Data Structures and Algorithms

Dr. Naveen Garg
Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Lecture – 33

In the last class we looked at the union find data structure. That was a data structure to maintain a collection of disjoint sets, under the operations of union. That was the operations we were doing in the sets, that was the operation which were modifying the sets or modifying the collection and the find operation was just to identify, given an element, which set it belongs to. We used this, we needed this data structure to implement the Kruskal's algorithm. Kruskal's algorithm was the first algorithm we looked at for computing a minimum spanning tree in a graph. It was an example of the greedy algorithm.

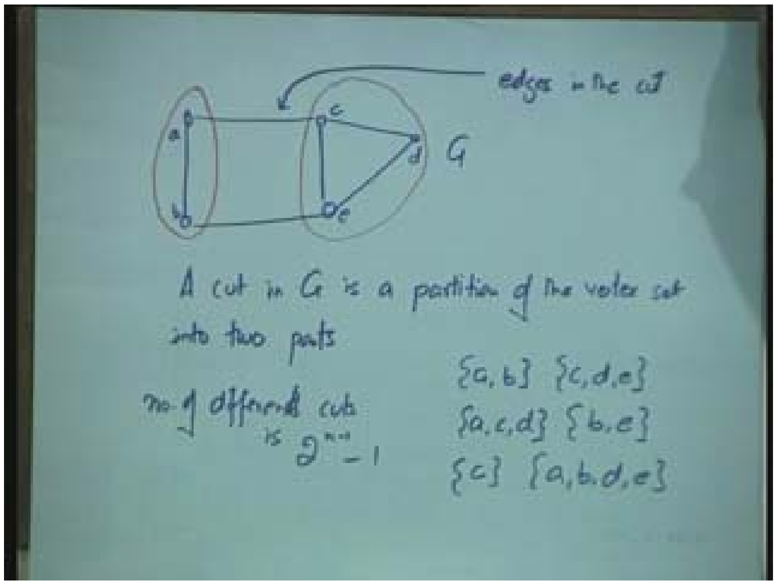
Today, we are going to look at another algorithm for computing the minimum spanning tree in a graph. This one is due to Prim and that is what we are going to discuss today. So, let me define the notion of a cut in a graph first, for you. So, recall that we are talking of undirected graph. A spanning tree, the notion of a spanning tree is defined only for an undirected graph. For a directed graph, there are different notions. We do not say spanning tree in a directed graph. It is only an undirected graph we are talking about here.

So, give you an undirected graph, a cut: let say, this is graph G and a cut in graph G is a partition of the vertex set into 2 parts. What does it mean? Let me partition, break vertex into 2 pieces. So let say, 1 piece is this and the other is remaining because, it is the partition. This way of splitting would define a cut.

So, if for an instance, I had vertices, a, b, c, d, e. So, then the first cut I am considering is a, b one side and c, d, e on the other side. There could be many other cuts possible, I could have a, c and d on one side and b and e on the other side or I could have only 1 vertex let say, c an one side and the other 4 on the, the remaining 4 on the other side. These are all examples of cuts in the graph. How many cuts can there be in the graph? 2 to the power n? 2 to the n or something? 2 to the power n minus 1. because, this partition is the same as saying, c d e and a b.

How did you come out with the number 2 to the n? Every element, for every element, there are 2 choices, either left or right. So, there are 2 to the n possibilities but then, but then, we are repeating. Each possibility is repeated twice. Once we will say, a b go on the left side, c d e go on the right side and the other time we will say, c d e go on the left side, a b goes the right side. But the partition is the same. So, that is why number of different cuts, 2 to the n minus 1, minus 1 also. Okay, then null? You are talking of the null (hindi) Sometimes, what we do is when we say a cut, so cut is really a partition of vertex a. But I sometimes would also say, these edges which are going from 1 side to the other side are called the edges in the cut. So, these edges could be called the edges in the cut, edges in the cut or edges of the cut, whatever. You understand which edges I am talking about? Edges which have 1 end point in 1 side of the partition and the other end point in the other side of the partition. So, you understand what a cut is. You need this notion and how to we use this notion?

(Refer Slide Time: 5:53)



A cut in G is a partition of the vertex set into two parts

no. of different cuts is $2^{n-1} - 1$

$\{a, b\}$ $\{c, d, e\}$

$\{a, c, d\}$ $\{b, e\}$

$\{c\}$ $\{a, b, d, e\}$

Suppose, I take a cut of my graph, some cut in the graph, S and I will denote other point, other side by S complement or V minus S where V is the vertex set. So, I have a graph G S V, the vertex set and e is the set of edges. So, S is the set on one side and the remaining is the set on other side. Now, let me look at all the edges which are in the cut.

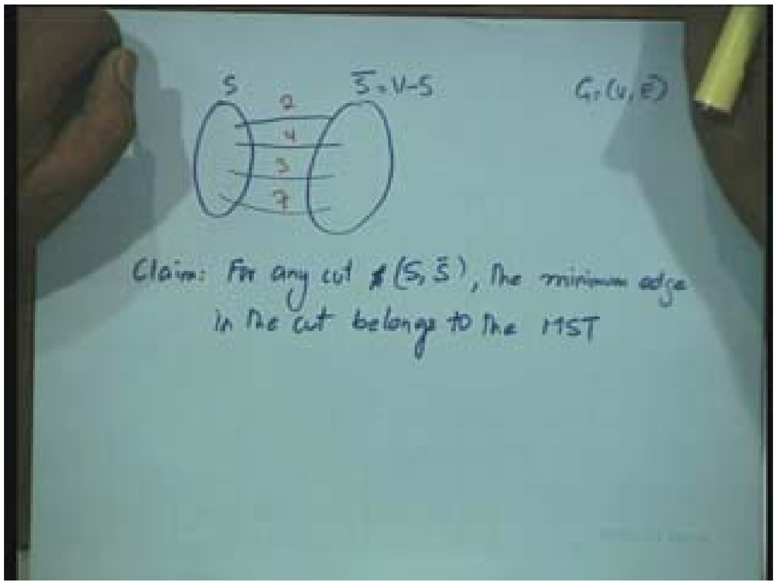
Recall, we are trying to compute the minimum spanning tree. So, I am now going to be taking of a particular property of the minimum spanning tree. So let me take this cut and let me look at the edges in the cut and let us look at their lengths of these edges. So, may be, this edge is length 2, this edge is 4, this is length 3, this is length 7.

I am going to be assuming that all the edge lengths are distinct. This is just to ease all my arguments, to simplify my arguments. All the algorithms, everything works even when edge lengths are not necessarily distinct. So, just to simplify my presentation here that I am going to assume edge lengths are distinct.

Now, the claim is that, so recall, when the edge lengths are distinct, there is a unique minimum spanning tree. We have discussed this before: there is 1 and only 1 minimum spanning tree. Now, the claim is that this edge which has length 2 will be a part of that minimum spanning tree.

Let me write down the claim more formally, for any cut, any cut S, S complement; the minimum edge in the cut belongs to the MST (hindi) 2 to the n minus 1 different cuts.

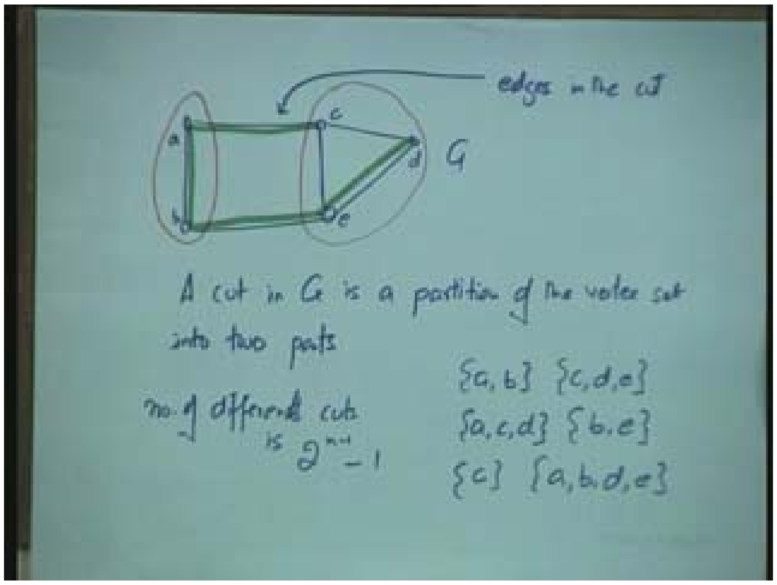
(Refer Slide Time: 8:20)


S
$\bar{S} = V - S$
$G = (V, E)$
2
4
3
7

Claim: For any cut ~~$\neq$~~ $(S, \bar{S})$, the minimum edge in the cut belongs to the MST

(hindi) 1? for instance (hindi) let us make a tree such that there will be both of these edges will be part of the minimum spanning tree or part of the minimum spanning tree. (hindi) both the edges are part of the minimum spanning tree. I could have my edge length so that this was my minimum spanning tree. I could choose my edge length so that this is the minimum spanning tree. So, it is not necessary that only, for any cut, there is only 1 edge which is the part of minimum spanning tree, there could be more than 1 edge which could be a part of minimum spanning tree.

(Refer Slide Time: 9:26)


edges in the cut
a
b
c
d
e
G

A cut in G is a partition of the vertex set into two parts

no. of different cuts is $2^{n-1} - 1$

$\{a, b\}$ $\{c, d, e\}$
$\{a, c, d\}$ $\{b, e\}$
$\{c\}$ $\{a, b, d, e\}$

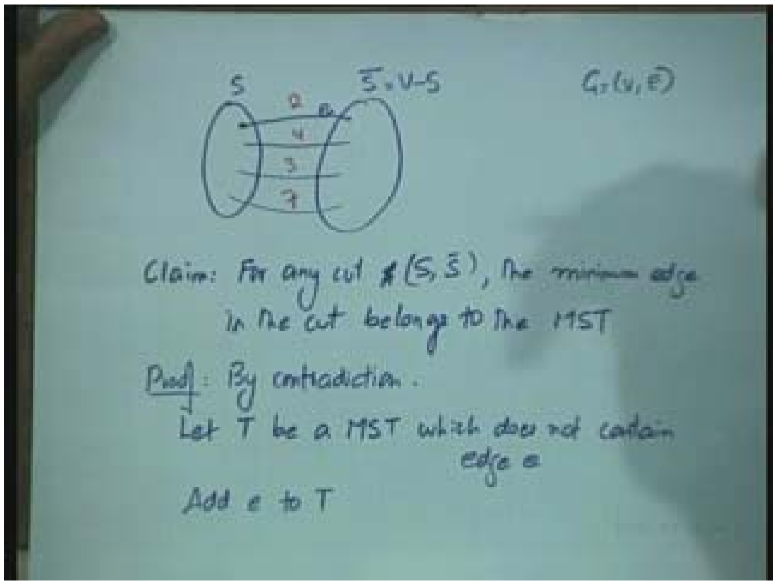
But, all we are claiming is that the minimum edge in the cut will belong to the minimum spanning tree (hindi) why? What is the proof? So, proof is by contradiction (hindi) you are saying that this is the minimum spanning tree but, that tree does not contain this edge. So, let T be a MST. MST is short for minimum spanning tree. T is an MST which does not contain edge (hindi) does not contain edge (hindi) which does not contain edge e.

So, what now? What will I do? I will add e to the tree, add e to T. What will happen? Cycle will be formed. Why will a cycle be formed? Because, these 2 vertices are already connected, these 2 vertices are already connected in the tree. So, there is some path going from this vertex to this vertex. Any path that goes from this vertex to this vertex, from u to v which connects u and v has to use 1 of the edges of the cut.

Think of this as, this is the river in between, this is 1 end, this is the other end. If you have to go from this end to other end, you have to use 1 of these bridges, no other alternative. So, it has to use 1 of these edges which means that the cycle that gets formed has to use 1 of these edges at least.

But, all these edges has length more than 2. So, what will I do? Same thing, I will remove this edge from the cycle and at that, this will reduce the length of the tree. So, let me write that down. So, we add e to T, let me continue.

(Refer Slide Time: 11:59)


S
$\bar{S} = V - S$
$G = (V, E)$
2
e
4
3
7

Claim: For any cut $(S, \bar{S})$, the minimum edge in the cut belongs to the MST

Proof: By contradiction.

Let T be a MST which does not contain edge e

Add e to T

So, after you add e to T, so let me addition of e to T forms a cycle. Say, capital C. C contains at least 1 edge of the cut. Yes or no? At least, 1 edge of the cut, this implies C contains at least 2? It will contain an odd number of edges. Added C, at least 1 edge, let me write down other than e, good. (hindi) e will have to be part of the cycle. It is because of the addition of e that cycle got formed. There was no cycle earlier, e has to be part of the cycle.

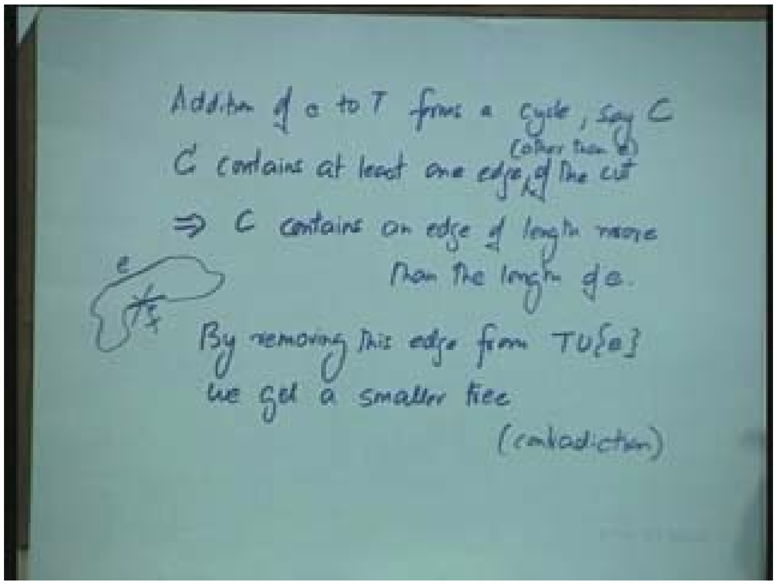
So, C contains at least 1 edge of the cut other than edge e. So, this implies C contains on edge of length more than the length of e. C contains an edge of length more than the length of e. They can be greater than … I just have to show that there is some edge in the cycle which has length more than e. No, we created the cycle, we brought in e, now how do we want to reduce the cost of the tree? By removing some other edge whose length is larger than e. By removing this edge from T union e, we get a smaller tree.

That is a contradiction, because we assumed that T was a minimum spanning tree. (hindi) This we have discussed before that, from a cycle I can remove the any edge and it will remain a spanning tree.

We can also remove that, I just have to show you that there is some edge in the cycle which can be removed. To just, I have to just eye with a contradiction. I am just arriving at a contradiction here, this is a mind game. We are not actually removing any thing, I am just proving a structural property. I am saying, in any cut the minimum edge has to be a part of minimum spanning tree.

I do not ever sit down and remove any thing. There is no algorithm does that. This is just a proof, proving this statement. (hindi) How do know that 1 of the edges of the cycle has length more than e? Because, we can only argue for the edges of the cut, for then we know for sure that they are all more than e. That is all, nothing more. Now, how are we going to use this claim?

(Refer Slide Time: 16:57)



Addition of e to T forms a cycle, say C

C contains at least one edge (other than e) of the cut

$\Rightarrow$ C contains an edge of length more than the length of e.

By removing this edge from $T \cup \{e\}$ we get a smaller tree

(contradiction)

So, Prim's algorithm exploits this simple fact: that, if you take any cut, the minimum edge in the cut will be part of the minimum spanning tree, always. Prim's algorithm is an algorithm which essentially is built around this simple fact. (hindi) So, let us understand what Prim's algorithm is.

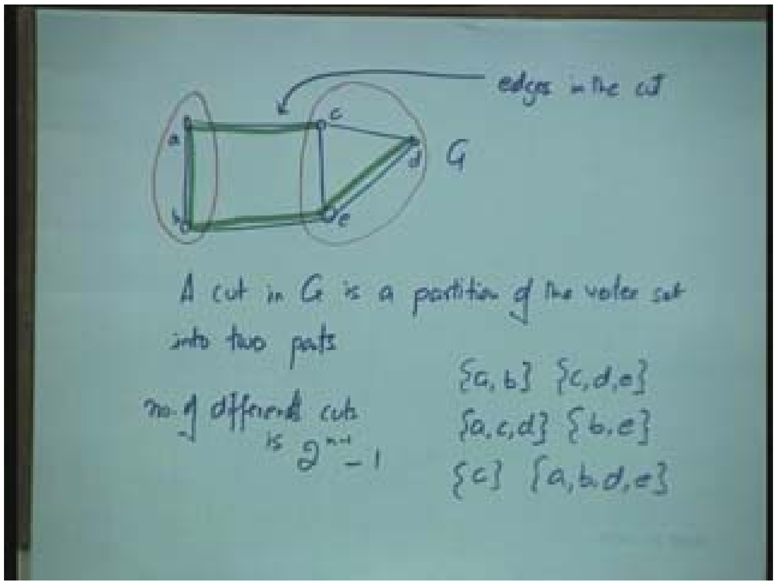
Let say this were my, this were my, what? Graph and let us give every edge length. So, I will just rapidly put down edge lengths, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, say, I missed out one, 13. Now, how does Prim's algorithm work? First, we start from some vertex and we call this the root vertex. Now, the first partition that I am considering, the first cut I am considering is the root verses everyone else. Root of 1 side and everyone else on the other side.

So, which edge has to be part of the cut? The edge of length 1, so, I will include this into my cut, into my which edge has to be part of the minimum spanning tree, edge of length of length1 and I will include it into my 3. Now, what is the cut I am going to consider? This and this, so I am going to, this is going to 1 side of the cut, this is going to be one side of the cut and the other side of the cut is going to be all the other vertices.

So, which are the edges which are part of the cut? 9, 8, 2, 3, 11, so which is the smallest? 2, so we know for a fact that this has to be there in the minimum spanning tree, this included (hindi) What is the cut I am going to consider? These 3 vertices on 1 side and all the other on other side, so side let me extend it like that: can everyone see this? This is 1 side of the cut and all the other vertices are on the other side of the cut.

So now, which are the edges in the cut? 9, 8, 7, 5, 13, 11: 3 is not in the cut, because both end points of 3 are on the same side. So, the smallest of these is 5. So, this gets included. Now, I am going to, you can now understand how I am going to extend it. This becomes my, 1 side of the cut and the remaining becomes the other side of the cut, 4 is in edge, 3 is some kind of a gone, so 9, 8, 6, 4 - no, not 10 – 13 and 11 (hindi) 4 (hindi)

(Refer Slide Time: 21:20)



A cut in G is a partition of the vertex set into two parts

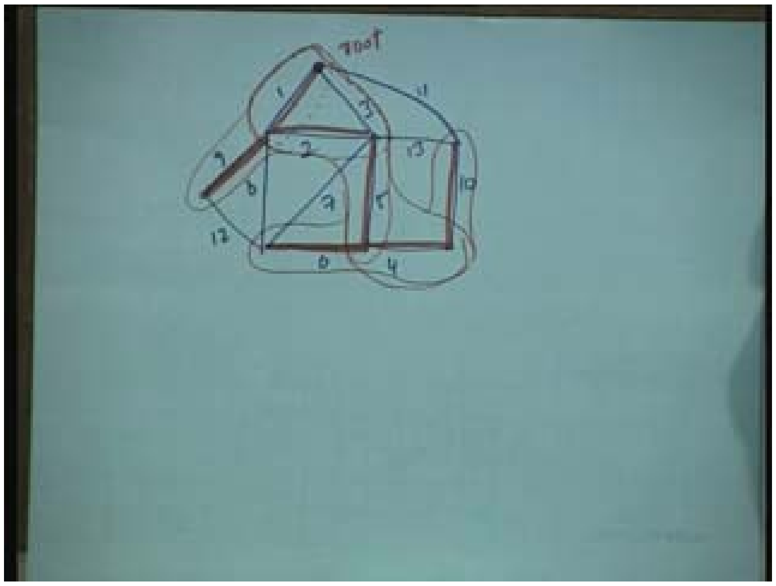
no. of different cuts is $2^{n-1}-1$

{a, b} {c, d, e}
{a, c, d} {b, e}
{c} {a, b, d, e}

Which are the edges in the cut? 11, 13, 10, 6, 7, 8, 9; so, six is the smallest, 8 gets included and now, my set becomes this. It is time, we finish, because it is getting very messy. So, which are the edges in the cut now? Is 8 in the cut? No, it looks like it is in the cut, but it is not because, both its end points are in the same side. So, it is 9, 10, 11, 13 also 12, of course. So, 9, 10, 11, 12, 13.

So, 9; 12 is not in the cut any more. So, it is only 10, 11, and 13 are the options, 10 is the smallest and we that we are done. Because, all the vertices are now included. Everyone understands the procedure, it is very simple. We have done the proof effectively. Not effectively, we have done the proof. Because, what did we say? We proved this claim that the minimum edge across the cut has to be part of the minimum spanning tree and that is the edge we are picking at every point. So, this is the minimum spanning tree.

(Refer Slide Time: 22:33)

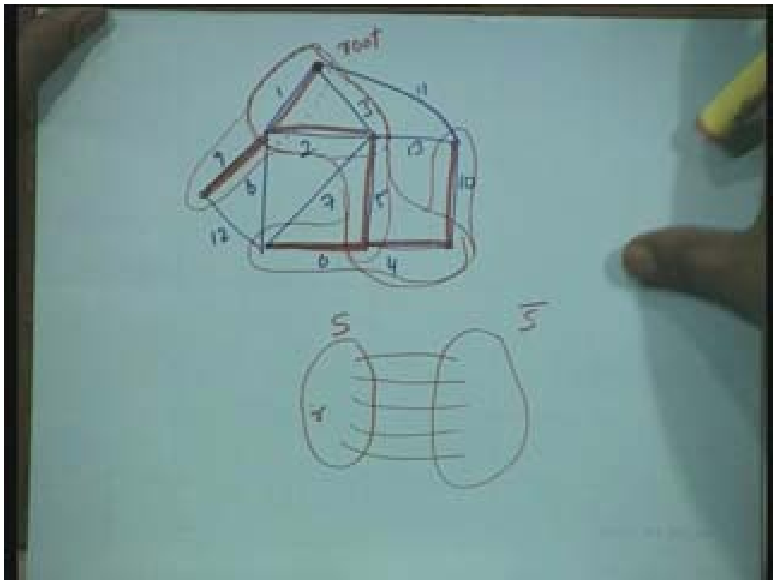


You will get the same tree, since this is unique, I assumed edge lengths are distinct, you will get the same tree when if you were to run the Kruskal’s algorithm. Everyone follows this? So, I am not going to write down the pseudo code for this, but you understand how the algorithm works. Very simple, so the key idea is that we have, so let us try and see how you would implement this algorithm?

So, what is it that you have to maintain? Of course, you have a data structure for the graph. Now, what it, what is the operation you have to do at each step? Add a vertex to a set? So at some, at any point you have the following; this is 1 side of the cut, your root is here. Let me call this set as S. So, the set is, this is going to be the set on 1 side of the cut, as in the vertices we have already reached, so to say, from the root.

So, the root is always the part of the set S and the remaining vertices, S complement of V minus S. I have to maintain this collection of vertices which are on 1 side. This can be done very easily by keeping 1 bit with every vertex. If the bit is 1 then that means, let say it is on the S side, if it is 0, it is on the S complement side. That is very simple. Now, what is it that I have to do at each step? I have to find out, I have to look at all of these edges and to find the minimum. How and … I am going to do this? (hindi) you look at all of this vertices, you look at their adjacent edges, for each edge you see whether the other end point is in S or not. If it is not in S then you look at its length and you look at all these edges and find the minimum. So, how much does the time take? How much time does it take? Order, you are going to each vertex, looking at all its adjacent edges; looking at all its adjacent edges, going to each vertex in S, looking at all its adjacent edges. What is the maximum time it could take? Order m at each step and you have n such steps. So, order m in time (hindi) I am not even writing it down because it is a very ….

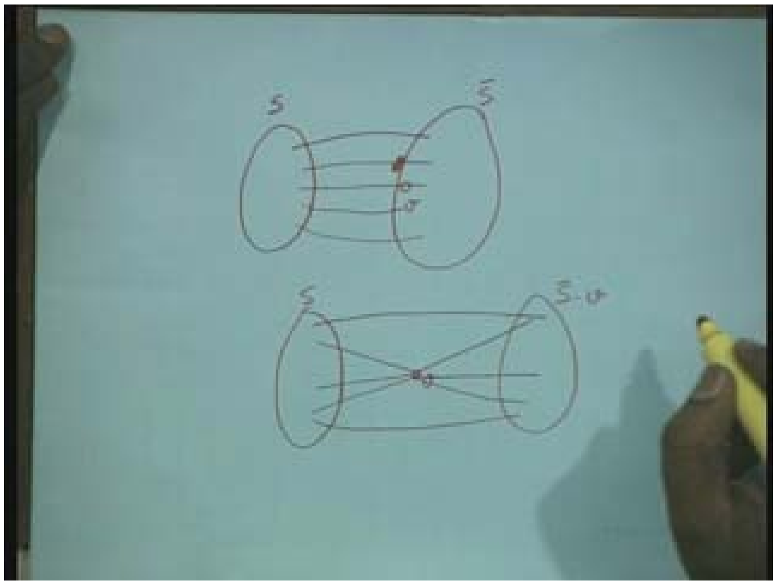
(Refer Slide Time: 24:56)



So let us do, look at something clever. What could be a clever way? Maintain a minimum? Let see, this is some idea: so, you are saying I have this set s, I have s complement and there is this vertex V, let us call it V.

I have a single this vertex out because this is the vertex which is now going to go from right to left. (hndi) You have figure out minimum of all of these edges, you know the minimum of these edges in the cut. Now, what is going to happen? V, let me draw it this way now.

(Refer Slide Time: 27:53)



Some edges are going to go from v to S, some edges go from V to S complement, there are of course, some edges which go from S to S complement. Now, right now, I know the minimum of these edges because V is the same. This picture is the same as knowing the minimum of the edges. When I move V from here to here, I want to find out the minimum of these edges. What will I do? Compare the? So I just, suppose I have kept track of the minimum of these, noting else. Let say that the minimum was 1, the minimum has to be an edge incident to this one: that you understand? So, that is why we pick this one. This is 1, this is 2, this is 5, this is 6 and this is 7 and then these are 8, 9, 3 let say.

Now, the new minimum, earlier the minimum was 1 and now the new minimum is 2. If I just know the minimum, it is not going to useful. So, you want me to keep the track of the second minimum also? Which does not go to? But, what do I know about V as in, V something that I find out. So, we know the minimum that is coming to this vertex. So suppose, we keep the track of the minimum to each of the vertices in s complement. Insert V? So, what he is saying is, but how to, what data structure should I use to keep these edges?

A heap? Min heap? So, that could be 1 possibility. I have a heap, does everyone know what a heap is? So, heap is the data structure which will and which I can put some elements, each one of them has certain priority or certain key and it will give me, I can use an operation called delete min which will remove the minimum element, I can find out what the minimum element is in the heap in constant time, I can insert element into the heap, I can also remove element from the heap. I can do all of these operations and except for find min, all operation take in log n time. Find min takes constant time. Everyone remembers at least this much.

So suppose, I were to keep a heap here, a heap which contains these edges, 2, 1, 6, 5, 7. Then, using find min I can find out what is the minimum edge. That will, once I know the edge, I know the other end point of vertex, I know both the end points of the edge, I know which vertex I have to bring in. When I have to bring in this vertex, I look at all the edges incident at this vertex.

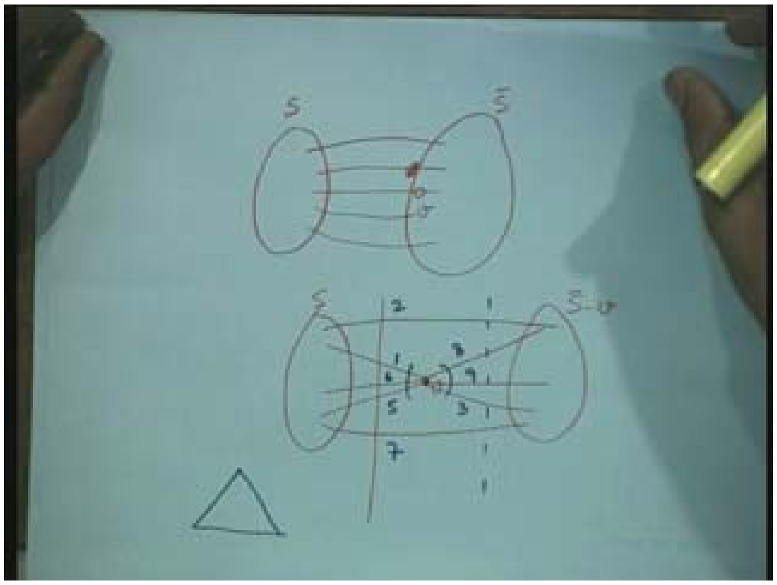
The edges which are going in S, I have to remove them from this heap and the edges which are going from this vertex to vertices not in S, I have to add them into the heap. But, let us keep it clean. Like, when we say that our heap is going to contain the edges of the cut, let it contain only the edges of the cut, because you are going to spend the same amount of time in any case. We do not search in a heap, please remember, heap is a very bad data structure for searching.

So, how do you find out where the edge is in the heap? Once again the same thing, when you put some information related to the edge into the heap, you do not just put it and forget it, you keep a track of where the information is, where the particular node is. We do not need to delete, but there is no harm in deleting also. No, you can also delete from a heap. Why cannot you delete from a heap?

So, there is some confusion here. Let me back up a little bit. We need to, I think I need to show this delete operation to you again. When we do or the other class we will do delete operation once again but take it from me, you can also delete from the heap in the same log n time. Of course, to delete you need to know where the element is, you are not searching for the element, you know where the element is and then you can delete from the heap in log n time.

So, that is what we are going to do. When this element comes in, I am going to insert 3, 9, and 8 and I am going to delete 1, 6 and 5 and I know exactly where 1, 6 and 5 are and this is how I will do the implementation. So at any point, what does the heap contain? Exactly the edges of the cut: exactly those edges and noting else. It is best to keep it clean so that you know what is happening. Now, how many operation are required on the heap? what kind how many?

(Refer Slide Time: 32:29)



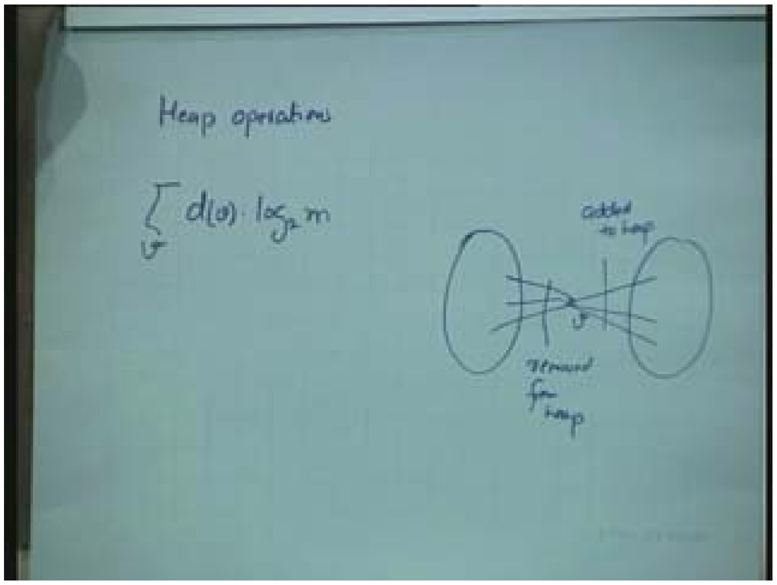
So, let us look at the heap operations. Let me draw the picture again for you. This is vertex V, it is coming in on this side. So, these edges have to be removed. These edges have to be added. Each remove and add takes log n time. How much time am I spending? Degree times log n for ever vertex that I process.

So, degree of V times log n and then this has to be summed over all the vertices, because I will processes each vertex exactly once. Log n, number of elements in the heap, which could be m. So, I should write log m. Number of elements in the heap which is the number of edges in the cut, could be as much as the number of edges. All that it will ever may be that but it could be as large as then in the worst case.

We are decreasing, but you know those are so small, that it will not going … even if you do a careful analysis, you will still get the some order. This is what the total running time will be. These are not the only operations; I also what have to do 1 find min. How many times do I have to do find min? Number of vertices time; n times. So, (hindi) n into order 1, constant time, that is a small order time in this one.

What else we have to do? What are the other operations you might have to do in the heap? Just insert and delete: and this operation, find mine. So, everyone understands what the procedure could be now? You start with your initial vertex, root and all the edges incident to the root will be your initial heap, will be the elements of initial heap. Then, you will do a find min that will tell you what vertex to include on that side. Then you are going to look at all the vertices adjacent to this vertex, all the edges that are adjacent to this vertex, beside, if you are to remove an edge or to add an edge. (hindi) Let say, I have an array S. S in an array, S of V equals 1 if V belongs to S, 0 otherwise.

(Refer Slide Time: 37:42)



What is this S? This S is the S side of the cut. Not visited, but reached; all those vertices that have been reached. So, this is 1 data structure we are going to have. Then, we are going to have a heap and let see what, so initially, I pick a vertex as the root and S of the that vertex root equals 1 and for all V, S of V equals 0. Initially, everything this is my initialization. Only the root is on, so, initially everything is 0 and S of r is set 1. This is my root vertex.

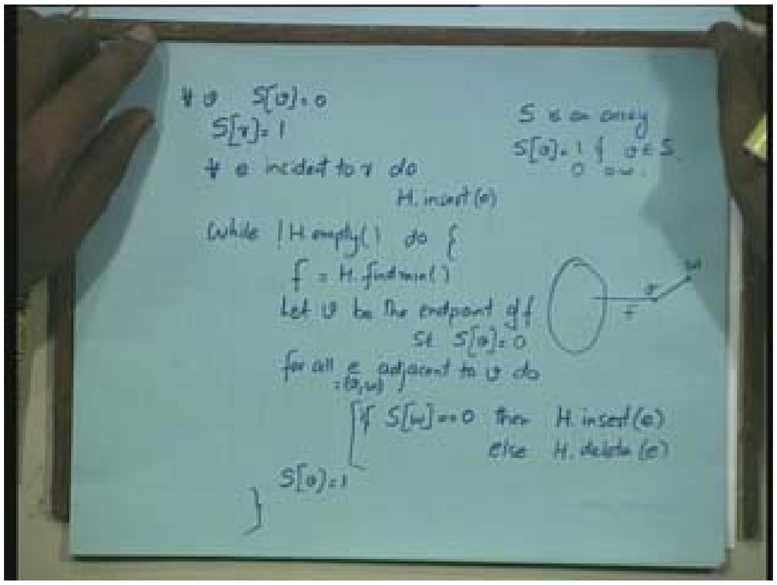
Now, what should I do? I should insert the edges incident at the root into the heap. So, what should I do? For all e incident to r, do H that is my heap, dot insert e. Some such thing, I have to insert the edge into the heap. So, everyone follows … (hindi) so (hindi) while not, I would say not H dot empty. (hindi) do. Now, what should I do? I should find the minimum edge. H dot find min equals let say, this is equal to, what will this return? This will return in edge, (hindi) and now what should do I with f now? (hindi)

I have to find the end point of f which is not in S. Let V be the end point of f such that S of V equals 0. (hindi) For all e, e adjacent to v do. What do I do? For all e equals V comma W (hindi) If S W equals 0 (hindi) then edge dot insert e, not w; (hindi) else H dot (hindi) this is the spanning tree, minimum spanning tree. (hindi) this entire thing is the part of the this loop. Clear to everyone? (hindi) As you can see, all the effort is being spent in this step or not too much actually in this step. How many times is this loop going to be executed? n times?

How many times is this loop, the while loop is going to be executed? n minus 1 or n times? n minus 1 times. Why, because every time I execute this loop 1 vertex comes into the set as n vertices (hindi) So, this is getting executed n minus 1 time. So, this statement is executed n minus 1 time but, this is again just a constant time operation. So, this is not too much time. (hindi)

It is this way the most of the work is getting done. For every if you are looking at all the vertices adjacent to the vertex V, degree and for each one of those vertices or edges, you are spending, doing either an insert or a delete. Log n, degree times log n, you have seen that before, summed over all vertices will be m log n. There is a modification to this scheme that can also be done.

(Refer Slide Time: 45:58)



```
∀ v  S[v] = 0
  S[r] = 1
∀ e incident to r do
                H.insert(e)
while !H.empty() do {
    f = H.findmin()
    let v be the endpoint of f
              st S[v] = 0
    for all e adjacent to v do
        =(v,w)
        [ if S[w] == 0 then H.insert(e)
                       else H.delete(e)
    S[v] = 1
}
```

S is an array
$S[v] = \begin{cases} 1 & v \in S \\ 0 & o.w. \end{cases}$

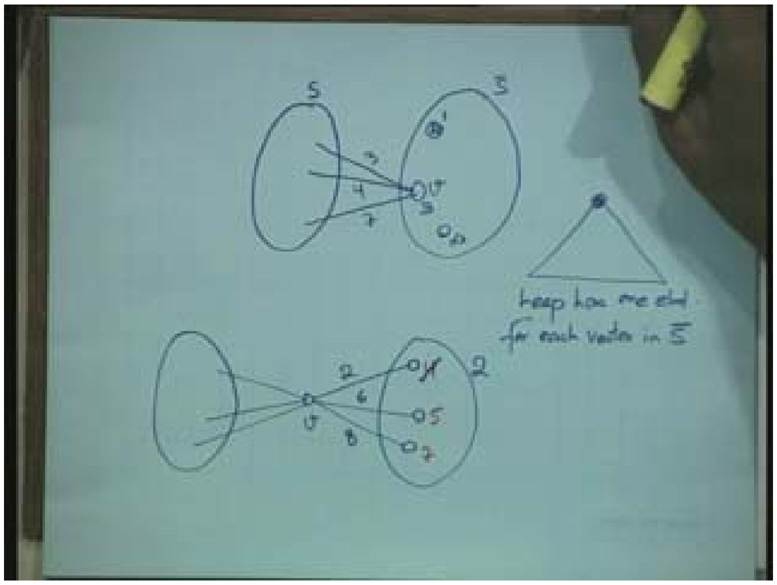
In this modification, what you do is you have your set S, you have your set S complement. Each vertex in S complement has a number on it. What is this number going to be? Let us look at all the edges going from this vertex to S. Suppose, these were the 3 edges going from vertex V to S, suppose these edge lengths are 3, 4, and 7 then this vertex is going to have a number of 3 return … (hindi)

What are we going to do with these numbers? We are going to create a heap. Now, my heap is a heap of vertices and not of edges. So, this heap has 1 element for each vertex in S complement. (hindi) What will be the minimum element in the heap? It will be the vertex which has the minimum edge incident at it among all the edges in the cut. So, it will be the vertex which has to go across. (hindi)

If I find the vertex the vertex which is sitting here at the top, the vertex with the minimum value, minimum label, let us call this labels: the vertex with the minimum label let say, it is this vertex (hindi) so, that means that the minimum edge going across this cut is 1. We are not keeping track of edges which are here, within here. This vertex will only contain a label which is equal to the minimum of the edges which are going across the cut. If there is a vertex which has no edge going across the cut, incident at it (hindi) so the minimum will tell me which is the minimum edge going across and that is the vertex that I have to move across and when I have to move across the vertex, what do I do? How do I update information? Very simple, this is vertex V (hindi) all those in S bar which are adjacent to this vertex. (hindi)

The minimum edge that is coming to this vertex, so far is 4. But now, this edge is also coming to this vertex, because this vertex going on that side (hindi) So, that is the only things we have to do now. So, what is the operation we have to do on the heap now? Update what? Update or decrease? We will only decrease the label, decrease priority. Decrease priority is the operation we have to do. Decrease priority (hindi) no insert, no delete. Find min, of course and decrease priority (hindi) since, I have 4 or 5 minutes, let me write down the code for even this 1. (hindi)

(Refer Slide Time: 52:23)



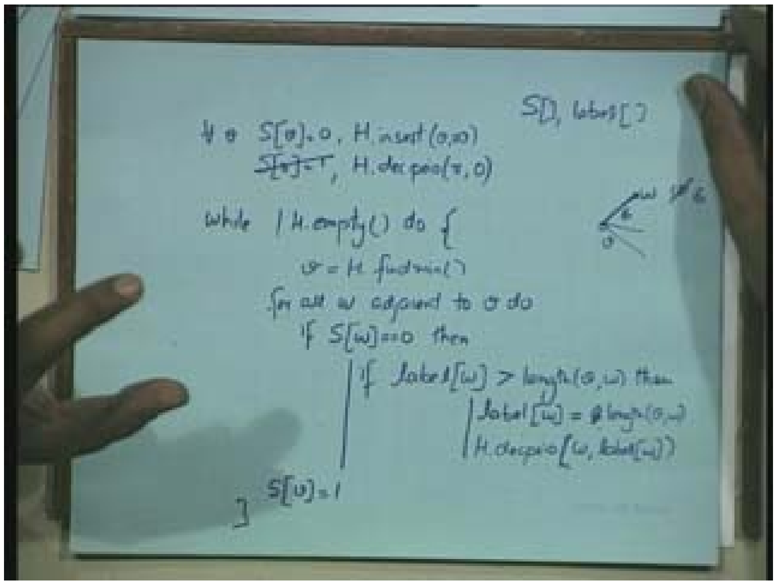
So, once again I will, as in the previous case I will have my array S, I will do the same initialization; for all vertex V, S V is initially 0 and S of r is 1. Now remember, the heap is going to contain vertices in S complement. H is going to contain vertices in S complement. What should I put initially in my heap? Everything? Except r? So for all V, S V is equal to 0 and let say, I do H dot insert V comma (hindi) distance from r, what does that mean? What does that mean? Length of the edge from r? Infinity (hindi)

H dot insert (hindi) No, no, everything is infinite only r is 0 (hindi) while H is not empty, we will do something. What will we do? We will take the minimum let say, that is the vertex V. I will take the vertex (hindi) for all W adjacent to V do. If W is adjacent to V what should I do? If S W equals 0, what does it mean? W is on the S complement side. Then what should I do? Then I have to update its priority. Then if what should be the new, what is the option here?

If, we will need a label array somewhere. If label of W (hindi) which is keeping track of all the labels, so if label W (hindi) is greater than length of V comma W then that means that this edge now, so what is it, what are they saying? (hindi) then label W equals 6, sorry, equals length V W and H dot decease priority W comma label W. (hindi)

So, this is an alternative way of doing. So, the same running time complexity. You will check that yourself. Why is that? because, once again we are looking at degree, for spending degree time at each vertex and degree times log n, because decrease priority also takes log n time.

(Refer Slide Time: 10:04)



```
S[], label[]

∀ v  S[v]=0, H.insert(v,∞)
     ~~S[s]=T~~, H.decprio(s,0)

while !H.empty() do {
      v = H.findmin()
      for all w adjacent to v do
        if S[w]==0 then
          | if label[w] > length(v,w) then
          |      | label[w] = length(v,w)
          |      | H.decprio(w, label[w])
  S[v]=1
}
```

Some over all vertices will come m log n and that is basically where most of the work is getting down (hindi) so with that I am going to end today’s lecture. So, we looked at Prim’s algorithm for computing minimum spanning trees and we saw 2 ways of implementing it.