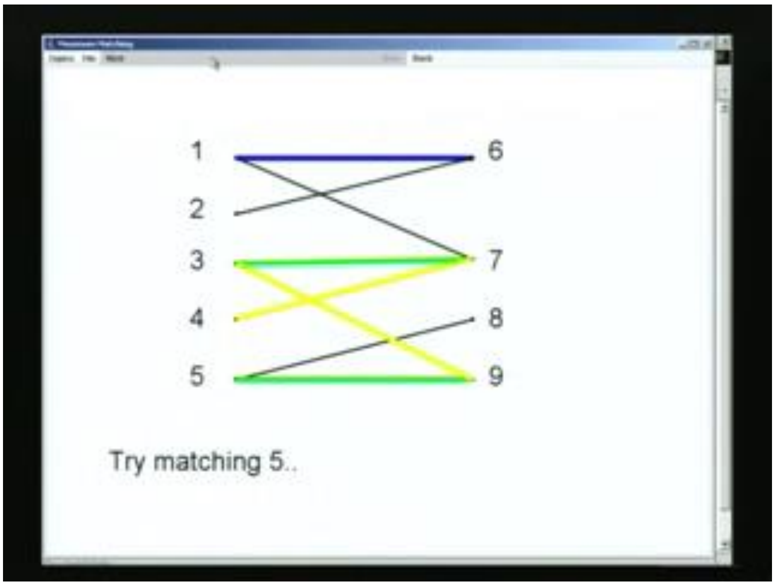


Vertex 3 is newly free

So, let us do that. So, I am going to use the color code green, the color green to show edges, which were in the matching before we started this entire procedure, but which we have just removed. Remember that yellow edges are the ones which we just added. So, they were not in the matching before, we started this procedure. But, they have just come into the matching. Green edges are the edges, which were in the matching earlier. But, now they have gone out of the matching.

If this edge from 3 to 7 goes out, what happens? Well, vertex 3 becomes free. Vertex 3 does not have an edge in the matching attached to it in longer. Remember this edge is just gone out of the matching. So, now we try to match 3. Well, we could try and match it back again to 7, but that seems foolish. We just removed this edge. So, we should not try to put it back again. So, let us try to do something else. So, the only else the only other thing that we can do is add an edge 3 to 9.

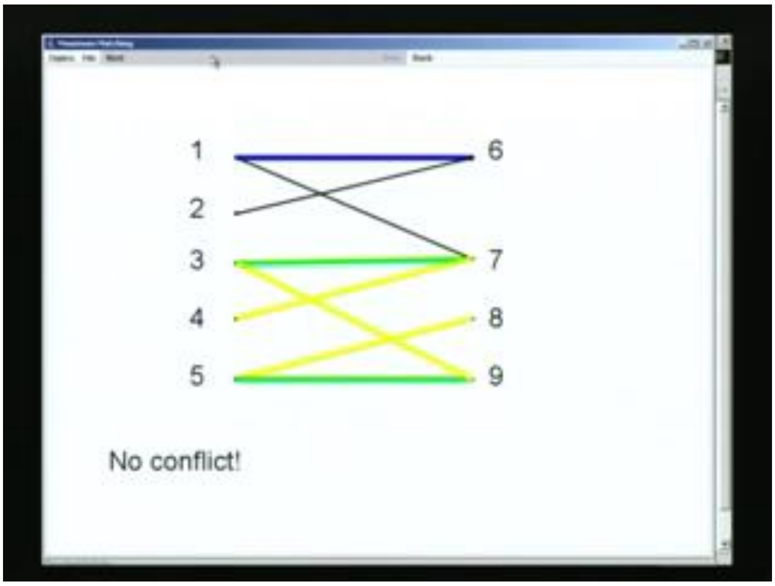
(Refer Slide Time: 12:38)



Try matching 5..

So, let us try doing that. So, now this yellow edge this yellow color says that, this edge has entered our matching and it was not there originally. Well, what does this do? It produces a conflict of time. How do we eliminate that conflict? Well, we remove this edge. So, this edge goes out of the matching. And therefore, we color it green. Now, it happens. Now, vertex 5 is free. So, we try to match it.

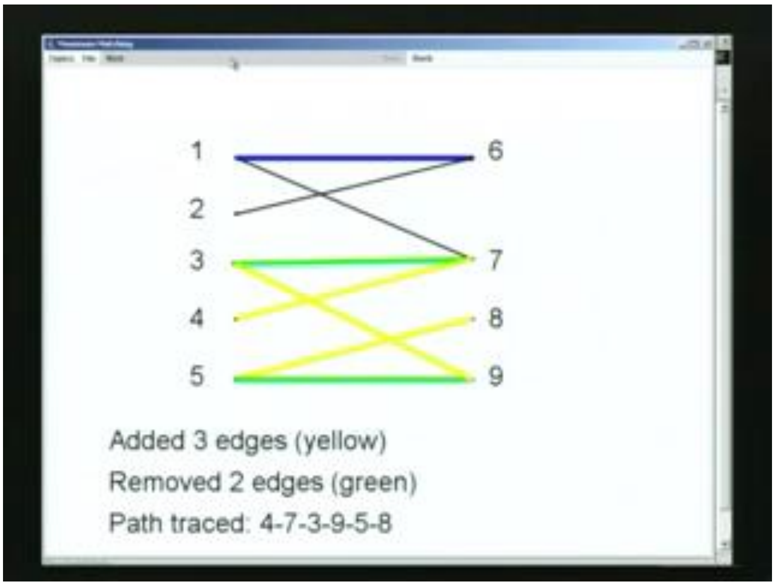
(Refer Slide Time: 13:15)



No conflict!

When we do match it, we match it to 8 and interestingly there is no conflict. So, this is the basic step that I mentioned. This is the kho kho sort that, I mentioned. Let us see, what it is accomplished.

(Refer Slide Time: 13:28)



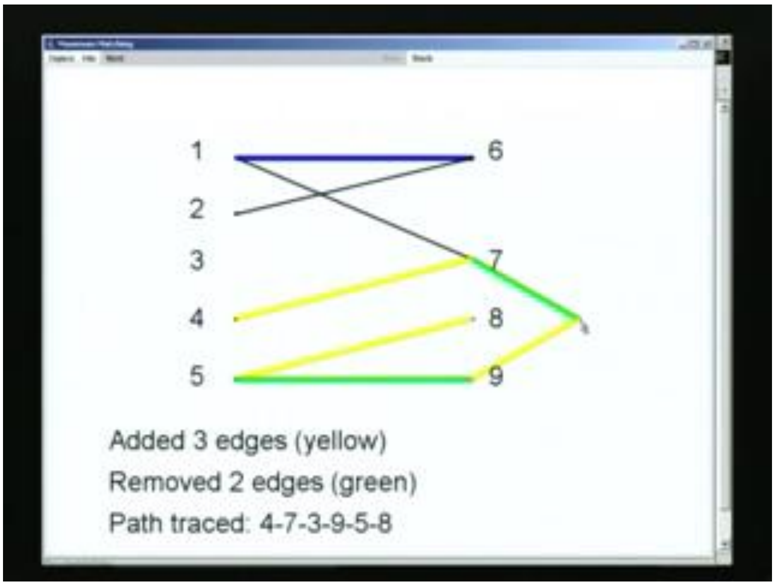
Added 3 edges (yellow)
Removed 2 edges (green)
Path traced: 4-7-3-9-5-8

So, we ended adding three edges. Which are the three edges we added? Well, these three this yellow edge, this yellow edge and this yellow edge. So, these three edges we added. So, now you know why I color those edges very, very carefully. So, that I can keep track of what exactly happen? What happened to the edges earlier in the matching? Well, some of them stayed. This edge was there in the matching earlier it state.

But, this edge which was there in the matching earlier went away. This edge which was there earlier in the matching, that also went away. But, two edges went away and three edges came in. So, now we have a better matching which consists of this edge, this edge, this edge and this edge. So, earlier we had three edges in the matching. Now, we have four edges in the matching. Now, the colors are going to tell us something more interesting, about what exactly happened.

And in fact, it makes sense to ask exactly how we traverse this graph in this entire process. So, as we executed this process we really traversed a path in the graph. So, we started at vertex 7. Then, we added edge 4 7. Then, we removed edge 7 3. Then, we added edge 3 9. Then, we added then, we removed edge 9 5 and added edge 5 8. So, that is exactly what happened? This, this, this, this, and this. In fact, there is more interesting pattern out here. So, let me show what that pattern is.

(Refer Slide Time: 15:14)



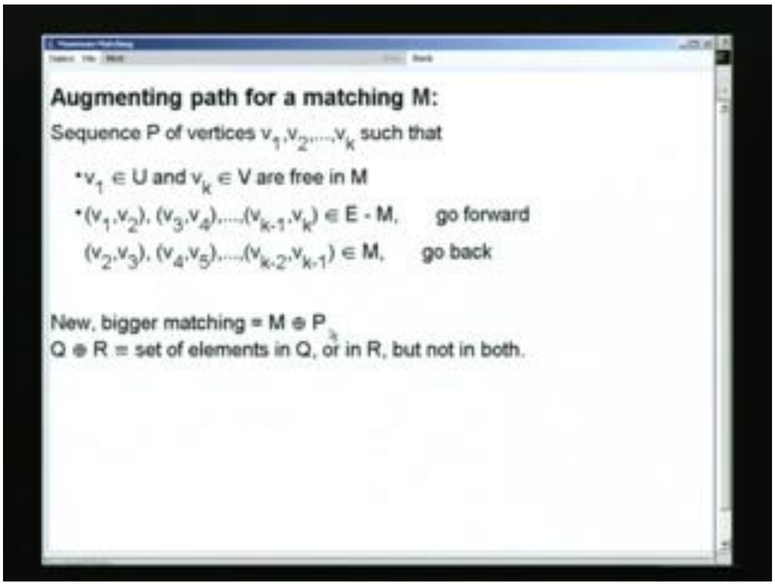
Added 3 edges (yellow)
Removed 2 edges (green)
Path traced: 4-7-3-9-5-8

Let me take this vertex 3 and let me drag it over here. So, now you should be able to see, what that pattern is even more clearly. So, if I look at the first edge on this graph, it is yellow. The second edge on this path is green, the next edge on this path is yellow again its green again it is yellow. So, in fact the colors of the edges alternate along this path. And that is of course, to be expected why, because we alternately added and removed edges.

In fact, there is one more interesting thing. So, whenever we were adding edges we were going forward in the graph and we were following edges which were originally not in the matching. When we removed edges, we were following edges which were originally in the matching. But, we were going backwards in the graph. And similarly, again we go forward backward and finally forward again.

So, you see that what we did, the path we traced at this entire procedure has lots of interest in properties. And in fact, this path that we traced is so important that it is given a name, it is called an augmenting path.

(Refer Slide Time: 16:28)



Augmenting path for a matching M:

Sequence P of vertices $v_1, v_2, \ldots, v_k$ such that

- $v_1 \in U$ and $v_k \in V$ are free in M
- $(v_1,v_2), (v_3,v_4), \ldots, (v_{k-1},v_k) \in E - M$, go forward
  $(v_2,v_3), (v_4,v_5), \ldots, (v_{k-2},v_{k-1}) \in M$, go back

New, bigger matching = $M \oplus P$

$Q \oplus R$ = set of elements in Q, or in R, but not in both.

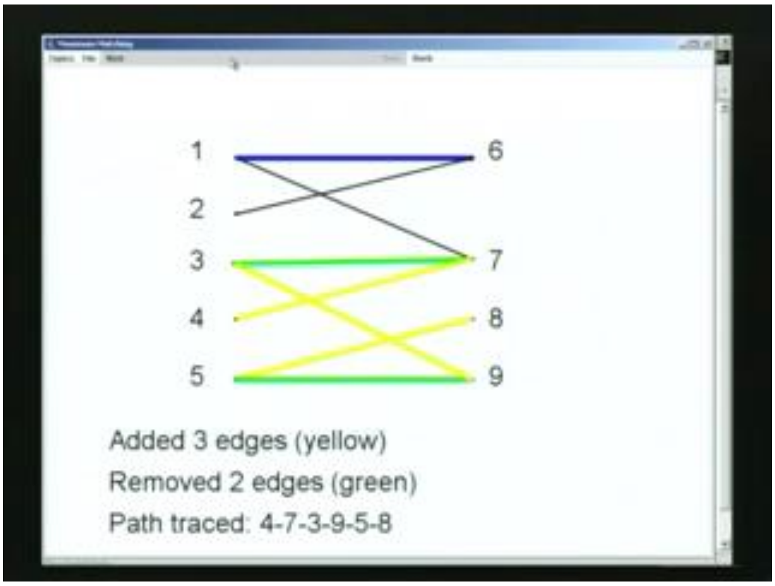
So, given a matching there is a notion of an augmenting path. An augmenting path is a sequence of vertices v 1 v 2 v k. Such that the first vertex is in the set u in the graph, the left the set of vertices on the left corresponding to the vertex 4 that we started off with and it is a free vertex. That is exactly, how we started out the construction. The last vertex is also free because, that is what enabled us to discover an extra edge.

And then, the intermediate vertices well the vertex from v 1 to v 2 was originally in the graph, but not in the matching. And that is exactly, what this set E minus M supposed to denote. E is the set of edges from that, I remove the set of edges which were in the matching earlier. So, this edge v 1 v 2 is now was supposed to be in this set E minus M. The next edge on the path however, v 2 v 3 is a backward edge and it belongs to the matching.

And this edge is the forward edge. So, we go forward then, we go backward then we go forward again potentially. Again we follow an edge, which is not in the matching and we go forward. And maybe we go backward again. And we do this several times, until we end with this set v that is what a augmenting path is. The operation of taking an augmenting path p and a matching m and generating a new bigger matching which we just did, we will abbreviate as M symmetric difference P. Why is this called symmetric difference?

Well, here is the definition of this circle plus operator. We will define Q circle plus R or Q symmetric difference R, as the set of elements in Q or in R, but not in both. So, in that sense it is the symmetric difference. So, it is the sense in which Q and R are different from each other, not their similarity, but there is difference. So, in fact you can see that the new matching is the symmetric matching is the symmetric difference of the old matching and the path. So, let us take a look at that quickly.

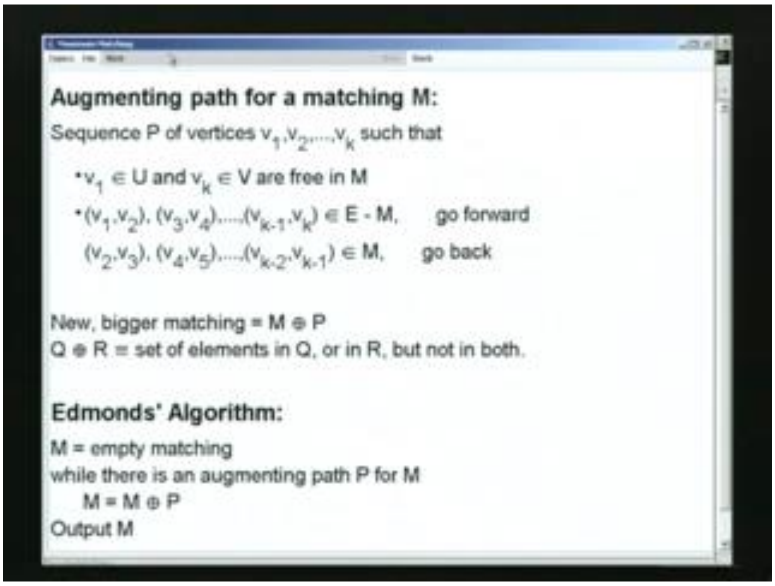
(Refer Slide Time: 19:04)



Added 3 edges (yellow)
Removed 2 edges (green)
Path traced: 4-7-3-9-5-8

So, the old matching was this matching 1 6 3 7 and 5 9. So, the old matching is the blue edges and the green edges. The path is this. So, the path is the yellow edges and the green edges. So, as you can see the green edges form the intersection between the path and the old matching. And therefore, they have been removed. And what has included in the new matching is just the difference, the symmetric difference between the path and the old matching. So, these are edges which were in the path, but not in the matching. This is also an edge this is an edge which is in the matching, but not in the path. And so, the new matching will just contain this and these.

(Refer Slide Time: 20:01)



Augmenting path for a matching M:

Sequence P of vertices $v_1, v_2, \dots, v_k$ such that

- $v_1 \in U$ and $v_k \in V$ are free in M
- $(v_1,v_2), (v_3,v_4), \dots, (v_{k-1},v_k) \in E - M$,   go forward
  $(v_2,v_3), (v_4,v_5), \dots, (v_{k-2},v_{k-1}) \in M$,   go back

New, bigger matching = $M \oplus P$
$Q \oplus R$ = set of elements in Q, or in R, but not in both.

Edmonds' Algorithm:

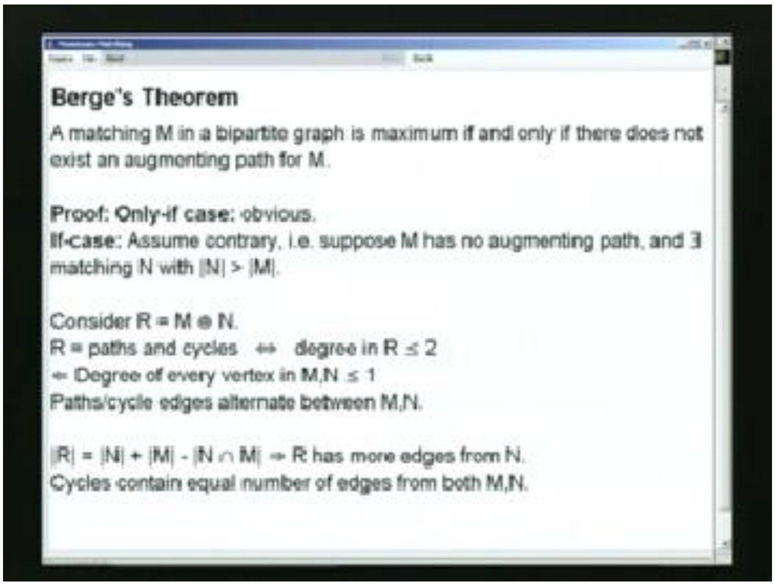
```
M = empty matching
while there is an augmenting path P for M
    M = M ⊕ P
Output M
```

So, augmenting paths play a big role seem to play a big role when we want to increase the size of our matchings. In fact, Edmonds who was a eminent computer scientist from, Ii guess from the previous century who is one of the who could even be called as one of the founders in some sense of analysis of algorithms. So, I am going to describe an algorithm which is credited to Edmond’s. So, the algorithm is essentially based on this idea of augmenting paths. So, we start off with M which is an empty matching which is an empty set.

So, there is nothing in it. And then, we keep on finding the augmenting path. So, we check if there is an augmenting path P for this M, defined in this sense. If there is then, we perform this operation and set M to the result. We keep on doing this, until we discover that we cannot augment our current matching. The moment that happens, we stop and we output M. So, this is the algorithm. Does it seem reasonable? Well, yes it seems reasonable.

We seem to be adding edges if possible, but there is a old problem which we had with our greedy idea as well. What if we discover that there is no augmenting path. Can we stop then really or it is possible that there is some bigger matching which we will find using this idea. So, this possibility can be ruled out and this is done by a theorem attributed to Berge.

(Refer Slide Time: 21:55)



Berge's Theorem

A matching M in a bipartite graph is maximum if and only if there does not exist an augmenting path for M.

Proof: Only-if case: obvious.
If-case: Assume contrary, i.e. suppose M has no augmenting path, and ∃ matching N with |N| > |M|.

Consider R = M ⊕ N.
R = paths and cycles ⇔ degree in R ≤ 2
⇐ Degree of every vertex in M,N ≤ 1
Paths/cycle edges alternate between M,N.

|R| = |N| + |M| - |N ∩ M| ⇒ R has more edges from N.
Cycles contain equal number of edges from both M,N.

So here is what Berge’s theorem says. Berge’s theorem says that a matching M in a bipartite graph is maximum, if and only if there does not exist a augmenting path for M. We will prove this in a minute, but I just want to persuade you that, this is exactly the theorem that we wanted. So, this theorem says that if ever we come to a point at which we cannot find an augmenting path. Then, we must have in our hands a maximum sized matching.

So, this theorem justifies or proves the correctness of Edmond’s algorithm. So, let us now prove this theorem. The proof is in two parts. Since, this is a if and only if theorem. The first part is the only if. So, what are we required to prove over here? If a matching in a bipartite graph is maximum then, there cannot exist an augmenting path. This should be quite obvious. If there existed an augmenting path, what would happen?

Well, we could add we could augment that path with our matching. We could compute M circle plus P and we would get a bigger matching. Is if that possible? No, because we said that M is already maximum. And therefore, the only if case is obvious. The interesting case is the, if case. We are going to prove this case by contradiction. So, how does this work? Well. So, let us assume the contrary.

So, let us suppose that M has no augmenting path. So, M has no augmenting path that of course, by itself does not make up the contradiction. So, we are going to assume that M has no augmenting path. And there exists a matching M such that, N is larger than M. So, this is the case that we really wanted to watch out with. We cannot find an augmenting path, neither have we got the best matching. So, this is the if case. And this is the case that we really want to rule out.

So, we are worried about such M. We have in our hands matching M. And we know that there is, we are assuming that there is a bigger matching N somewhere out there. Well, at this point it is natural to ask, what is the difference between M and N? So, that is exactly what we do. So, we will ask will say, let R be the symmetric difference of M and N. And then, let us investigate what this, what the properties of the symmetric difference are.

I claim that, so we are going to look at this symmetric difference R. My first claim is that this R must be composed of paths and cycles. Why is that? Well, before saying why that is, let me just say that the claim is equivalent to stating that the degree of every vertex in R is at most 2. Remember M and N are sets of edges. R is also a set of edges. And it is over the same vertex set u and v. So, I can talk about the degree.

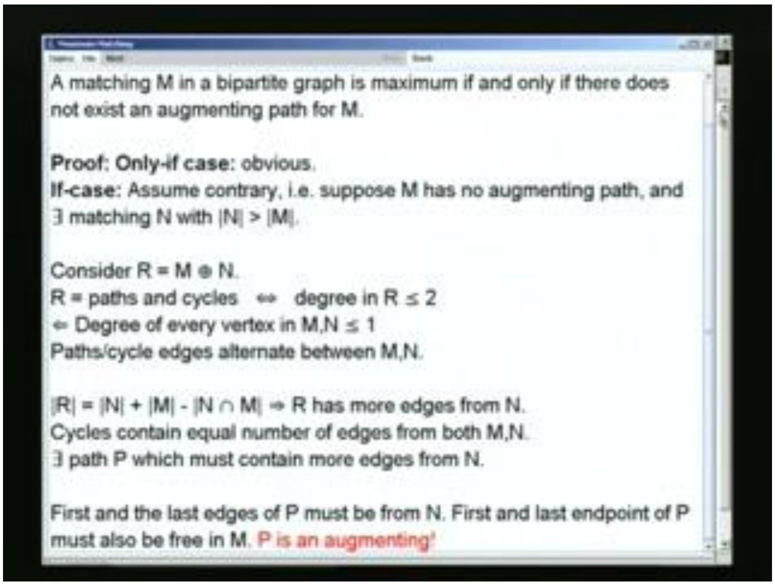
So, I claim that if R is made up of paths and cycles, it is the same thing that saying the degree of every vertex in R is at most 2 obviously. The degree of any vertex in a path is 2, the degree of any vertex in a cycle is also 2. So, if I can prove this I would have proved this. Well, why is this true? Well, just examine how we constructed R? We constructed R by taking some edges of M and some edges of N. But, note that the degree of every vertex in M and N is at most 1.

The edges of M and N are such that, on every vertex at most 1 edge from either M or N is an incident. So, even if I take the union forget the symmetric difference, even if I take the union the degree of every vertex in M and N will be at most. Since the degree of M and N in every vertex is at most 1, the degree of the union will be at most 2. So, from this it follow that this is true. Therefore, this is same as this. Here is an interesting fact about R.

So, it consists of paths and cycles, but the edges in R alternate between M and N. Why is that? Well, could there be two consecutive edges in R coming from M. No, because M has been defined to consist of edges which are incident on vertices. But, at most one edge is incident on any vertex. So, in M itself two edges are never incident on the same vertex and therefore, when those edges going to R this property will stay around and similarly for N. And therefore, we know that the paths and cycles inside R consist of alternate edges of M and N.

Some more properties, what is R? Well R is the symmetric difference. So, that is as good as saying that, we take the union and then we remove the intersection. So, the size of R is size of N plus the size of M minus the size of the intersection. But note that, N has bigger size than M. So, we subtract something, but in the end we subtract something we just common to both M and N. So, in the end R must have more edges in it from M rather than from N. So, we just how to examine the implications of this.

(Refer Slide Time: 28:28)



A matching M in a bipartite graph is maximum if and only if there does not exist an augmenting path for M.

Proof: Only-if case: obvious.
If-case: Assume contrary, i.e. suppose M has no augmenting path, and $\exists$ matching N with $|N| > |M|$.

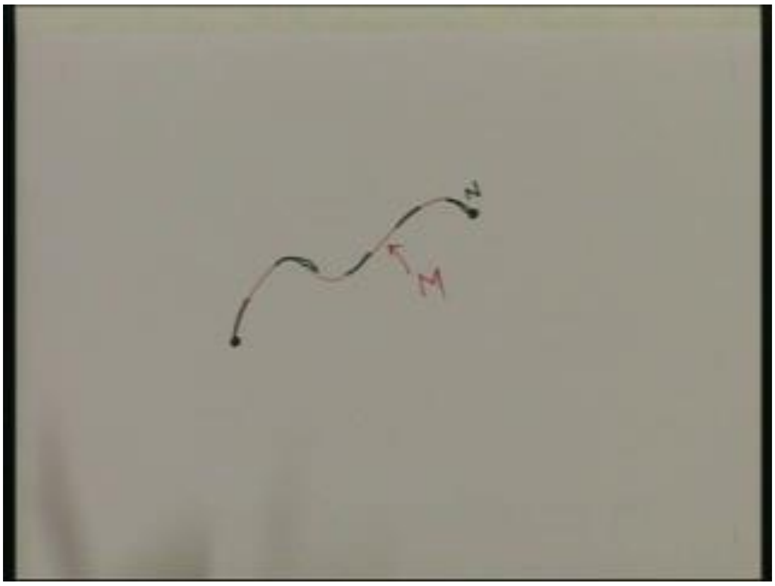
Consider $R = M \oplus N$.
R = paths and cycles $\Leftrightarrow$ degree in $R \leq 2$
$\Leftarrow$ Degree of every vertex in $M,N \leq 1$
Paths/cycle edges alternate between M,N.

$|R| = |N| + |M| - |N \cap M| \Rightarrow$ R has more edges from N.
Cycles contain equal number of edges from both M,N.
$\exists$ path P which must contain more edges from N.

First and the last edges of P must be from N. First and last endpoint of P must also be free in M. P is an augmenting!

Let us look at cycles first. Each cycle consists of an equal number of edges from both M and N. So, if we just look at the cycles this fact cannot be explained. So, what first happen? There has to exist a path, which must contain more edges from N. If it contains more edges from N, what do we know about such a path? Well, let us take a picture.

(Refer Slide Time: 29:03)

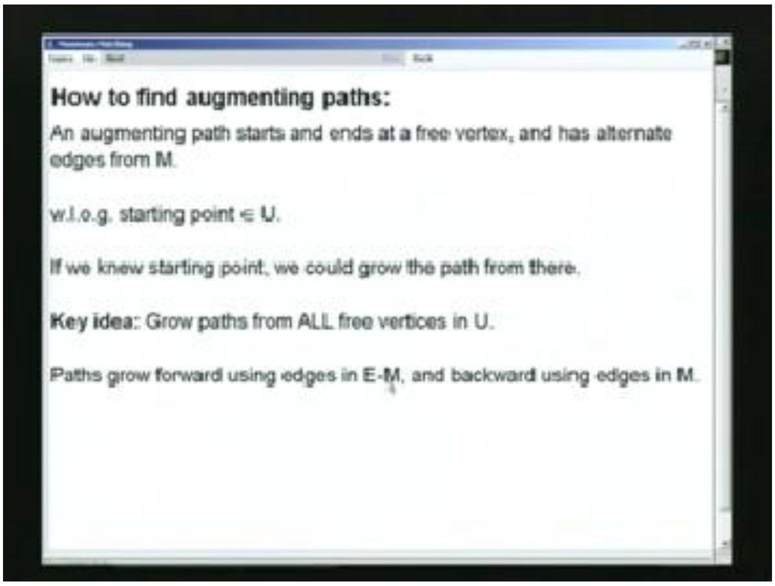


Let us draw a picture over here. So, here is this path. It contains more edges from N. So, let us say N edges are in black. So, if it contains more edges from N clearly, the first edge has to be from N then, from M then, from N then from M. And finally, the last edge must be from N. So, now what do we know about these vertices? So, will these vertices have an additional edge either from M or N going out.

Well, if there was a red edge going out then, this could be a path which have been continued. We could have considered that itself, but this is the maximal path that we are considering. So, there cannot be a red edge going out of here. There cannot be a black edge going out over here, similarly over here. So, that means that the first and the last end points of P must also be free in M and what is this.

This is simply the definition of an augmenting path. So, this path p is an augmenting path for this matching M. So, we have proved that an augmenting path exists and in other words, this you can improve your path. And we started off by saying that, M has no augmenting path. So, we have got to a contradiction. So, this proves Berge’s theorem. So, we have proved Berge’s theorem.

(Refer Slide Time: 30:35)



How to find augmenting paths:

An augmenting path starts and ends at a free vertex, and has alternate edges from M.

w.l.o.g. starting point ∈ U.

If we knew starting point, we could grow the path from there.

Key idea: Grow paths from ALL free vertices in U.

Paths grow forward using edges in E-M, and backward using edges in M.

So, we know that augmenting paths are useful and not only useful, but they are sufficient. So, the only question that, remains is can we find augmenting paths quickly? So, what are the properties that we know about augmenting paths? Well, an augmenting path starts and ends at a free vertex. It starts at a vertex, say U and ends in a vertex V. So, without loss of generality or vice versa, so without loss of generality we can assume that the starting point is U.

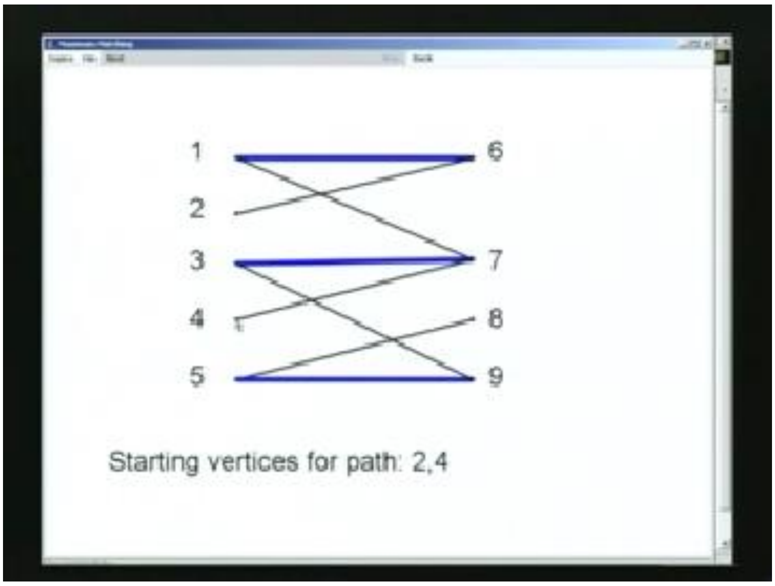
So, we are given a graph and we want to start an augmenting path. We want to check whether there exists a augmenting path, starting at some vertex in U. If we knew where the path started, we could just try growing the path out of that vertex. So, we could try something like depth first search or something like that. And try going out from that vertex into the rest of the graph. But, we do not know where it starts. So, here is an interesting idea.

So, since we do not know where it starts, we are going to start growing from all the free vertices in U. Now, we know something more about augmenting paths, which is that the paths have to grow forward using only edges in E minus M. E difference the edges which are in E, but not in the matching E minus M. And the path must grow backwards using edges, which are in the matching. You may do this several times.

It may go forward and go backward and then, forward again several times, but every time it goes forward. It must use an edge which is not in the matching. And every time it goes backwards, it must use an edge which is in the matching. Finally, if we reach a free node or a free vertex in the set V, which is the set on the right side through any paths, whatever we are done.

So, that is how we are going to grow these parts. And if we reach a free vertex, great we are done. We would have discovered a path. Now, it turns out that you can package this set of ideas very nicely as a breadth first search on a new graph. Well, on a slightly a graph which has been derived from this M and G. This graph is going to be very similar to G, but it is going to be crucially and slightly different. So, let us take a look at this. But let us first take an example first of, how we can do this?

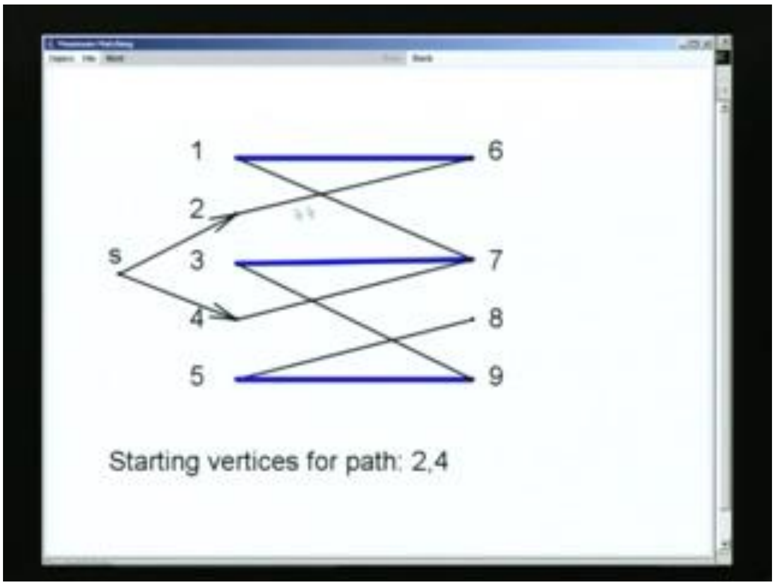
(Refer Slide Time: 33:28)



Starting vertices for path: 2,4

So, here is our old graph and here is our old matching. So, let me try to see how this idea will work out, on this graph. So, the first point was to look for a free vertex, actually a free vertex on the U side. We say here two vertices are free, vertex.2 and vertex 4. These are the free vertices. So, these are the vertices from which we can start the paths. So, we could say for example, that let us grow paths from 2 and 4 going in the forward direction and backward direction as we just described.

(Refer Slide Time: 34:08)



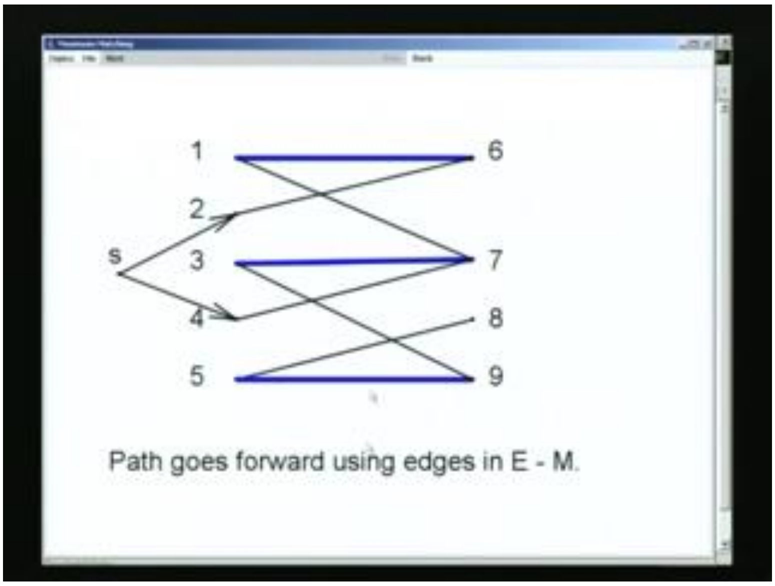
Starting vertices for path: 2,4

Instead of that, just to make our description nice and compact. We are going to throw in a new vertex, we will call S. We will also throw in two edges, going out of S to both the free vertices or to all the free vertices, whatever free vertices there are. And in fact, we are going to direct these edges. So, earlier graph was an undirected graph. this new graph is going to be a directed graph.

We can do breadth first search or any kind of search on an undirected graph, just as well as on a directed graph. So, here we are going to do it in a directed graph. So, now instead of saying that we go upon paths 2 and 4, notice that we can just say grow paths out of S, just a single vertex S. So, we want to grow out of S. And once we do and if we do that, we will naturally hit 2 and 4 which is where you want to go anyway.

What do we next? Well, we want to grow the path itself. And for growing the path, we need to use edges which do not belong to the matching. And furthermore, we know that these edges will be used only in the forward direction.

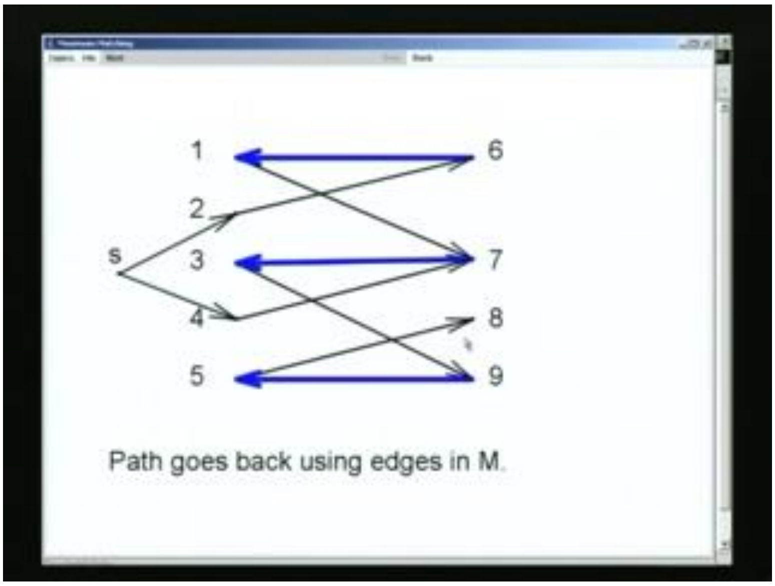
(Refer Slide Time: 35:24)



Path goes forward using edges in E - M.

So, a natural way to enforce this constraint is to say, this is the constraint that we want to enforce. So, the natural way of doing that is to direct these edges in the forward direction. So, every time we grow the paths. We know that the edges which are not in the matching will be used, only in the forward direction. So, we put a direction on them. So, that forces the search to use them, only in the forward direction.

(Refer Slide Time: 35:53)

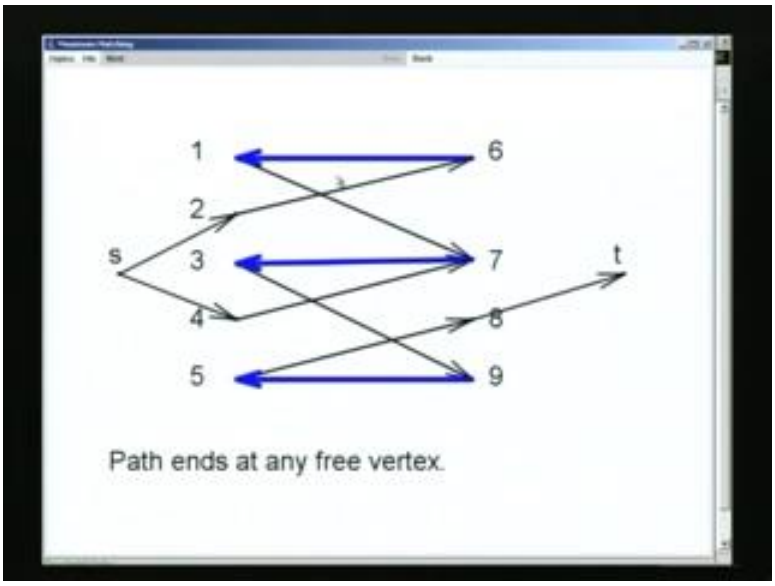


Path goes back using edges in M.

Now, what happens when the path the augmented path goes backwards? Well, it uses edges, which are in the matching. If it uses edges which are in the matching, it only uses them going backwards. So, we put backward arrows on this. So, the idea now is, we start over here we keep we go forward we are allowed to go forward. Then, we are allowed to go forward using edges, is not in the matching.

We can come back, we can go forward we can come back, every time we come back we need to use edges in the matching. Every time we go forward. We are supposed to use edges, which are not in the matching. And finally, we would like to end up with a free vertex in this set V. Again there could be many free vertices over here. And instead of saying, let us end up saying free vertex in this set V. Here is what we will do?

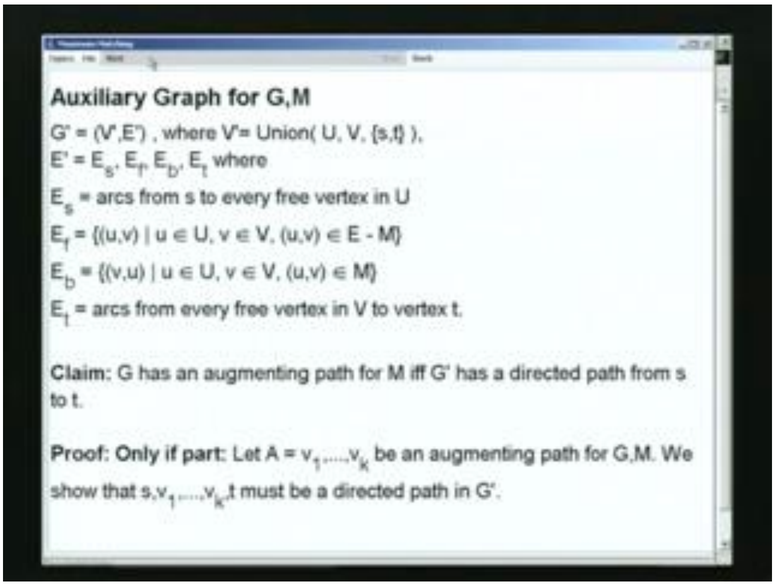
(Refer Slide Time: 36:44)



Path ends at any free vertex.

Instead of saying that, the path must end at any free vertex we will put a vertex t out here. And all the free vertices we will connect it to t. In this case, there is one free vertex. So, this is what we will connect to t. So, now what is the problem that we want to solve? While we want to ask the question, does there exist a path from s to t in this directed graph. Notice that by putting directions, we have essentially enforced all the constraints that we wanted on that augmenting path.

(Refer Slide Time: 37:18)



Auxiliary Graph for G,M

G' = (V',E') , where V'= Union(U, V, {s,t}),
E' = E_s, E_f, E_b, E_t where

E_s = arcs from s to every free vertex in U

E_f = {(u,v) | u ∈ U, v ∈ V, (u,v) ∈ E - M}

E_b = {(v,u) | u ∈ U, v ∈ V, (u,v) ∈ M}

E_t = arcs from every free vertex in V to vertex t.

Claim: G has an augmenting path for M iff G' has a directed path from s to t.

Proof: Only if part: Let A = $v_1, \ldots, v_k$ be an augmenting path for G,M. We show that s,$v_1, \ldots, v_k$,t must be a directed path in G'.

So, let us formalize this. Let us state this algebraically. So, we will define this directed graph and we will call it the auxiliary graph, for G and M. So, what does this graph look like? First of all, we will get symbol G prime. It is going to consist of vertex set V prime, which I will define in a minute and edge set. So, these edges are actually directed. So, this edge set is E prime. So, what is the vertex set?

V prime is the union of set U in the original graph, the set V in the original graph and these two vertices, which we added the s vertex and the t vertex. Let us define the edge set E prime consists of these edges, which goes out of s. So, just to remind you. So, these are the edges which are E s edges. E f is the edges, which are the forward edges. So, let us go over them step by step. E f are the set of edges in the forward direction that is, what this f is supposed to end up.

So, they consist of the arcs of the form u v, where u belongs to the left hand set of vertices, v belongs to little v belongs to the right hand side vertices of capital V and u v is not in the matching, but it is in the edge set. E sub b are the backward edges, the edges which we directed backwards. And these are simply all the edges in the matching. So, these are all the edges which were not in the matching which we directed forward.

These are all the edges which in the matching, which we directed backwards. So, notice that this is u to v whereas, this is v to u. And finally, we are going to put edges out of every vertex, every free vertex in V to the vertex t. So, this defines our auxiliary graph.

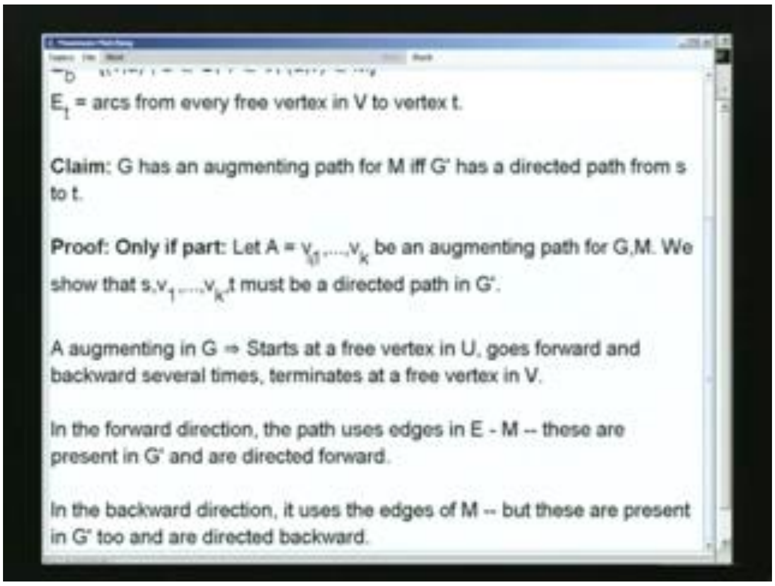
So, now comes our main claim. Our main claim says that, G has an augmenting path for this matching M if and only if, G prime has a directed path from s to t. We will prove this, we just point out why this claim is important.

Notice that, we did not have any specific way of finding augmenting paths. However, given a graph G prime does it have a directed path from s to t. That we know how to find it completely? That is just simple, breadth first search, depth first search or whatever you like. So, this claim would enable us to find augmented paths very quickly. And that is why, it is a very significant claim. So, let us prove it. The proof really goes along.

Pretty much along the lines of the construction, as I explained it in a minute ago, but I will try to explain it formally right now, without reference to a specific graph. So, let us look at the only if part first. So, the only if part says that suppose G has an augmenting path then, G prime must have a directed path from s to t. So, the only part says that let A be an augmenting path for G M then, we will show we must show that s v 1 v k the same path, but in the new in the graph G prime, must be a directed s t path.

So, I hope there is no confusion. Because, we are using the same set of vertices, but we have carefully defined everything in this definition. And also in the previous graph in the previous picture, we actually use the same set of vertices. And we just transformed our original G to G prime. By context you should know, when I refer to over vertex whether I am referring to it inside G prime or inside G. So, we had given this augmenting path v 1 to v k. And we want to show that, this must be a directed path in G prime.

(Refer Slide Time: 42:04)



E_t = arcs from every free vertex in V to vertex t.

Claim: G has an augmenting path for M iff G' has a directed path from s to t.

Proof: Only if part: Let A = $v_1, \ldots, v_k$ be an augmenting path for G,M. We show that $s, v_1, \ldots, v_k, t$ must be a directed path in G'.

A augmenting in G ⇒ Starts at a free vertex in U, goes forward and backward several times, terminates at a free vertex in V.

In the forward direction, the path uses edges in E - M -- these are present in G' and are directed forward.

In the backward direction, it uses the edges of M -- but these are present in G' too and are directed backward.

What do we know about augmenting paths? It starts at a free vertex in U, goes forward and backward several times and terminates at a free vertex in V. So, v 1 is a free vertex and the path goes forward from v 1. When it goes forward, we must have an analogous edge in G prime. So, all that you need to do is that, analogous edge is present in G prime. Note that in the forward direction, the path users edges in E minus M. This is inside G.

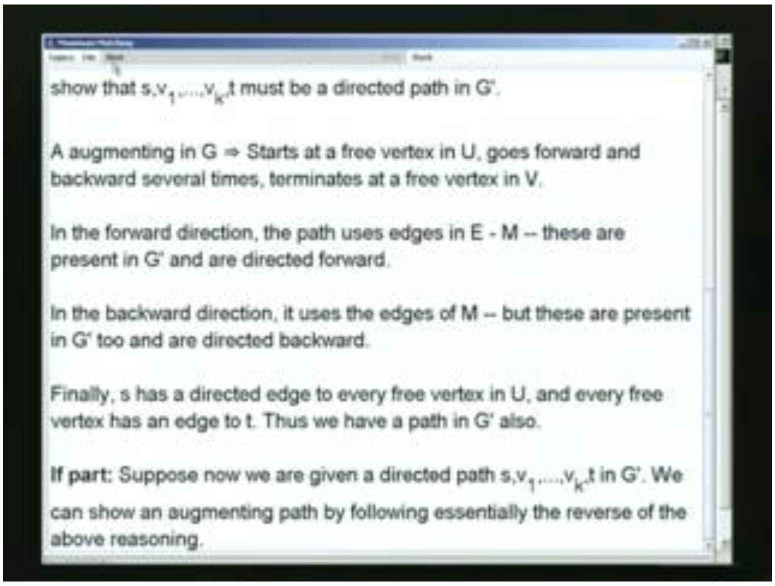
An augmenting path in the forward direction, uses edges which are not in the matching. The path is moving forward, but it is using edges which are not in the matching. Now, these edges are in fact present in G prime. These are exactly the edges. They are present in G prime and in fact, they are directed in the forward direction. So, if we look at the forward going edges in this, they are all present in G prime and they are all oriented properly.

They are oriented as per the movement of the direction, movement of the path direction. What does the path do? Well, the path can go backward. When it goes backward, it uses the edges of M. But, these edges are also present in G prime and they are directed backwards. So, again this is exactly what we wanted. Well, there is no coincidence over here, because we arranged it to be this. So, in some sense if you followed that example then, this should not be surprise to you at all.

So, again going back to this, v u is an edge going from v to u and that is how we have oriented it. If for all such edges which belong to the matching. So, what have we proved then? We have proved that, this portion which is an augmenting path which is also present in G prime and it has proper orientation. So, all that we need to argue is the s to v 1 connection and v k to t connection. So, what do we know about s to v 1.

So, s has a directed edge to every free vertex in U. And of course, v 1 must be a free vertex. So, this since it is an augmenting path. So, s to v 1 must be an edge in G prime. Similarly, v k to t also is an edge, because every free vertex has an edge s to t. So, that is also present in G prime and thus we have a path in G prime also. So, this entire thing is a path in G prime exactly as we wanted.

(Refer Slide Time: 45:14)



show that $s, v_1, \ldots, v_k, t$ must be a directed path in G'.

A augmenting in G ⇒ Starts at a free vertex in U, goes forward and backward several times, terminates at a free vertex in V.

In the forward direction, the path uses edges in E - M -- these are present in G' and are directed forward.

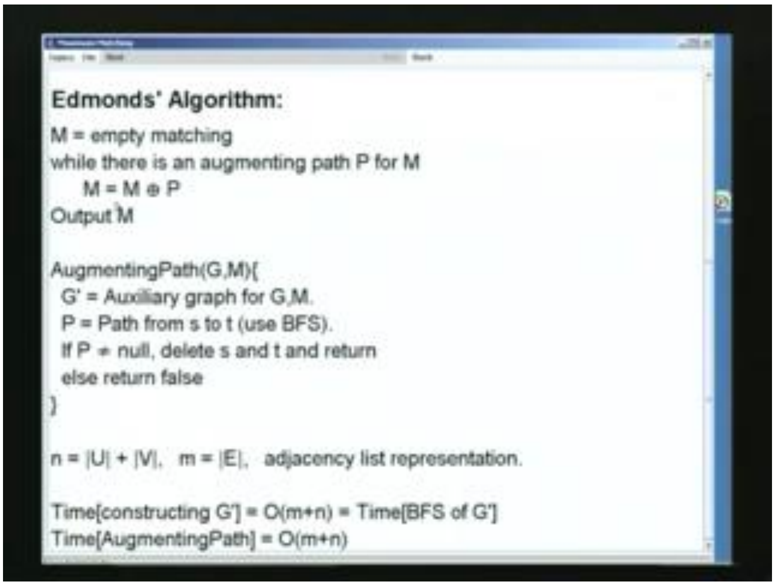
In the backward direction, it uses the edges of M -- but these are present in G' too and are directed backward.

Finally, s has a directed edge to every free vertex in U, and every free vertex has an edge to t. Thus we have a path in G' also.

If part: Suppose now we are given a directed path $s, v_1, \ldots, v_k, t$ in G'. We can show an augmenting path by following essentially the reverse of the above reasoning.

The if part, the if part says that if G prime has a directed path from s to t then, G must have an augmenting path. So in fact, you will see that exactly the reverse of this reasoning will accomplish the, if part also. So, let us summarize what we have done. So, we have defined this auxiliary graph. And using the auxiliary graph, all we have to do is just find the path in it. And we get an augmenting path. So, we find a path from s to t and we get an augmenting path.

(Refer Slide Time: 46:03)



Edmonds' Algorithm:

```
M = empty matching
while there is an augmenting path P for M
    M = M ⊕ P
Output M

AugmentingPath(G,M){
  G' = Auxiliary graph for G,M.
  P = Path from s to t (use BFS).
  If P ≠ null, delete s and t and return
  else return false
}
```

n = |U| + |V|,   m = |E|,   adjacency list representation.

Time[constructing G'] = O(m+n) = Time[BFS of G']
Time[AugmentingPath] = O(m+n)

So, now that brings us back to our algorithm. So, that brings us back to the algorithm. So, we just have to build up on this step of finding an augmenting path, but we know how to do that. So, here is our augmenting path procedure. So, we construct G prime just as we defined a minute ago then, we find p the path from s to t. We can use breadth first search for it or we can use depth first search, it does not matter.

But, somehow we do it. And then, if p is not null then, we delete that as t and return the augmenting path. This return path will be used over here to augment the matching. So, let us now analyze this. So, let us say n denotes the cardinality of U plus the cardinality of V or the total number of vertices. Let m denote the cardinality of the edge set of the original graph. And let us assume that, the graphs are represented in the adjacency list representation.

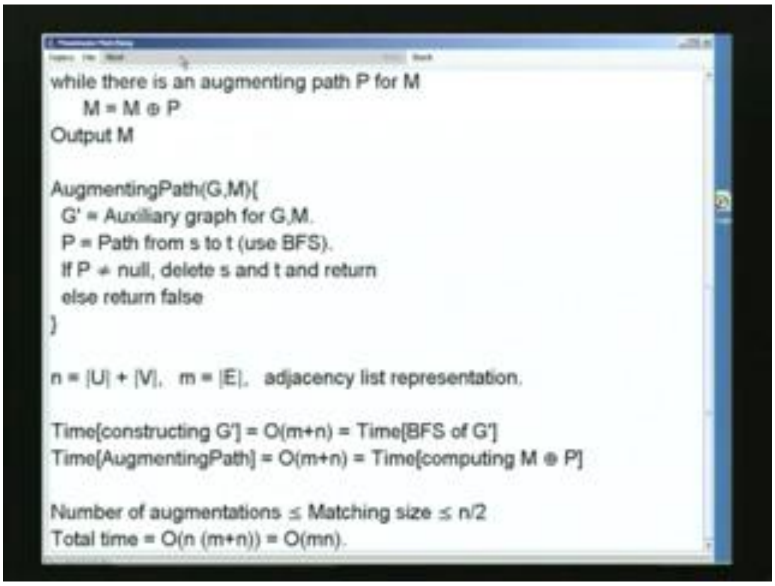
In fact, we will keep m also in some adjacency list representation. Just for the purpose of simplicity of thinking about this whole thing. So, what do we know now? Well, we have to analyze what the time for constructing G prime? And G prime is and then, doing the breadth first search and so on. So, here we construct this graph G prime. So, how did we do that? We took G and we took the matching and we took its union, we oriented the edges.

So, in any case that can be done in time proportional to the sizes of two graphs, which is O of m plus n. The next step over here is to find the path from s to t using BFS. BFS

Breadth First Search takes time again O of m plus n. So, this step also takes time O of m plus 1. So, both of these steps take time m plus n. Time for this entire procedure, I claim this is going to be O of m plus n, this entire procedure augmenting path.

That is, because this part deleting s and t can be done in constant time. And therefore, this entire thing is just the sum of this plus this, which is O of m plus n. So, this takes O of m plus n time. So, the only question that remains is, how many times do we augment? Well and how do we do this augmentation itself? In fact, in m plus n time you can compute M augmented with P also. Because, that is just going over the graph completely once even that, will take time O of m plus n. How many augmentations do we do?

(Refer Slide Time: 48:54)



```
while there is an augmenting path P for M
    M = M ⊕ P
Output M

AugmentingPath(G,M){
  G' = Auxiliary graph for G,M.
  P = Path from s to t (use BFS).
  If P ≠ null, delete s and t and return
  else return false
}
```

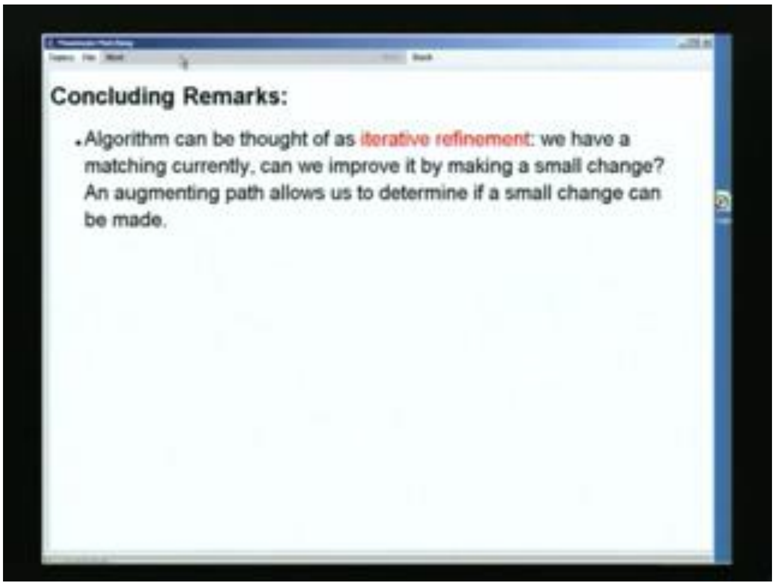
$n = |U| + |V|$, $m = |E|$,   adjacency list representation.

Time[constructing G'] = $O(m+n)$ = Time[BFS of G']
Time[AugmentingPath] = $O(m+n)$ = Time[computing $M \oplus P$]

Number of augmentations $\leq$ Matching size $\leq n/2$
Total time = $O(n\ (m+n)) = O(mn)$.

Well, what do you know about matching size? The matching size is at most n by 2, since the number of vertices is n. And so, the number of augmentations is going to be at most n by 2. So, now we know what the total time is, the total time is n by 2 multiplied by this or in other words, it is O f n times m plus n m is typically larger than n and so, we can write it as O of m n. So, that completes the analysis of the algorithm, the description and the analysis of this algorithm.

(Refer Slide Time: 49:29)

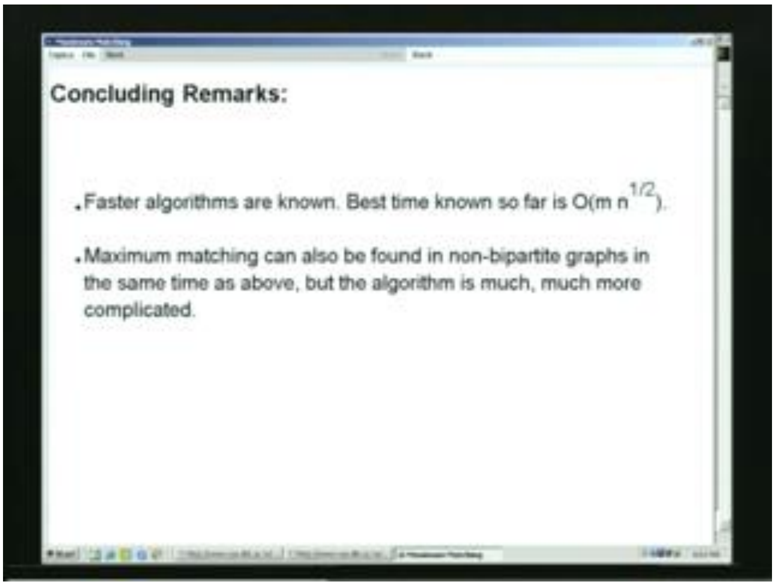


Concluding Remarks:

- Algorithm can be thought of as iterative refinement: we have a matching currently, can we improve it by making a small change? An augmenting path allows us to determine if a small change can be made.

So, let me make a few concluding remarks. We can actually think of this algorithm, as an iterative refinement. What does that mean? Well, we have a matching currently and then, can we improve it by making the small change. So, an augmenting path essentially allowed us to determine if a small change can be made. This is not the fastest algorithm. In fact, an m root n algorithm is known not just m n. So, m root n algorithm is known.

(Refer Slide Time: 50:01)

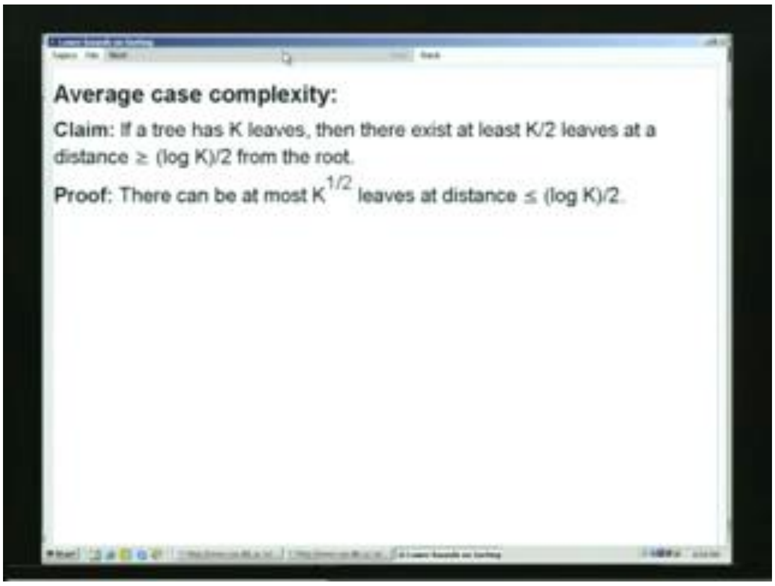


Concluding Remarks:

- Faster algorithms are known. Best time known so far is $O(m\, n^{1/2})$.
- Maximum matching can also be found in non-bipartite graphs in the same time as above, but the algorithm is much, much more complicated.

And in fact, we can define this problem for non bipartite graphs. So, that also turns out to be useful very often. And as it turns out that, similar bounds can be found for the non bipartite case. So, maximum matching can also be found in non bipartite graphs.

(Refer Slide Time: 50:26)



Average case complexity:

Claim: If a tree has K leaves, then there exist at least K/2 leaves at a distance ≥ (log K)/2 from the root.

Proof: There can be at most $K^{1/2}$ leaves at distance ≤ (log K)/2.

In the same time as above, but the algorithm is much, much more complicated. And I will stop here.