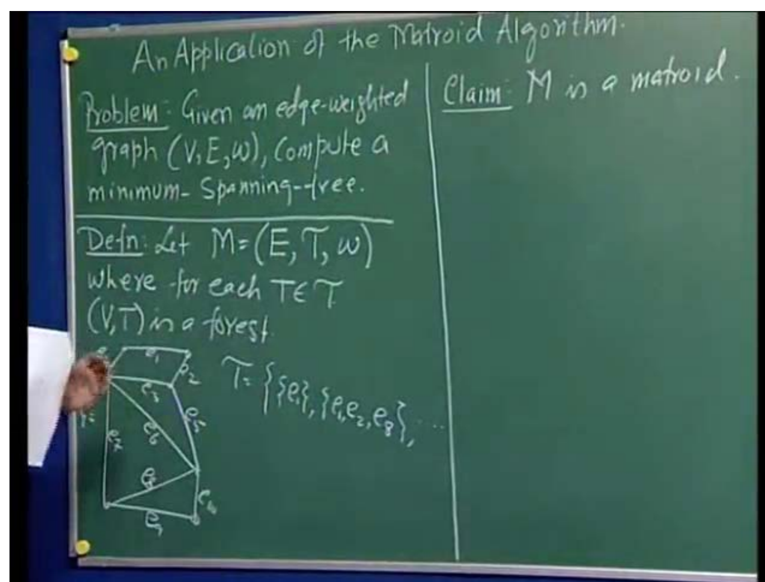
Computer Algorithms-2
Prof. Dr. Shashank K. Mehta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 6
Minimum Spanning Tree

Hello. Today, we will discuss an application of the Matroid Algorithm that we had studied in the last lecture.

(Refer Slide Time: 00:24)

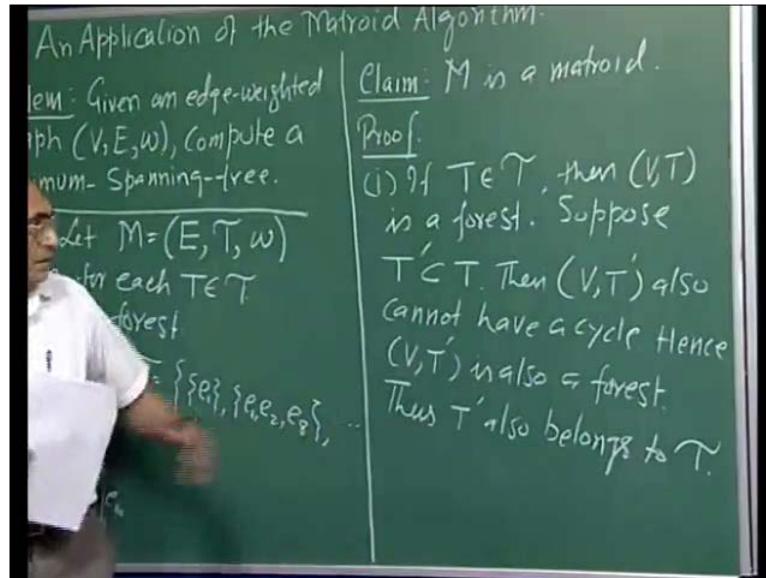


We had described the problem, which was, given an edge weighted graph compute a minimum spanning tree. So, we want to compute a spanning tree of a given graph whose cumulative edge weight is the least. Now, let us first show that this problem can be converted into a Matroid problem. So, let us define a Matroid as follows. So, let M be this re couple, where E is the edge set of the graph, T is the subsets of E, which we will describe in a minute and w is the same weight function on the edges. That was given in the graph, so where the members of key where for each T and T, V T is the forest.

So, for every subset of E along with the vertex set V forms a forest. That is how the script T has been defined. Now, recall that a forest is any sub graph of our graph, which has no cycle. So, for example, here if this is my G, I am ignoring the weight right now. Then some of the members of script T will be, if I label them as e 1, e 2, e 3, e 4, e 5, e 6,

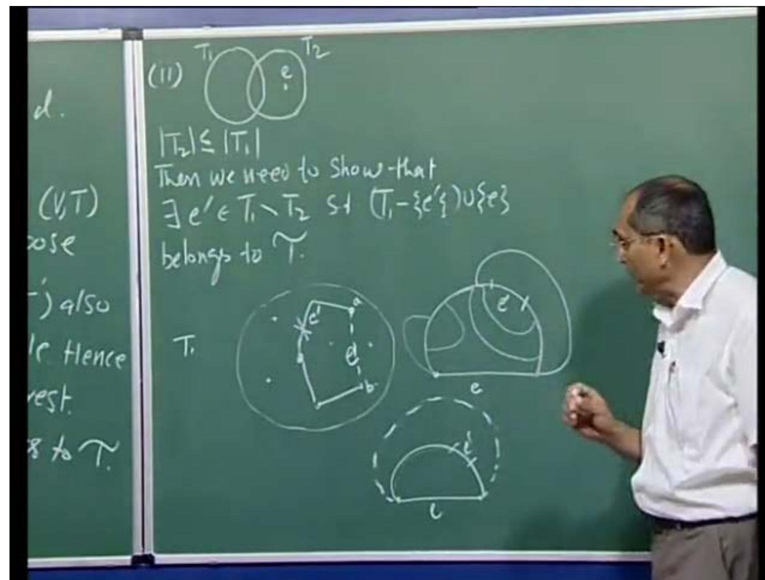
e 7, e 8, e 9 and e 10, then the possible members are, for example, just e 1, maybe e 1, e 2, e 8, there are members. Notice that this one defines a graph containing this edge, this edge and this edge. They do not form any cycle. Hence, they become a member of script T. Now, we have defined all the 3 components of this angle. Our claim is that M is a Matroid. Now, there are 2 properties that M should satisfy to be a Matroid.

(Refer Slide Time: 05:17)



So, first thing is the subset property is, that if a T is a member of script T, T prime is a subset of T, then T prime should also be a member of script T. Now, first notice that if T belongs to script T, then V T is the forest. Now, suppose T prime is a subset of T, well T prime also cannot have a cycle. The reason is that we have only deleted some of the edges from this graph. Hence, we cannot have cycle here. Hence, V T prime is also a forest. Thus T prime also belongs to the script T.

(Refer Slide Time: 07:01)



The second property requires that if we have 2 members of T script T, when V T 1 and T 2 and T 3 is not bigger than T 1 in cardinality. Let us say, there is some element e. There is some edge in T 2, which is not in T 1. Then we need to show that there exists some edge e prime in T 1 minus T 2 such that, T 1 minus e prime union e belongs to script T. This is what we have to establish. So, let us say this forest, say T 1 is symbolically this structure and of course, there are vertices of V in this edge e which is a not present in T 1. So, e is not a member of the graph T 1.

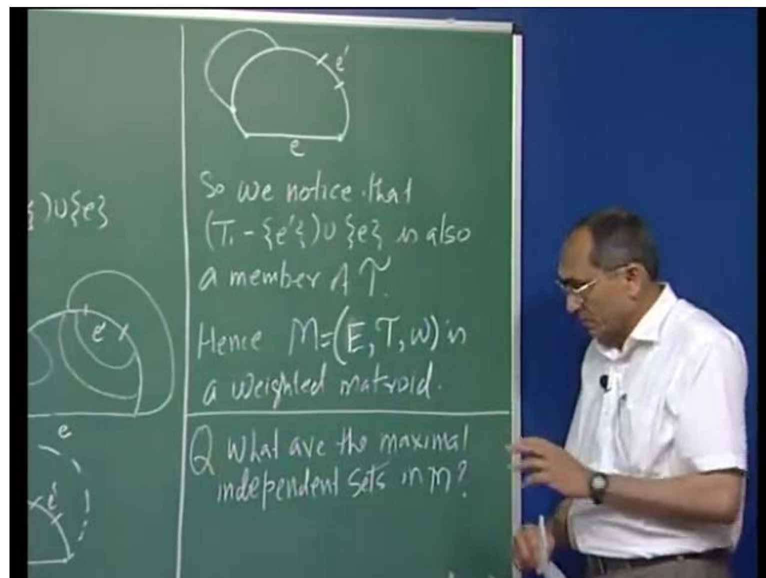
Suppose, we incorporate, we add this edge into T 1 and let us suppose these are the 2 vertices a and b. If we deposit e inside this graph T 1, then maybe a cycle is formed, but notice that without e there was no cycle in the graph. If cycle is not formed, then by deleting any edge from T 1 minus T 2 will also keep it a forest. There is no harm in case there is a cycle formed. Then, probably there is some path. The solid edges are members of T 1 and now by putting e we have formed a cycle.

In this case suppose I delete any one of these edges. Suppose, we delete any one of these edges, call it say e, this is my e prime. Now, I want to show that this again is free from any cycle. So, let us suppose this is still a cycle. There is a cycle in T 1. After incorporating e and deleting e prime, suppose, there is a cycle in that case, that cycle must contain e in it, because without e anywhere there was no cycle in the graph.

So, let us say more symbolically. Here, is my e. We have 1 path which contained e prime in it. Now even after this is deleted, there is a cycle in it. So, that cycle, since it is suppose to pass through e that cycle may have some part here may go through, this may go out this with this way. In other words, that cycle may share some of the edges of the original cycle and may have some other edges outside it. Now, what we have is this second cycle which does not require e prime. Now, in that case let us first of all, visualize a simple situation where, none of the edges of the second cycle are shared with the original cycle.

In that case, it would have been simply e here. This would have been this way. This was e prime and then there was a second cycle sharing no edge with this part. Only the edge e which was a new edge shared. Then, notice that in our T 1 you already have a cycle. Because this does not use e, therefore, it is a cycle of T 1. But, that is not possible because our T 1 is a forest. So, it is not at all possible. So, now we have to see if any edge is shared, what happens to the general situation where the two paths share some of the edges?

(Refer Slide Time: 12:59)



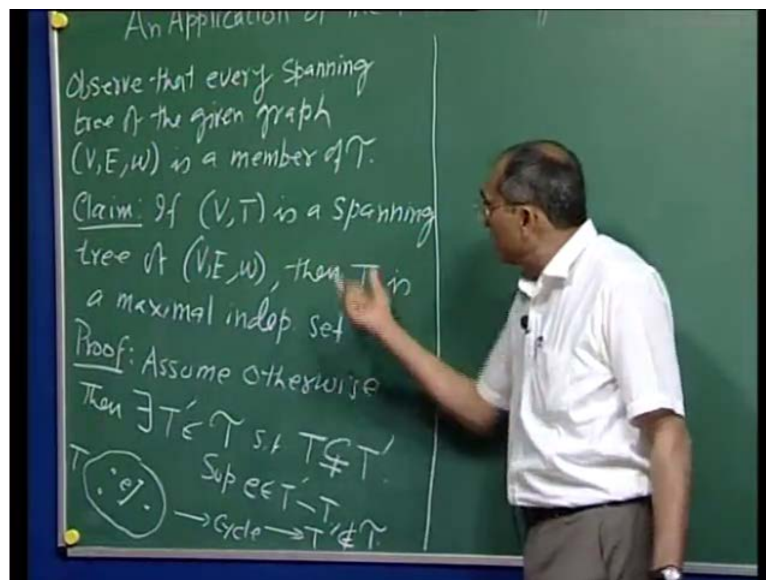
So, let us say this is our new edge e and let us suppose this was the original cycle with this being e prime. Now, the other cycle that also contains e will begin from here and end here. Again, we can visualize the same thing. Now, this cycle when it begins it may continue with the original cycle. Then at some point it splits. Now, starting from here we

proceed until it shares one of the vertices of this cycle. This vertex could very well be this vertex as well. So, now the thing is, if we notice this particular portion is again a cycle which is a cycle of T 1, because all the edges that are involved belong to T 1.

Now of course, if a vertex had been this one, then I would have had this as a cycle in T 1. Once again, this is not possible because our T 1 is given to be a forest. Hence, it is not possible that after deleting e prime there is a cycle there. So, we notice that T 1 minus e prime union e is also a member of script T. Now, we have found that both the conditions of a Matroid are satisfied by M. Hence, E T w is a weighted Matroid. Now, we want to know what are the maximal independent sets in this Matroid?

What are the maximal independent sets? Well, the maximum maximal independent sets are those which have the maximum number of edges, those subsets. Now, if my graph G is connected then an e spanning tree of my graph is a member of script T, is an independent set..So, let us step back to some observation. We observe that every spanning tree of the given graph V E w is a member of T.

(Refer Slide Time: 16:48)

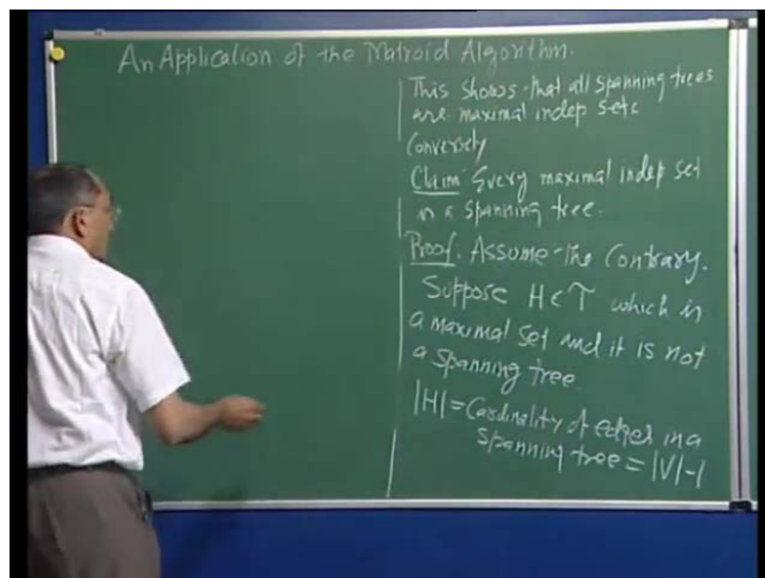


Now, here we are showing that the graph is connected. Now, we claim that the spanning trees are maximal independent sets. Now, if they are not, then a spanning tree must be contained in another member of T. So, claim if V T is a spanning tree of the given graph, then T is a maximal independent set of script T. If this is not maximal, then this must be a proper subset of some member of script T. So, suppose assume otherwise, then, there

exists e prime in script T, such that T is a subset, but, not equal of T prime. Therefore, there is at least 1 edge in T prime not present in T. Now you look at the graph corresponding to T.

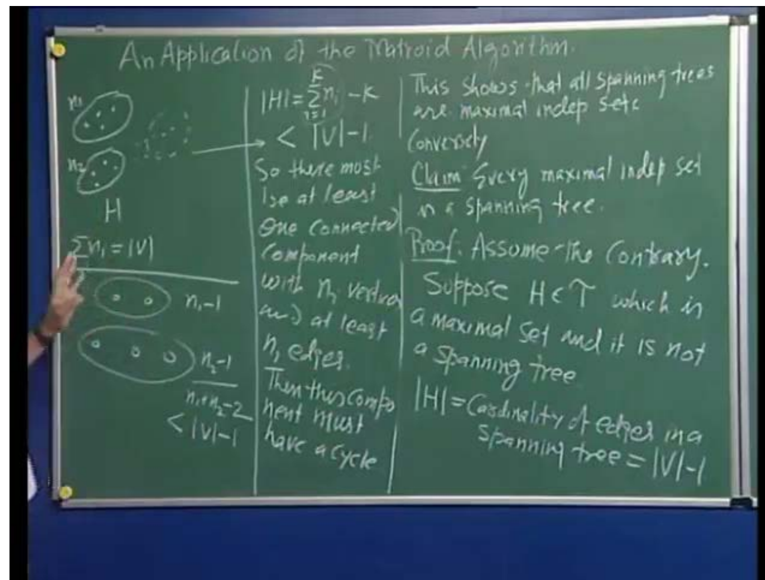
This is my T because it is a spanning tree. Then, there is a part between every pair of vertices in it. Now, suppose there is and suppose E belongs to T prime minus T, so let us say this is E. But E being a spanning tree, there is a path already present from this vertex to this vertex. If I add a G, then I am forming a cycle. If I am forming a cycle here, this no longer qualifies to be a member of script T. Hence, this is not possible. So, we conclude that this leads to a cycle. That means T prime does not belong to script T. This contradiction establishes that spanning trees are maximal members of the script T.

(Refer Slide Time: 20:22)



So, this shows that all spanning trees are maximal members. Conversely, we will now claim every maximal member is a spanning tree. But to prove this, we will use the fact that all maximal independent sets have the cardinality. Now, suppose we assume the contrary. Suppose, we have a member H of T which is maximal set and it is not a spanning tree. If it is not a spanning tree, but it is still a member of this, then there is 1 thing. We can of course, conclude that the cardinality of H is equal to the cardinality of edges in a spanning tree, which is of course, mod V minus 1. Now, if this is not a spanning tree, then there must be at least two components in this graph.

(Refer Slide Time: 23:17)



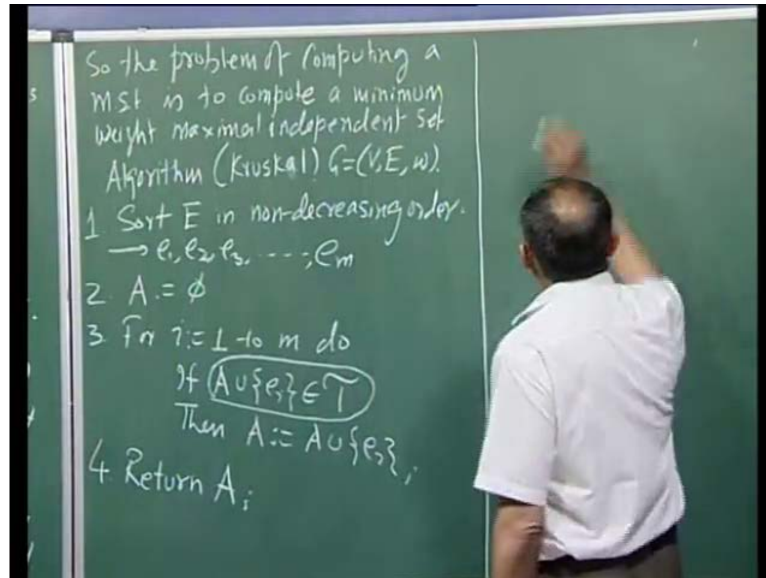
So component means, that H nears at least 2 parts, may be more. So, entire H may be other parts. This is how it looks like. Then the number of vertices may be let us suppose n 1, n 2 and so on in each of the connected part. Notice that, by connected component this means that there is no edge going from here to this or there is no path from here to here or from here to any other component, there is no edge going to outside, there is no edge going outside this and so on. So, the sum of n i is of course, V and the number of edges is mod V minus 1.

So, in this case let us suppose, we have 2 components say n 1 is 2 and n 2 is 3 and then there are 4 edges in it. If there are n 1 minus 1 edges in this and n 2 minus 1 edges in this, then this will add up to at least n 1 plus n 2, at most. So, if we have n 1 minus 1, n 2 minus 1 edges at most in these, then this will add up to this which is less than mod of V minus 1. So, in general let us suppose, in this picture if the number of edges is at most n 1 minus 1, at most n 2 minus 1, and so on. Then, number of edges will be H, will be summation n i, i going from one to k, their k components n 1 minus k, which is strictly less than mod of V minus 1, because, this mod of V, the total number of vertices.

Hence, this is not possible. So, there must be at least 1 connected component with n i vertices and at least n i edges. If there are n i vertices and n i edges and if the graph is connected, then this cannot be a tree. Then, this component must have a cycle, which implies that such a graph H must be a spanning tree. There is no other way out of this.

So, we have now concluded that every maximal number of script T is a spanning tree and only spanning trees are the maximal independent sets. With this, now the problem of computing minimum spanning tree reduces to the following.

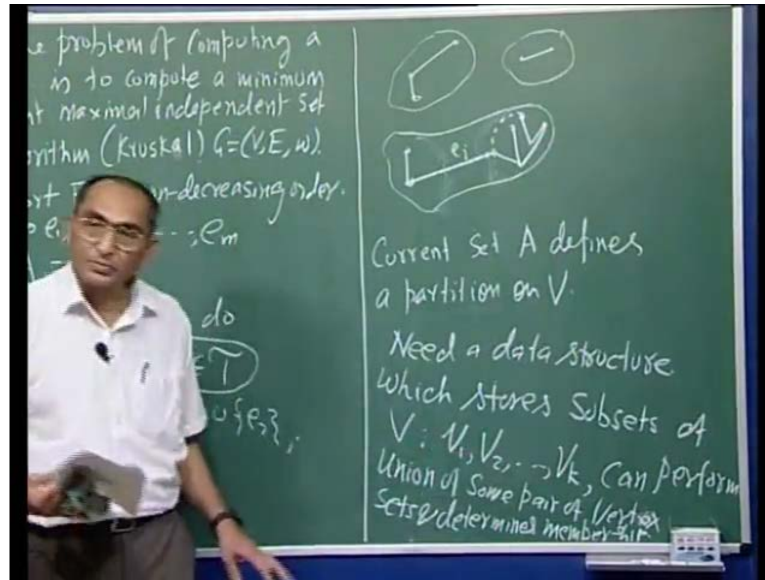
(Refer Slide Time: 27:29)



So, the problem of computing a minimum spanning tree that is m s t is to compute a minimum weight maximal independent set, for which we know an algorithm. Now, we can write down our algorithm and this algorithm is due to Kruskel. So, first step to sort our given graph is G equal to V T and weight W. Sort E in non decreasing order. Note that this time we are trying to compute a minimum weight maximal independent set. We have to pick the least weight edge first, which gives me e 1, e 2, e 3 and then we set our A to be an empty set. Now for A, let us say there are n edges. So, for i equal to 1 to M do, if a union e i belongs to script T, then and finally, return A i.

This is our original algorithm. In this particular instance, this becomes the way for computing n M S, but, of course,, we have this problem that we have to figure out how to test the membership of T. The criterion for this to be a member of T is that this should be cycle free. That is necessary and sufficient condition for this G to be a member of T. Now, let us try to see what does it involve? If you notice, the current A e itself is a member of T. So, this is an H set which does not form a cycle in the graph currently.

(Refer Slide Time: 31:15)



So, A may look like the set of edges due to A, will look like this. Now suppose a new edge e i comes, if this edge is running from one component to the other. Then we can be sure that this is not going to create a cycle. The reason is that there is no path from here to here. Hence, this edge if there was a cycle, then there should have been a path from here to here. So as long as the 2 end points of e i are into different components, this is the safe bed. On the other hand, if the 2 end points of e i are in the same component. So, for example, this then we are sure that there will be a cycle. The reason is this is a connected component. So, from the original set of edges, there is a path from here to here. We have an another edge and we are incorporating, so that closes a cycle.

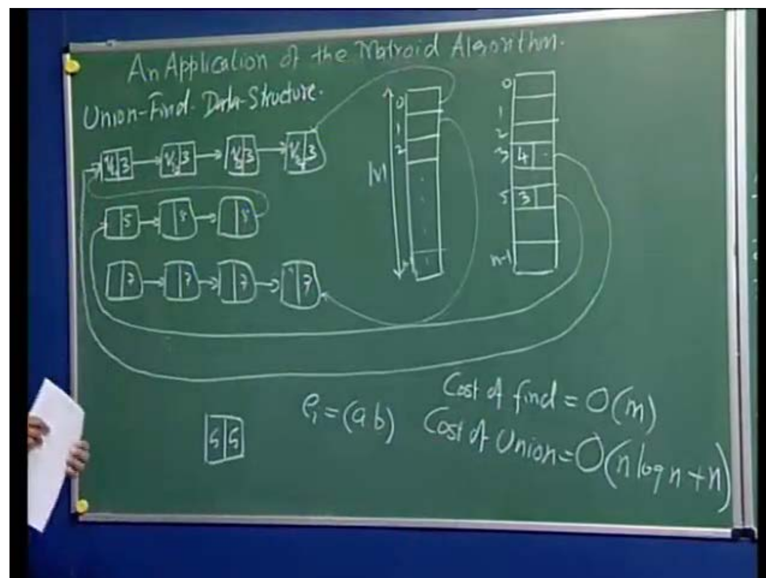
So, the necessary and sufficient condition for this test is that the 2 end of vertices of e i must belong to distant components in the graph V A, where A is the current set of it. This problem is the problem of union. What we will do, is we will visualize the set of vertices in 1 component as one of the subsets of a partition. So, current set A defines a partition on V, the vertex set. Each connected component is going to give me 1 set of the partition. These are the vertices in that set, the second one contains these 2 vertices, the third one contains these vertices, and so on. So, we have to have a data structure which can store a partition of V.

When asked whether the given 2 vertices belong to the same partition or 2 different partitions, it should be able to find out the answer to that question. Once it finds out,

suppose this is the new edge, it finds out that the 2 end vertices of e i belong to different partitions. Then I will incorporate A into e i. Then, this will become a member of A and the new partition will be the result of clubbing these 2 parts. I will have to combine these two. So, now we need a data structure which stores subsets of V, namely V 1, V 2, V k. These are the vertices corresponding to various parts of this partition.

Their union is V. Their intersection is empty. Because, it is a partition it can perform the union of some pair of vertex X. Further, given the vertex, it can identify which part that vertex belongs to, the membership in the 2 part and determine membership. These are the tasks that this data structure has to do; it should be able to store a partition. It should be able to perform union of any pair and then, given the vertex it should be able to tell us which part such that the vertex belongs to. So, now such data structures are meant for so called union-find problem.

(Refer Slide Time: 37:36)



This particular problem is known as union-find, performing union and determining the set to which a given vertex belongs, that is union data structure. So, there are efficient data structures for this. I will give you one example of which it is not as efficient as another well known data structure, but, it is an easy to understand. So, what we will do is in this case we will store each partition into a link list and so on. So, here we have vertices of capital V 1, here we have vertices of capital V 2 and so on. When we have to

perform union, all we have to do is connect an edge pointer from end of one to the start of the other. That is how we will be able to perform a union.

Now, we will do 1 more thing that decides storing the vertex identity. So, we have some vertex identity. We have vertex small v i 1, v i 2, v i 3 and so on. We will also store the identity of this set. So, let us say the set identity is 3. It could be any single number. The set identity of this is a five, this maybe seven. So, we need some supporting structure. What we will do is, we will have 1 array with size mod of V. So, this is vertex. This is for vertex 0, this is for vertex 1, vertex 2 and n minus 1. This will have a pointer through the corresponding node in the link list. So, this is say vertex V 0, V i 4 is V 0. Maybe, V 1 is somewhere here and so on.

We will store a pointer in this to the node of the corresponding vertex. In addition to this, we will have another array where we will have the same size and these will be the identifiers of the set. So, we have 1, 2, 3. So, this will be pointing to the start of link list 3, corresponding to 3. Maybe this is 5, so this will be pointing to 5 and so on. In addition we will also save here this size of the lists. So, for example, here we will store. So, there are 2 parts. Here is the pointer and here is the number of nodes in that list. Over here, it is 3 and so on. This is the information we will store in this data structure.

So, initially what will happen is that all the vertices, because initially we have empty A, so each individual vertex will form a connected component. There are no edges inside it. Each link list will contain only 1 element. If the element is, say 5, then we will also name set number is 5. We will keep the same number. Each of these will be pointing to 1 link list containing only 1 node and initial cardinalities of the lists will be 1. When we have to perform a union we will first look at the 2 sets that we have to perform the union on. The 2 sizes that we have, we will pick the smaller one.

We will walk through this and re-label this by the label of the larger one. So, suppose we have to perform the union of these 2, the label of the larger one is 3 and label of the smaller one is 5. We will walk through this. We will re-label them as 3, come to this point and then from a pointer from here to here. That way we will have now a single link list containing these elements. We will do the necessary book keeping here. We set this to 0. Set this pointer to now and this pointer will be now pointing to. That way, the union

will be performed. It is very important to keep in mind that we will always re-label the smaller set, a smaller link list.

Now, as far as find is concerned, if I want to find out which set does element 1 belong, I will just go to the node and we will look at the set identity, which is this and this will tell that which set does 1 belong to. It belongs to set identified with 7. So, these 2 operations can be performed in this data structure. When a new edge e I, which has 2 and vertices a and b is to be incorporated in our set, we will check which set a belongs to, and which set b belongs to. If they have the same identity, then we will drop it because in that it is going to form a cycle.

If they belongs to 2 different sets, then we will actually go to the corresponding sets, perform their union and update the data structure. This is all we have to do. Now, in this data structure, what is the cause of find and union? For every edge, we are going to find the sets of the 2 vertices and then we are going to perform union. If these are 2 distinct sets, notice that in the process this link lists eventually staring from single. They will eventually form a single list containing all the vertices, because we are going to form a spanning tree. A spanning tree has got all the vertices in 1 single component.

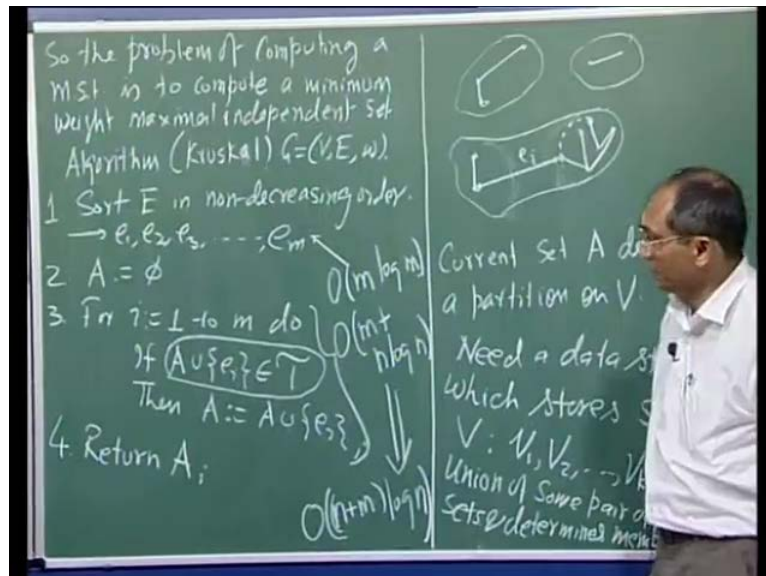
So, these link lists will eventually go into the single list. So, cumulative cost of the union we will identify separately and as far as the cost of find is concerned, it is a unit cost for finding out the set to which a belongs. The unit cost of finding out their d goal, so the total cost the cost of find is order m. This m is the number of edges, because for each edge we have to do constant amount of work. The cost of union, we will find out cumulatively. The way we go about it is as follows. Notice that the total cost of union is the total number of re labeling we perform.

The reason is, when we perform union, we walk along the list and we go along by changing the label as well, and then finally, we perform 1 unit cost of forming this additional link. The cost, therefore, associated with this operation is the size of the smaller set plus a unit cost, which we will count for later on. Now, what we will therefore, do is the cost of the union is that number of re labels performed on all the nodes. Now, notice that a certain node goes through several re labeling as it gets unioned over a period of time.

But, every time when you re label it becomes a member of a larger set, not only a larger set, but that new sets will be at least twice as large as this set. Because, in this case when we are unioning this with this now, this node is a member of a set of 7 vertices. So, each time the cardinality of its set at least doubles and hence the re-labeling cannot happen more than log n times, because you cannot double it more than log n time. Now, each node is re-labeled log n times.

So, the cost of union will be order log n for each node and there are n nodes. So, it is n log n. This is the cost of re-labeling. In addition there is a constant amount of work we do for every union, but every time you do a union you reduce 1 set from the partition. Initially we had n members in the partition. Eventually, it turns out to be 1. So, we do an additional order n work. Hence, the total cost is order n log n for the union operation.

(Refer Slide Time: 50:07)



Now, let us recall that in this algorithm this task crosses order m log m. This we have figured out. This thing costs us order m plus n log n, that is the union and find cost. So, the total cost of this, notice that log m is same as log n. The reason is m cannot be more than n square. So, log m is 2 log n, which is order log n. Hence, the total cost of the algorithm is order n plus m log n for the entire algorithm.