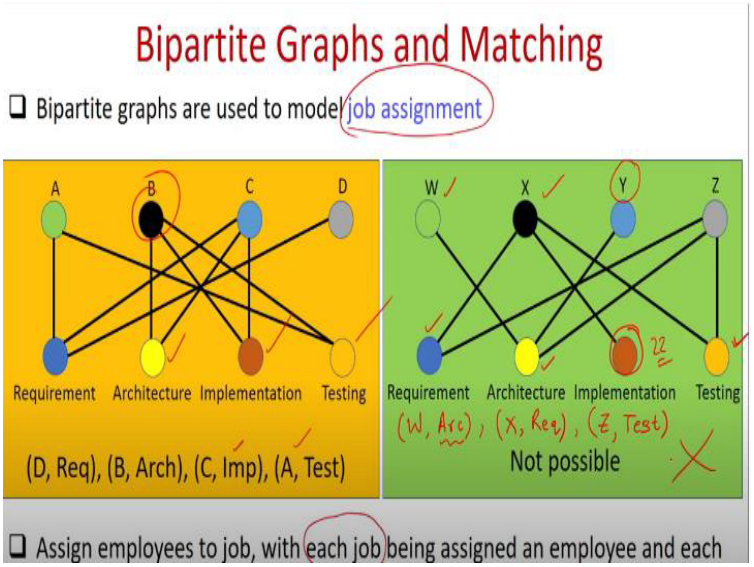


So, let us begin with bipartite graphs and matching. So, just to recap in the last lecture we discussed what are bipartite graphs? They are the graphs where the vertex set can be partitioned into two disjoint subsets V_1 and V_2 where for each edge in the graph one of the end points is into the subset V_1 and the other end point is in the subset V_2. And it turns out that bipartite graphs are very important class of graphs used to model various real-world problems.

And one of the problems for which they are used extensively is that of job assignment. So, let me demonstrate the job assignment problem with this example. Imagine you have two organizations, organization 1 and organization 2. Both of them are trying to build a software and say the software has four modules. It has a Requirement module, Architecture module, Implementation module and Testing module.

And say both organizations have their respective employees. Organization 1 has employees A, B, C and D. Each of these employees can handle one of the four modules required in the software and the skill set of each employee is modeled by adding an edge between the node representing that employee and the node representing the corresponding skill set which that employee can handle.

So, for instance this edge here denotes that the employee A is capable of handling Requirement and this edge from A to Testing denotes that he is capable of handling the Testing module as well. In the same way the employee B can handle three of the modules. It can handle Architecture, it

can handle Implementation and it can handle Testing and so on. Similarly, the edges are added in this graph representing the skill set of the employees in the organization number 2.

Now the job assignment problem is the following. Since we have to build a software, we have to ensure that each of the four modules namely Requirement, Architecture, Implementation and Testing are taken care. So, we want to assign employees to do the jobs as per their skill set such that each job is attended by some employee. But at the same time we do not want to be very strict or unfair with an employee by assigning multiple jobs as per its skill set.

So, for instance I do not want to do the following. I take the employee B and assign Architecture, Implementation and Testing all three to employee B. That I do not want to do. So, I want to assign each employee at most one job. When I say at most one job that means it is fine if I do not assign any job in my job assignment or when I am assigning job to the employees to a particular employee it is fine.

I just want to ensure that each job goes unattended and at the same time I do not assign multiple jobs to any employee when I am doing the job assignment. So, that is my job assignment problem. So, it is easy to see that one of the possible job assignment in organization 1 is the following. I take or I ask employee D to take care of the only job that it is capable of doing namely Requirement and then I take the employee B.

So, employee B can handle Architecture, Implementation and Testing. So, I choose architecture for employee B. I pick employee C and assign the Implementation module to employee C and then I take the employee A and assign the Testing module to employee A. So, I can come up with one job assignment in the organization 1 satisfying this criteria. But if you take the second organization it is not at all possible to come up with the job assignment in organization 2 satisfying these conditions. Why so?

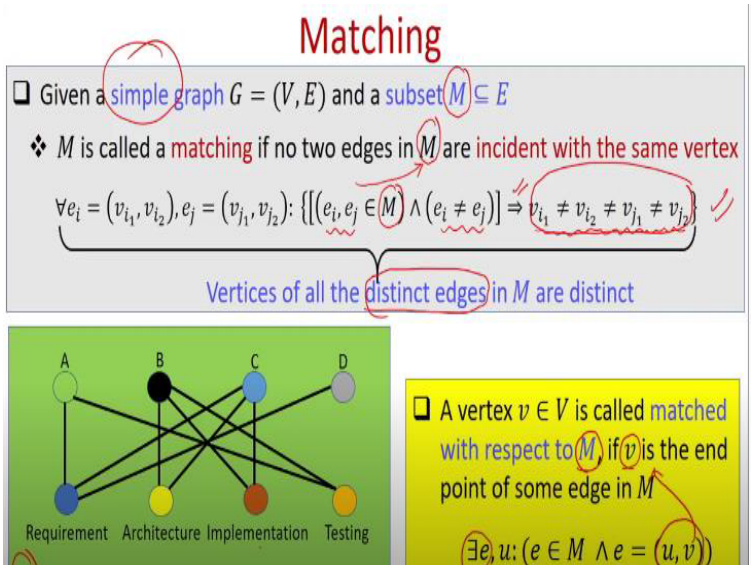
So, let us try to come up with one possible job assignment and then we will see that we will get stuck here. So, for instance let us start with W and W is capable of only handling Architecture. So, I can say W is assigned Architecture. So, we have taken care of W and Architecture. Then we go

to X. X can handle Requirement, Implementation and Testing. So, let us assign the Requirement module to employee X.

Now let us go to employee number Y. Employee number Y can only handle Architecture. But I have already assigned the Architecture module. So, no need to again assign it to any other employee. So, I can say let us not assign any job to employee Y and then if I go to employee Z he can handle Requirement, Architecture and Testing. Requirement and Architecture are already taken care then I can assign only Testing to employee Z.

But in this whole process in this scheduling the module Implementation is not assigned to any employee. It goes unattended. And you can try other possible combinations as well. You will see that irrespective of the way you try to assign the jobs you cannot come up with a job assignment satisfying these conditions. You cannot do that in this graph.

(Refer Slide Time: 06:52)



So, whatever we have discussed in this example can be modeled by a concept called matching. So, imagine you are given a simple undirected graph. That is important. Then a collection of edges denoted by M is called a matching if no two edges in this collection of edges M are incident with the same vertex. Formally what I am saying here is the following. You take any pair of edges $(e_i, e_j) \in E$, if they are different then all the four end points, two end points of e_i and the two end points of e_j they should be distinct, they should be different. I stress here this condition $e_i \neq e_j$ is

important here in this implication. Because if I take the same edge e_i and the same if I take the edge $e_i = e_j$, of course, the first one of the endpoints of e_i will be the same as one of the endpoints of e_j and so on. So, this inequality would not hold if my $e_i = e_j$.

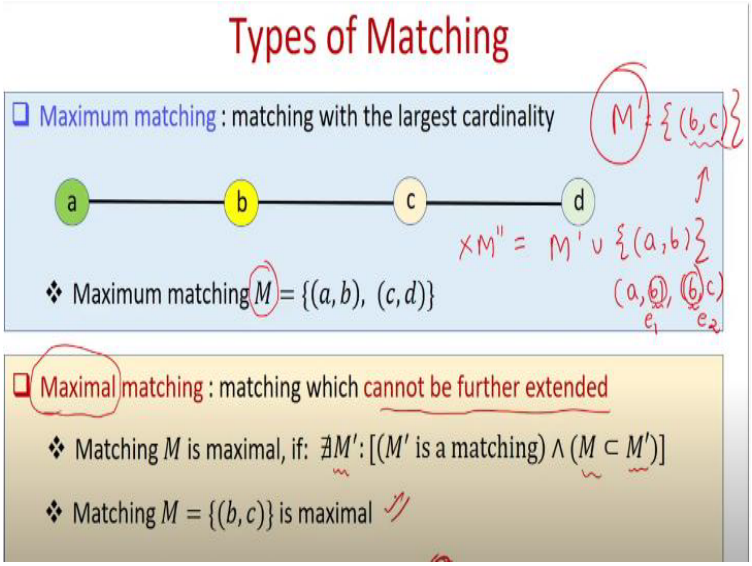
So, this condition this implication should hold for every pair of distinct edges e_i and e_j from my collection of edges that I have picked. I stress here that there is no restriction on $|M|$. I can just pick one single edge from my edge set that will constitute a matching or I can pick two edges from my edge set and if they can satisfy this implication that will also be considered as a matching and so on.

So, what this implication means on a high level is the following. If I take the vertices of all distinct edges in my collection of edges M they should be distinct. So, again if I take the job assignment problem and organization number 1, what I actually did there was I picked the matching. My matching was this collection of edges namely I took the edge (D, Requirement), (B, Architecture), (C, Implementation) and (A, Testing).

And you can check here that you pick any $e_i \neq e_j$, which are distinct in this collection of edges M, their four end points will be distinct. So, if I have a matching M then a vertex v in my graph will be called matched with respect to that matching if the vertex v is the end point of some edge in my matching otherwise the vertex v is called unmatched. So, formally this means that there should exist some edge small e and some vertex u such that the edge $(u, v) \in G$ is there in my graph and that edge is nothing but the edge e.

If that is the case then I will say that the vertex v is matched. In fact the vertex u is also matched with respect to the matching if I have an edge of the form (u, v) if I have an edge of the form (u, v) in my collection of edges M that I have picked.

(Refer Slide Time: 10:17)



So, for instance if I take this graph here and if this is my matching M then I have vertex number A matched. Because there is an edge with one of its endpoints being A. I also have vertex B matched with respect to M because I have an edge with B as its endpoint and so on. Now we have different types of matching. So, we have what we call as maximum matching and it is a matching which has the largest cardinality.

That means it is the collection of edges whose cardinality is the largest. That means you cannot have a matching whose cardinality is bigger than the collection of edges of the matching that you have picked. So, if I take this graph and the maximum matching here is the collection of edges (a, b) and (c, d). Its cardinality is two, whereas if I take the matching M' equal to (b, c) then that is not the maximum matching. Why it is not the maximum matching?

Because the cardinality of $|M'| = 1$. By the way the matching M', this collection M' is indeed a matching because the end points are distinct here. You do not have first of all the second edge here. So, if you just take any edge by default it is a matching. So, this is a matching, M' is a matching and it is not a maximum matching because you have only one edge in M' whereas you can have another matching M which has two edges in it.

So, M' is not a maximum matching here. However, the matching M' is maximal matching. So, there is a difference between maximal matching and maximum matching. So, what is the maximal

matching? A maximal matching is a matching which cannot be further extended and what do I mean by cannot be further extended? By that I mean that you cannot find another matching M' such that M' is a superset of your matching M.

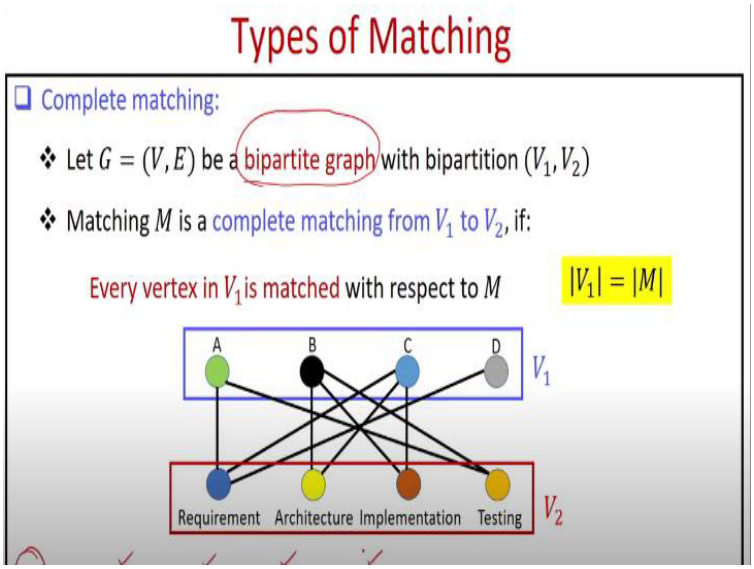
Because if you can find another matching M' which is a super set of M that is equivalent to saying that you can extend, that means you can add further edges in M and still get a matching of bigger cardinality. That is what is the interpretation of cannot be further extended in this definition. So, that is why your matching M' is a maximal matching because you cannot add any other edge now in M' and still get a new matching. Why so?

Because I cannot add the edge (a, b) in M', because if I add the edge (a, b) to get say M' and M'' is not a matching. Because in M'' you will have the edges (a, b) as well as (b, c) and you do not have all the endpoints of any pair of distinct edges distinct. So, here you have the edges e_1, e_2 such that one of the endpoints of edge e_1 is same as one of the end points of the edge e_2 which goes against the definition of matching.

So, that is why M' here in this example cannot be further extended and hence it is a maximal matching. Whereas if I take the matching (c, d) in this graph it is not maximal. Why it is not maximal? Because I can further extend my matching M and get a bigger matching, namely I can add the edge (a, b) in this matching and it will still constitute a bigger matching here. So, that is why there are two types of matching here, maximum matching and maximal matching.

Of course, every maximum matching is a maximal matching but other way around need not be true. You may have a matching which is just maximal but it is not maximum and one example is this. A matching (c, d) is the maximal matching. Sorry, one example here is this the matching (b, c) it is a maximal matching but it is not maximum matching.

(Refer Slide Time: 15:02)



Now we have another kind of matching called as complete matching and what is a complete matching? So, imagine you are given a bipartite graph with bipartition (V_1, V_2) then a matching in this graph is a complete matching from $V_1 \rightarrow V_2$. And when I define a complete matching it is very important whether the complete matching is from $V_1 \rightarrow V_2$ or from $V_2 \rightarrow V_1$. They will be different in general. So, here I am giving the definition of a complete matching from $V_1 \rightarrow V_2$.

So, the matching M that I have picked is called a complete matching from $V_1 \rightarrow V_2$ if every vertex in V_1 is matched with respect to my matching M or equivalently $|V_1| = |M|$. This is because if every vertex in V_1 is matched that means I have edges in my matching M where for every vertex small v in V_1 there is a corresponding edge with v as its end point in my matching M.

And as per the definition of a matching I cannot have any other edge as well present with the same endpoint v. That is not possible if I already have an edge with v as its end point in my matching M. I cannot include any other edge in my matching M different from this edge which has v as its end point. That automatically means that the number of edges in my matching M that I have picked should be exactly same as the number of vertices in my subset V_1.

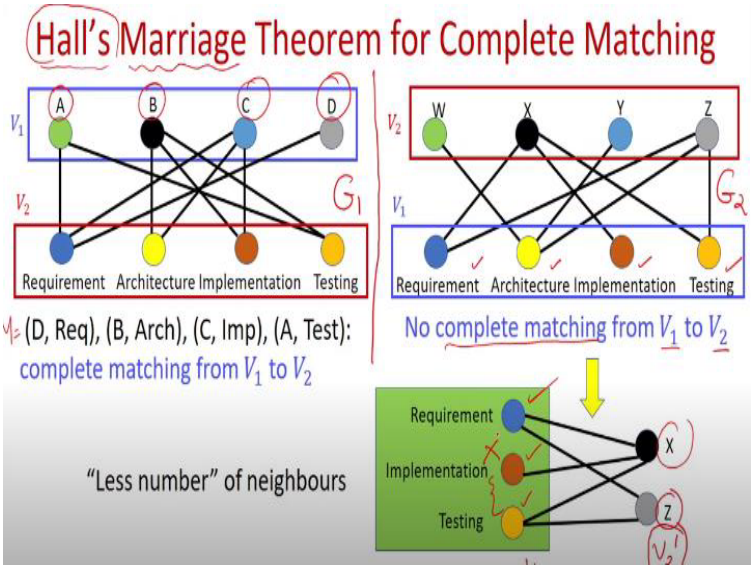
If that is possible if that is the property of my matching then I will say that my matching M is a complete matching from $V_1 \rightarrow V_2$. Whereas if I want to find a complete matching from $V_2 \rightarrow V_1$ then the matching should be such that its cardinality should be exactly equal to the cardinality of

V_2. Namely it should be ensured that every vertex in V_2 is matched and that is possible only if the $|M| = |V_2|$

So, for instance if I take this bipartite graph and it is easy to see that this is a bipartite graph because I can put A, B, C, D in one subset and I can put requirement, architecture, implementation and testing in another subset. And this is a bipartite graph but it is not a complete bipartite graph and when I am defining complete matching I do not need a complete bipartite graph.

It is defined with just bipartite graph. Why this graph is not a complete bipartite graph? You have many missing edges here. So, for instance I do not have the edge from D to testing and so on. This edge is not there. So, if this is my V_1 and V_2 and if I pick this matching M then it constitutes a complete matching from $V_1 \rightarrow V_2$. Why it constitutes a complete matching from $V_1 \rightarrow V_2$? Because all the vertices in V_1 they are matched, A is matched, B is matched, C is matched, and D is matched. In fact, in this particular example this matching is also a complete matching from V_2 to V_1 as well. Because I can interpret this matching as if it covers each and every vertex in the vertex set V_2. So, requirement is covered or matched, architecture is matched, the node implementation is matched and the node testing is also matched.

(Refer Slide Time: 19:06)



So, now let us talk about a necessary and sufficient condition for checking whether there exists a complete matching in my given bipartite graph or not. So, if I am given a bipartite graph and if I

want to check whether there exists a complete matching from V_1 to V_2 or a complete matching from V_2 to V_1 there should be some procedure for doing that.

Can I say that there exist some graph theoretic properties by checking which I can declare whether the given graph has a complete matching or not? And there exists a very simple condition that condition is also called as Hall's marriage problem. If you are wondering why the term marriage is coming here, because the condition is given in the context of a bipartite graph where the vertex set V_1 and V_2 corresponds to the set of boys and the set of girls and the edges basically in the graph represents the preferences that the boys have for the girls.

So, if some boys prefer some girls and vice versa you will have the corresponding edges between the vertex sets in V_1 and V_2. And we would like to find out a complete matching from $V_1 \rightarrow V_2$ in the sense that we would like to ensure that each boy gets a wife or a bride with respect to the preferences that he has and we want to ensure that all the boys are covered. So, that is equivalent to saying that I want to find out the complete matching from the set $V_1 \rightarrow V_2$.

So, that give the necessary and sufficient condition for that problem and that necessary and sufficient condition is also called as the Hall's marriage condition. So, I have given here two graphs. So, this is my graph number G_1 and this is my graph number G_2. The interpretations of V_1 and V_2 are different in G_1 and G_2. In G_1 the vertex set V_1 are my employee set and the vertex at V_2 are the skill set. Where in my graph G_2, I have made the skill set as the vertex set V_1 and employee set as the vertex set V_2.

So, you can see that in the graph G_1 I have a complete matching from V_1 to V_2. That means I can ensure that all the nodes in $V_1 = \{A, B, C, D\}$ are matched with respect to the matching that I have formed here. Whereas if I take the graph G_2 then I cannot have a complete matching from V_1 to V_2. Now why it is not possible to have a complete matching from $V_1 \rightarrow V_2$? And we already saw when we discussed the job assignment problem that in this specific graph G_2 it is not possible to do the job assignment.

So, recall the job assignment problem was that we want to ensure that each of the four modules namely requirement, architecture, implementation and testing are taken care by some employee. But at the same time I do not want to assign more than one job to any employee. That problem can be modeled as if you want to find out a complete matching from $V_1 \rightarrow V_2$. Because if you want to find a complete matching from $V_1 \rightarrow V_2$ that is interpreted as if you want to pick a subset of edges in G_2 such that no two distinct edges in that subset has a common endpoint.

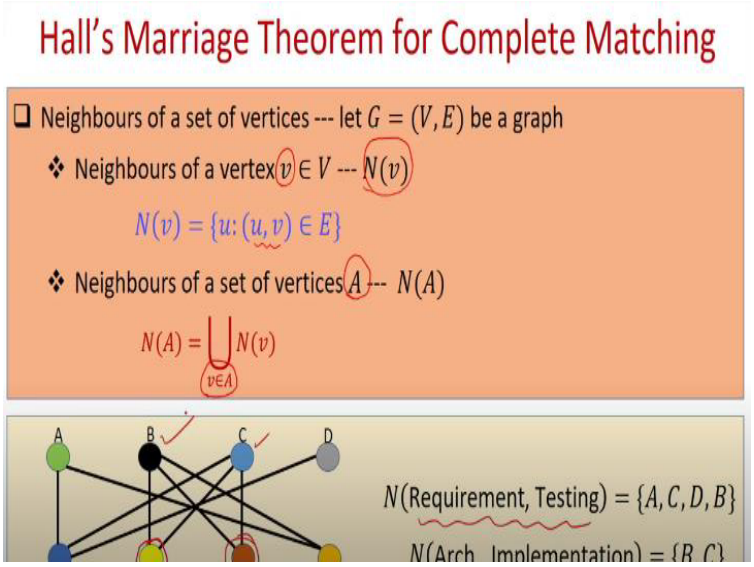
That is equivalent to saying that there exists no employee who is assigned multiple jobs and at the same time you want to take care of all the four modules. So, that is why complete matching from $V_1 \rightarrow V_2$. I do not want to cover all the employees, it is fine if some employee is not assigned any job. My goal is to assign or ensure that each job is taken care. So, that is why the complete matching is from $V_1 \rightarrow V_2$ and it turns out that in graph G_2 there is no complete matching from $V_1 \rightarrow V_2$.

So, if you are wondering that why a complete matching is possible in graph G_1 but why a complete matching is not possible in G_2, what exactly is the difference? What exactly is the source of problem in graph G_2? Well the source of the problem in the graph G_2 is this specific sub portion of the graph. If you take this specific sub portion of the graph then we have three modules namely the requirement module, implementation module and testing module and there are only two employees who can take care of these three modules. So, in terms of graph theoretic properties I have a subset V_1' here and a subset V_2' here such that the number of neighbors of V_1' is less than the number of nodes that you have in V_1'. So, the number of neighbors all together is two but you have three nodes in V_1'. So, how can it be possible that you take care of requirement as well as implementation as well as testing?

And at the same time ensure that neither the employee X nor the employee Z is assigned more than one job. That is not possible because if at all requirement is to be taken care it can be taken care only by either the employee X or by the employee Z. Once you assign it to employee X, say for example, then you cannot assign employee X any other job. So, you will be now left with two more modules to be taken care. But now you have only one employee left. How can that be possible? You will be able to only assign testing to employee Z and now the module implementation will go unattended. So, that is the source of problem in graph G_2 which is not

allowing you to find out the complete matching from V_1 to V_2 and that is precisely the formal statement on the necessary and sufficient condition given by the Hall's marriage theorem.

(Refer Slide Time: 26:03)

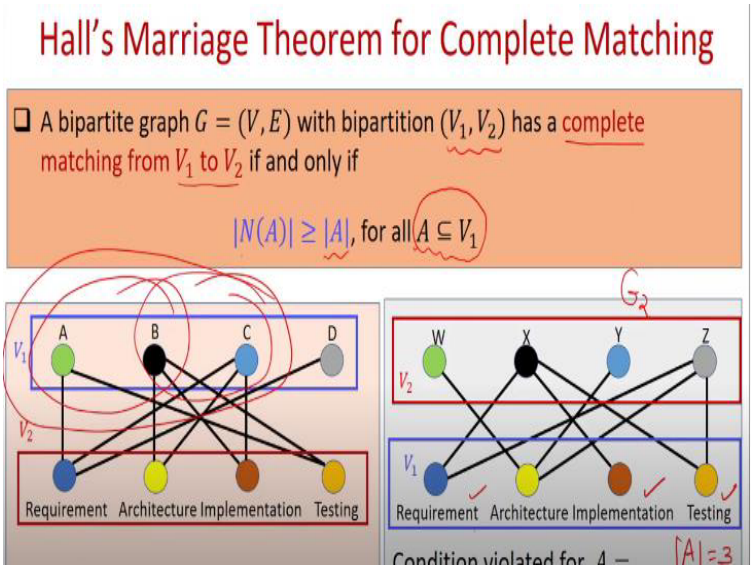


So, to understand the necessary and sufficient condition given by the Hall's marriage theorem, let us first introduce a few definitions here and notations. So, if I have a graph G then the neighbors of a vertex v, small v is denoted by this notation N(v) and what exactly is the neighbor set of the vertex v? $N(v) = \{u: (u, v) \in E\}$W. So, it is basically saying that the neighbours of vertex v are all the vertex u such that u is one of the end points of an edge incident with the vertex v. Of course, if there is a self loop at the node v then v also will be considered as its own neighbor. And once we have the definition of a neighbor of a single vertex then the definition of a neighbor of a vertex at A is the following. So, if A is a set of vertices that the neighbor of this set of vertices in A is basically the union of the neighbors of all the individual vertices in that.

So, for instance if I have this bipartite graph well by the way the neighbor and the neighbor of a set of vertices is defined with respect to any graph but we will be using it in the context of bipartite graph and we will be discussing the Hall's marriage theorem. So, if I consider this bipartite graph and then if this is my set of vertices then its neighbor set is A, C, D and B. Why? Because the neighbor of the vertex requirement are C and D and the neighbors of testing are B and A. So, if I

take their union, I get the set $\{A, C, D, B\}$. Whereas if I take the subset {architecture, implementation}, so the neighbor of architecture is only C and neighbor of implementation is B.

(Refer Slide Time: 28:34)



So, what exactly is the necessary and sufficient condition given by the Hall's marriage theorem for complete matching? It states the following; the Hall's marriage theorem statement is the following. If you are given a bipartite graph where (V_1, V_2) is the bipartition then for complete matching from V_1 to V_2 you need the following necessary and sufficient condition. You need the following. You take any subset $A \subseteq V1$.

That subset could be an empty subset,ϕ. It could be a subset with just one vertex or it could be the whole set V_1. You take any $A \subseteq V1$., the number of neighbors of that subset A should be greater than equal to the number of nodes in the set A, $|N(A)| \geq |A|$, for all $A \subseteq V_1$. Of course, the neighbors of the set A will be in the subset V_2 because that is the definition of your bipartite graph. You cannot have the neighbors of the subset A within the vertex set V_1 itself.

The neighbors of A will be in the other subset in your bipartition namely V_2. So, what it says is you want to find out the complete matching from V_1 to V_2, this is a bipartite graph and if you have any subset A here, it should have either the same number of neighbors in V_2 as the number of nodes in A or more neighbors. It cannot have less number of neighbors in V_2 and that precisely

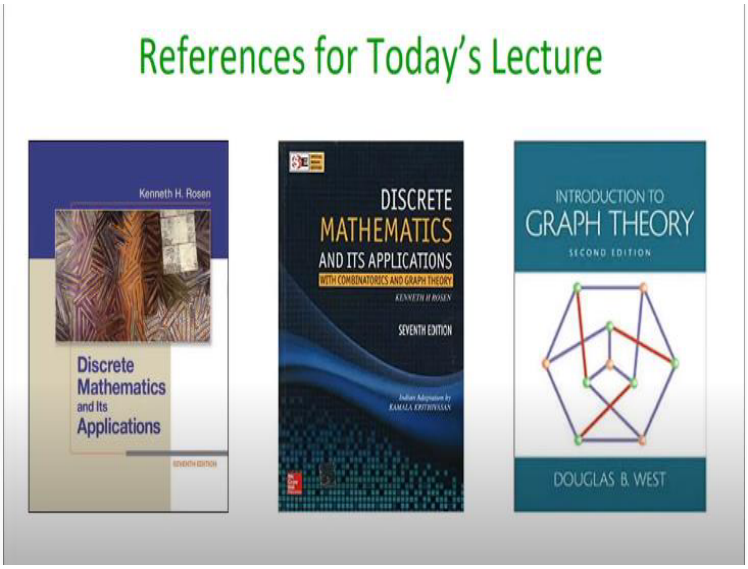
was the source of problem due to which we do not have a complete matching in the graph of organization number 2.

So, if you take this graph this was the graph for organization number 1 and in this graph we are able to find out a complete matching from the vertex set V_1 to vertex set V_2. This is because you take any subset of V_1, you take either {A, B} or you take {B, C} or you take {A, B, C}, you take any subset. That subset has either the same number of neighbors as the number of nodes that in the subset that you are taking or more number of neighbors in the other subset in your bipartition.

But if you take this graph, graph G_2 then this condition is violated. Specifically, if you take your subset to be {requirement, implementation, testing} namely A is equal to 3. Then the cardinality of its neighbor set is what? So, requirements neighbor are X and Z, implementation neighbors is only X and testing neighbor is only X. So, the number of neighbors all together for your subset A that you have chosen here is 2. So, you have three nodes to be covered or matched namely requirement, implementation and testing. But there are only two employees namely X and Z who can handle those three modules.

So, the only way that you can ensure that all the three modules requirement, implementation and testing are taken here is that you assign more than one job to employee X or Z which goes against the definition of complete matching. So, we will prove this condition or the theorem Hall's marriage theorem in our next lecture.

(Refer Slide Time: 32:37)



I conclude today's lecture. These are the references used for today's lecture. To summarize, in this lecture we introduced the notion of matching. We saw various types of matching like maximal matching, maximum matching and complete matching and we saw the necessary and sufficient condition for the existence of a complete matching in a bipartite graph namely the Hall's marriage theorem. Thank you!