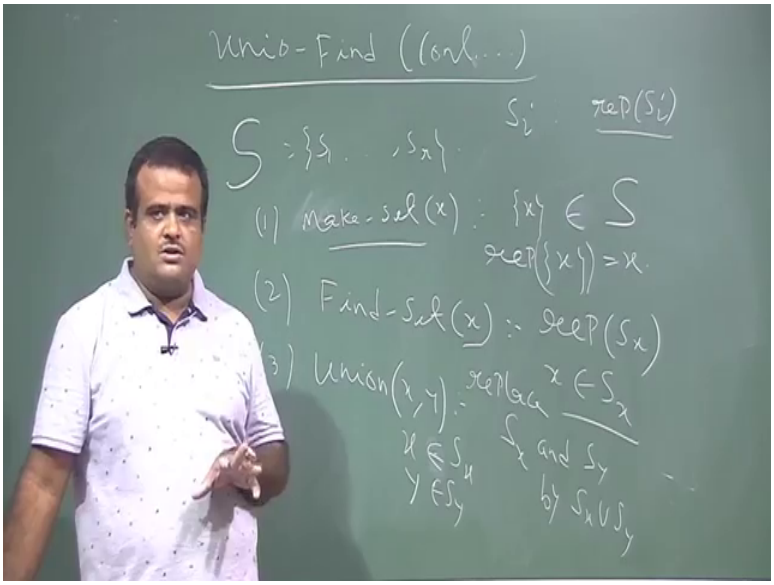
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 52
Union – Find

So we have we have seen the. So, we are talking about data structure for a dynamic set of sets I mean multiple sets. So, in the last class we have seen one solution. So, basically we have given. So, what is the problem we have given some sets of sets.

(Refer Slide Time: 00:34)



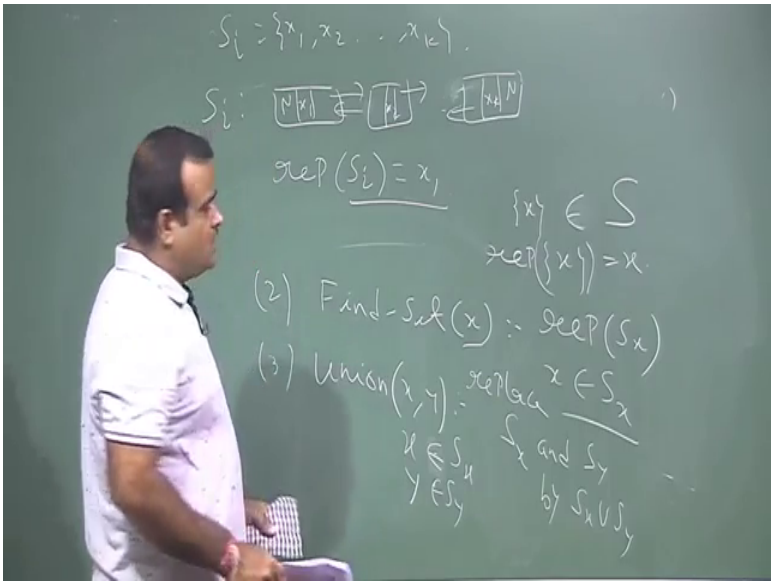
S k there are say r set and. So, this is dynamic set; that means, we can we can insert any set. So, we are looking for a data structure for this dynamic collection of sets as that we should able to perform the operations like make set. So, we have given element x we should able to. So, so from this S i there is a element which is called rep of S i representative element, which is representing the whole set S i.

Now, make set is basically we are x is not belongs to any of this given any of the set. So, this we are going to add this, this we are going to add this collection and the rep of this is basically because there is only one element. So, this is one operation another operation is find set. So, you have given a element. So, you want to written the set where this element is belonging. So, in order to determine the whole set we can we just written the

representative element. So, you written the rep of S x where x belongs to that is that set. So, this is the operation and then the union.

So, we want to do the union of x y. So, you have given 2 element say x belongs to S x and y belongs to S y. Now we want to replace S x and S y by their union. We replace S x and S y by S x union S y. And we want to have a representative element of this this new S x union S y set. So, this is the 3 operations we need to perform. So, we have seen one data structure that is the doubly connected linked list.

(Refer Slide Time: 02:57)



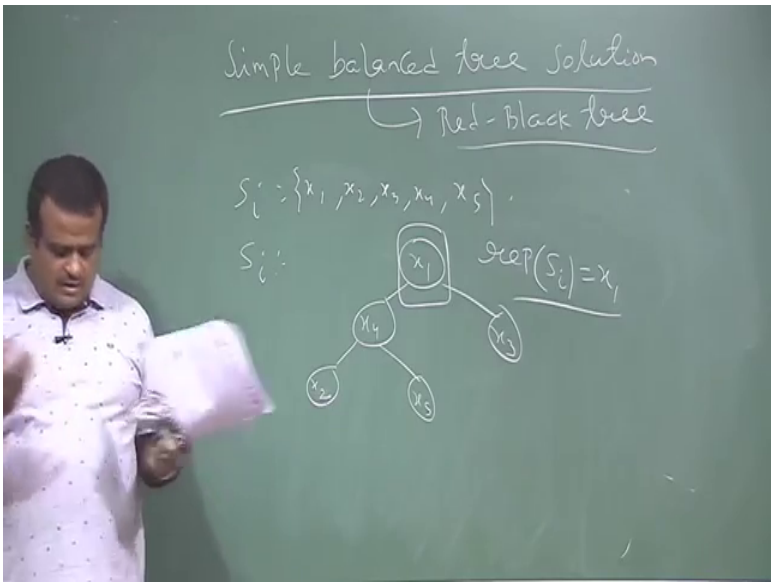
So, we represent each S i So, say S i is x 1 x 2 x k. So, this we have seen we use a x 1 sorry. So, x 2 x k. So, dot, dot, dot. So, this is basically null this is null.

So, this is the way we represent a set. So, using a doubly connected linked list. So, we have seen in the last class this data structure. And representative element is the first element of the first node the first node. So, this is a unordered list because the sets is itself is unordered. So, this is a unordered list. So, this representative element is the first element of that first node the value of the first node. So, that is x 1. So, this is the way and we have seen how to make a set making a set means we just we just create a node. And this is a basically collection of doubly collection linked list. So, this is a data structure we are using. And then we have say in the find set find set means we know the position of that x. So, we go there and we travels the list and we back to the first element.

And we written that and union union means we have added the another doubly connected linked list.

So, for that also we need to travel. So, this find set and union took us order of n times if the size is n. So now, we want to think how we can do it in better way? Like at least login time. So, for that we know any data structure we which can give is log in performance. So, what about balance tree? So, we can think for a simple balance tree solution.

(Refer Slide Time: 04:57).



So, let us this is the second proposal simple balanced tree balanced tree. So, this is the second proposal. So, one balanced tree we have seen in our class that is red black tree, this is a balanced tree. So, so what about where to what. So, the idea is whether we can use the balanced tree for our set.

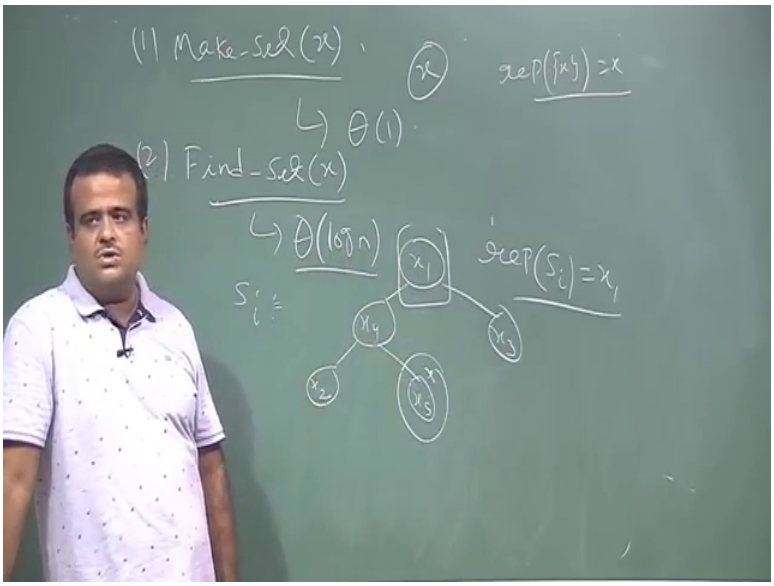
So, suppose so, how we can do that? Suppose we have a set S, S i which is basically say x 1 x 2 say x k. So, we want to have a tree for this and which is a balanced tree. So, we do not care about the key value because this set is unordered. So, there is no we ignore the. So, the this is the unordered set. So, there is no ordering of this element. So, there is no question of say binary sets tree property. Because we there is no ordering of the element. So, this key we are ignoring the key value ignoring the key ignoring the key this key.

So, once we form a this red black tree we have a key value and based of that key value we form the binary statistic property. I mean any node in the lefts of tree is less than the key value of the that route any node in the writes up trees key the greater than the key value. So, that we are ignoring the key value because the sets is the stay on or collection of disjoint element yeah, distinct element is said. So now so, we have a tree, we have a tree. So, we just make a balanced tree. So, we have a balanced tree while we are in the do not bothering about the value of this nodes because this is say this is just un orders collection.

So, but the structure should be balanced. And then what is the representative element? So, representative element can be the route of the tree. So, that is the way. So, for example, suppose we have given this said say 5 element say x 1 x 2 x 3 x 4 x 5. So now, how to form a tree? So, suppose this is x 1 and we do not care about the value because this is un ordered list. So, x 4 can be come here x 3 we just do care about the structure this tree should be balanced the height is should belonging x 2 x 5.

So, suppose this is our S i. Now what is the rep of S i rep S i is basically the route, rep of S i is basically x 1 route of the element. So, this is our data structure. So, we just make a balanced tree with this un ordered list and un ordered with this elements and we just ignore the key so, but the structure should be balanced height should belonging. So, this is our proper data structure, now we want to see how we how first we can perform those 3 operations like make set. So, how to make set? So, how we can? So, first operation is make set.

(Refer Slide Time: 08:54).



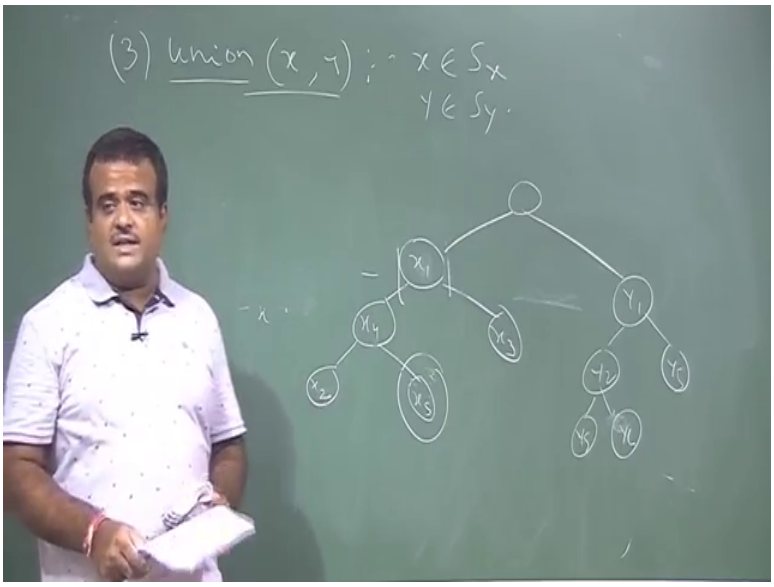
So, we have given a element x which is not there in the collections. So, we want to create a set we want to insert a any insert a set in this is the insertion operation in that collections keep test. So, this is basically we just create a note that is it we just create node x. This is a single node tree just crate a node and the and this is the rep of this x is basically the first element this is route is x itself. So, this is the way. So, what is the time complexity of this constant? So now, we want to see how we can do the find of find set find of x.

So, find of x take how long time. So, find of x mean suppose we want to find the. So, you suppose this is our x x 5. So, we know the position of this. So now, we have to written the representative element of this. So, how to written the So, you have to get the you have to reach to the we have to go to the parent of this then parent of this. So, we written the x 1. So, that is the that is the operation for find set. So, we want to written this S i, because we know S is belongings to S i x is say x 5 now we have to written these x 1. X 1 is the route of that. So, in order to reach to the route, what is the time? Time is log n because this is the balanced tree.

So, this will take log n time. So, find operation we will take log n time. Because we know the x say we want to fa find the set where x 5 is belonging. So, that is basically S i S that is basically this tree now you want to we have to written the reprehensive element of this that is basically the route. So, to reach to the route we have to go to the height of

this tree. Because this is the balanced tree. So, this is log n now how to perform this union operation? So, that is the that also you want to see whether you can do it in log n time or not.

(Refer Slide Time: 11:34).



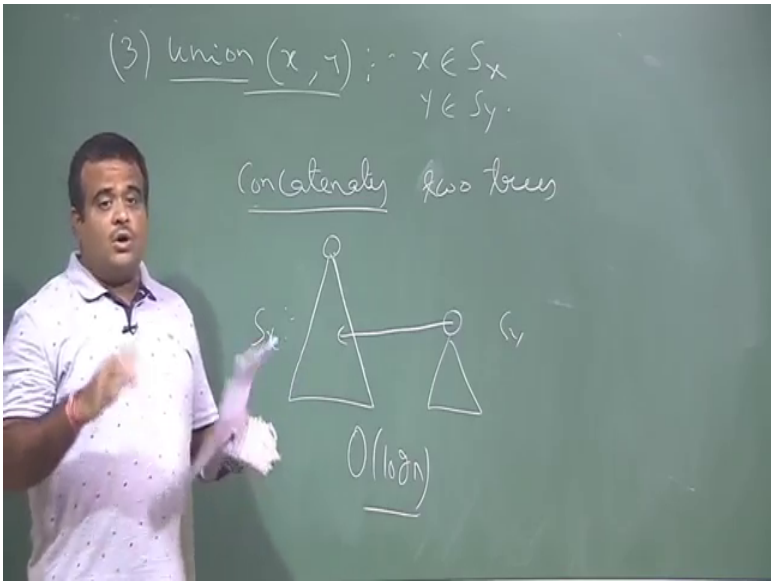
So, union So, suppose we have 2 element x and y suppose say x belonging to the set and y belonging to the another set and we want to replace this 2 set S x and S i by their union and we have to.

So, how to do that? So, this is suppose this is our S x this is our S x S is somewhere here and we have a another set say S y. So, if this is say S I. So, this is say y 1 say y 2 y 4 y 5 y 6 something like that. So, we have 2 sets S x and S y, now the question is how we can merge this 2 set? How we can merge this to tree, and in order to get a balanced tree again. So, we have to get the ultimately this union should be a set and union is a set and that is also be a tree. So, that has to be balanced. So, what is the proposal?

So, we can think of having say. So, how we can merge this 2 set. So, we can have a new route like we can have this S element over here then we can do like this kind of think or else we can just. So, this is already a balanced tree. So, we can just keep on insert this nodes. So, we have y 1 y 2 like this. So, we can insert this node into this tree. So, that again we will take in log n time because if there are n elements in the set y order of n element then this itself will take l log n time, but we want to do it in log n time. So, that

is the idea of concatenation. So, if we have a 2 red black tree. So, that is the idea of concatenation like.

(Refer Slide Time: 13:58).

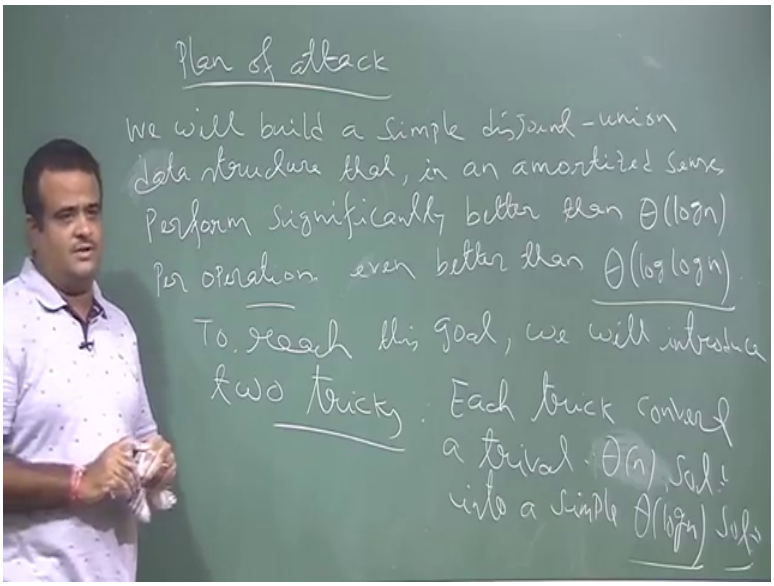


So, this we will do the concatenation of 2 trees concatenates this 2 red black tree.

So, like we have this tree we have a bigger tree. Let say this is say S x we do not know S a S y may be bigger. So, for exam this is another tree S y. Now we can add this tree over here by concatenate operation we have seen this operation in a red black tree. So, this is basically these we will take and this is basic we are just concatenate this tree over here. And this will take log n time the height of this tree. So, basically we take this element then we search the position of this element. So, this way we will do. So, this we will take log n time. So, this is the concatenates operation. So, this is log n time, but So, this is the logarithm time you are doing. So now, we want to do something better on this like, we want we want to build a we want to do log log n time, how we can achieve a log log n time?

So, this is this is the this is what we are going to do now is called data structure augmentation. So, we want to argument our data structure So that we can do the operation in faster way. In a amortized sense. So, we want to do lots of amortized analysis. So, there we will we will see how we can get log log n log log log n.

(Refer Slide Time: 15:58).



So, this is the next things we will discuss this is the let us give a outline of this what we are going to do that is called plan of attack. So, this is sort of outline what we are going to do I mean how we can. So, this is basically bunch of augmentation you will do in our data structure. So, basically you have to basic data structure one is w connected linked list another idea is the tree.

Here it is a red ba balanced tree. So, we will see a general tree. And then we will see how we can argument this 2 data structure and we will see we will do some amortized analysis in order to get the performance is log n log log n like this. So, the idea is to we will built we will built a simple disjoint union data structure data structure that in amortized sense we will do the amortized analysis amortized sense perform significantly better than log n, perform significantly better than log n per operation. So, we have basically 3 operations ok.

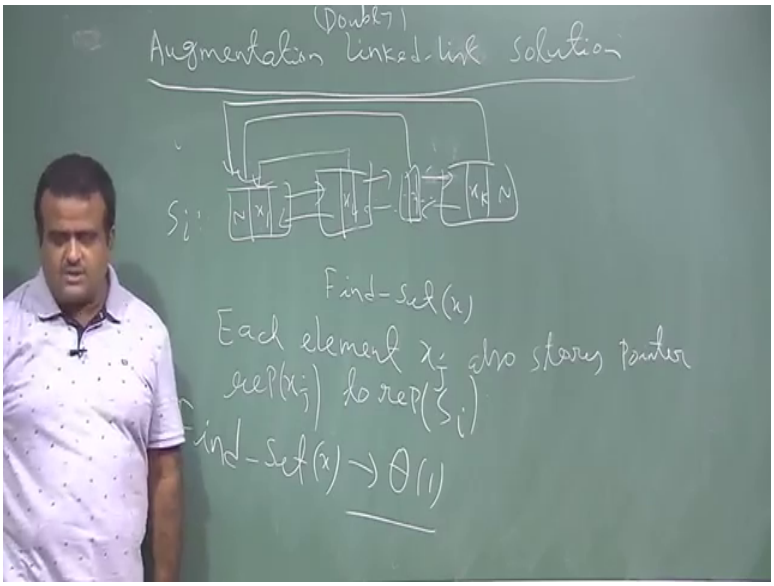
So, you want to make it faster on this 3 operation like make anywhere makes make set is always constant. So, that we are not bothering, but other 2 operation like find set for the w connected linked list find set we will take order of n time. So now, the question is how we can do some sort of augmentation or how we can do some. So, after augmentation how we can do the analysis amortized analysis to get it log n not only log n log n operations even better than even better than log n. So, log n better than log n is means log log n and so on, but not constant because log log log n. So, it is not constant

basically, but it tending to. So, this log this we have seen in one data structure what is that the band (Refer Time: 19:22) was data structure ok.

So, they are we have seen the fixed in our successive problem. So, they are we have seen log u, but they are we have used 5 tricks to achieve that, but here we are going to use the 2 tricks only 2 achieve this. So, to reach this goal. So, this is our goal. So, to reach this goal we will use 2 tricks. This goal we will introduce we will introduce we will introduce 2 tricks. Each tricks convert this operation form theta n to log n. Each tricks convert a trivial solution a trivial theta n solution to solution into a simple log n solution. So, that is the going simples log n solution each of this string. So, theta n solution means we have seen a doubly connected linked list. So, how we can, but this will be in amortized sense this analysis is amortized analysis. So, we will talk about that.

So, first trick is on this doubly connected linked list. So, we will do some augmentation on this doubly connected linked list and then we apply this trick. And the second trick is on the tree. So, we will apply the trick on the tree data structure. So, let us talk about first trick I mean, let us go back to the data structure which is we use for on the w connected list. So, this is the we will do some augmentation.

(Refer Slide Time: 21:56).



So, augmentation on the on the linked list. So, we have a linked list solution. This is basically doubly linked list. So, doubly linked list linked list we will do some

augmentation in the doubly linked list data structure augmentation to have the better performance on this operations like a ok.

So, just to recap. So, we have a said S i. So, it should be this is x 1 x 2 x k. Now what is our data structure or data structure? Is just simply a w connected linked list. So, x 1 So, this is our S i. So, x 2 dot, dot, dot x k. So, this is basically this is null this is null. So, this is the data structure we have use this is the doubly linked list we have used for our this problem. Now we have seen for this data structure suppose we have a node S of what is the find operation find set of x suppose x is here. So, we know the position of x.

So now to reach to the representative element what we did we have to come back to the beginning. So, that was taking log n time. So now, how we can augment this in order to achieve this in a constant time. So, how to augment this data structure So there we can achieve this in a constant time. So, any idea? How we can make this operation find set faster? I mean theta one time in set of theta of n times. So, you have to do some augmentation in this a list. So, what we do? So, what if we store So each element in each element other than this we store the representative element; that means, each element we have a list to the each element we have a list to the beginning. So, we have a we store the pointer of the first element.

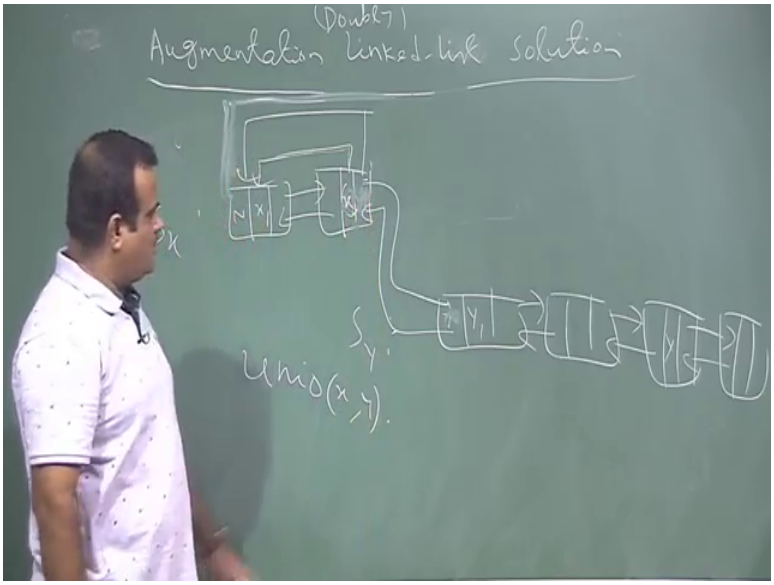
So, this is the augmentation we are going to do for our doubly connected linked list. So, we store the we store the each element each element x j each element x j, also store pointers pointer and that pointer is called rep of x j pointer to Rep of S i.

So, that is basically the first element. So, you have storing the pointer of this S j. So, each of this will point to the first element. So, that then the find set is easy. So, if we have a link from this to this. So, suppose this is x. So now, to get the representative we just take this pointer and we take this x 1. So, that will return. So, that will find set we will take constant time, find set we will take constant time after doing this augmentation. So, this is not a amortized analysis this is the exact analysis. So, this will take constant time. Now we want to see how we can manage the union.

We have already seen make say make said that is constant you are not bothering about that. So, we will see how we can manage the union operation. So, that is the that is our next discussion how we can achieve the union operation, how we can perform the union

operation of this 2 set. So, what type of augmentation we will do because for union if you have another set y So, if you have another set y.

(Refer Slide Time: 26:49).



So, say y 1. So, this is say we have say 2 element in this set. So, this is only 2 element say x and we have a y which is say 4 element.

So, this is also a doubly connected linked list. So, this is S x. So, this is say x and this is somewhere y is there. So, this is S y this is y 1 this is the representative element of S y. Now we want to do the union, union of x comma y. So, for union not we are doing? We are just linking this to this and this to this. This is the union operation we are doing. So, for that what we have to do? We have to go back from here to here so, but we have the link for this. So, so that that way we can, but we want to do further augmentation. So, that how to now if you just joint this like this, then again we have to for all this element we have to go to the beginning.

So, this we will take again linear time if the size of this is linear. So, anyway you will continue this in the next class.

Thank you.