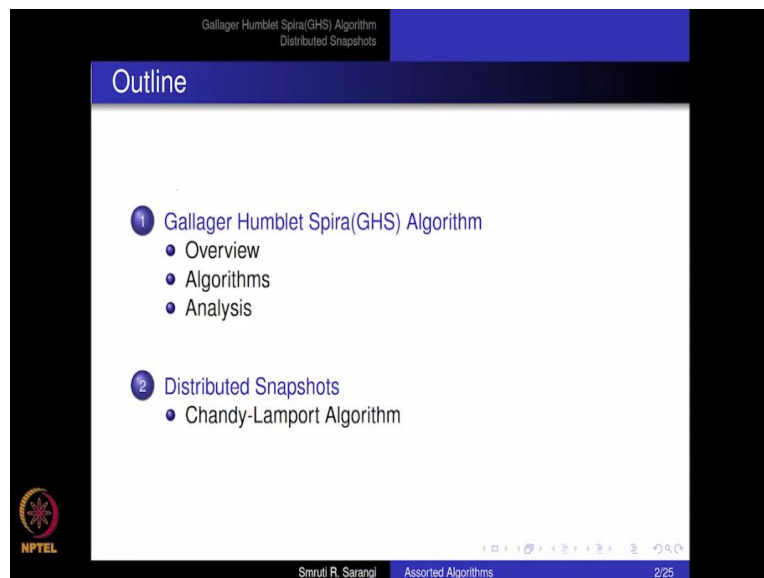


Advanced Distributed Systems
Professor Smruti R Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
Lecture No 10
Distributed Minimum Spanning Tree and Distributed Snapshots

Welcome to the lecture on Distributed Minimum Spanning Trees and Distributed Snapshots. So, what we had seen in the earlier lecture on leader election that if we are able to somehow overlay a tree on top of a network, on top of distributed nodes if you can somehow overlay a tree instead of a ring which we have been traditionally used to we doing. So, then what we can do is many distributed algorithms becomes simpler. For example, electing a leader, finding the smallest element in the minimum finding all of that becomes much easier.

So, in this lecture as well we will discuss one way of taking a snapshot of a network that also becomes easier if we have a tree. So, which tree should we choose? Well, a good tree is a minimum spanning tree the reason being it minimizes the length of the edges so it kind of keeps things close by.

(Refer Slide Time: 01:22)



So, we will discuss the Gallager Humblet Spira Algorithm the GHS Algorithm. We will discuss the overview the algorithm and the analysis and then everything about distributed snapshots.

(Refer Slide Time: 01:42)

The slide is titled "Properties of an MST" and is part of a presentation on the "Gallager Humblet Spira (GHS) Algorithm" for "Distributed Snapshots". It contains the following sections:

- Uniqueness:** If each edge of the graph has a unique weight, then the MST is unique.
- Construction based on Least Weight Edge:**
 - A fragment is a sub tree of a MST.
 - An outgoing edge of a fragment has one endpoint in the fragment, and one node outside the fragment.
 - Proposition:
- Theorem:** If F is a fragment and e is the least weight outgoing edge, then $F \cup e$ is also a fragment.

The whiteboard contains handwritten notes in red ink:

- Kruskal's algorithm
- Prim's algorithm
- Inductive

Below the text are three diagrams illustrating the concept of a fragment and its extension:

- A simple graph with a highlighted subgraph labeled F .
- A graph where a highlighted subgraph F is connected to a node outside it by an edge e , forming $F \cup e$.
- A more complex graph with a highlighted subgraph labeled "fragment" and an edge e connecting it to the rest of the graph, forming $F \cup e$.

So, let us first look at some basic properties of an MST. So, it is important that before the MST is understood the following algorithms the Kruskal's algorithm and the Prim's algorithm both of these are understood quite thoroughly including the groups. It is very important to go through both of these algorithms. The Prim's algorithm and a Kruskal's algorithm for sequential MST finding and the proof of the Prim's algorithm particularly is very important.

So, the inductive proof of the Prim's algorithm you should go through it. So, now I will have suggested a few, well so I have already suggested algorithms, but you can look at it from a popular text on algorithms first and the proofs are important.

So, now without proving, without going in a depth I will list a few properties of an MST that we shall use. The first is the property of uniqueness which says that if each edge of the graph has a unique weight then the MST is unique. So, this goes without saying this is easy to prove. So, this is our first point that in each edge of the graph is unique then the MST on a whole is unique.

So, you will not have two MST. Of course, if you have edges with same weights then you could have a non-unique MST in a sense two MST with a same weight otherwise it will not happen. Furthermore, here is one more theorem that is a direct outcrop of the proof of the Prim's algorithm which says that since construction based on the least weight edge. So, let us consider a fragment as a sub tree of a MST.

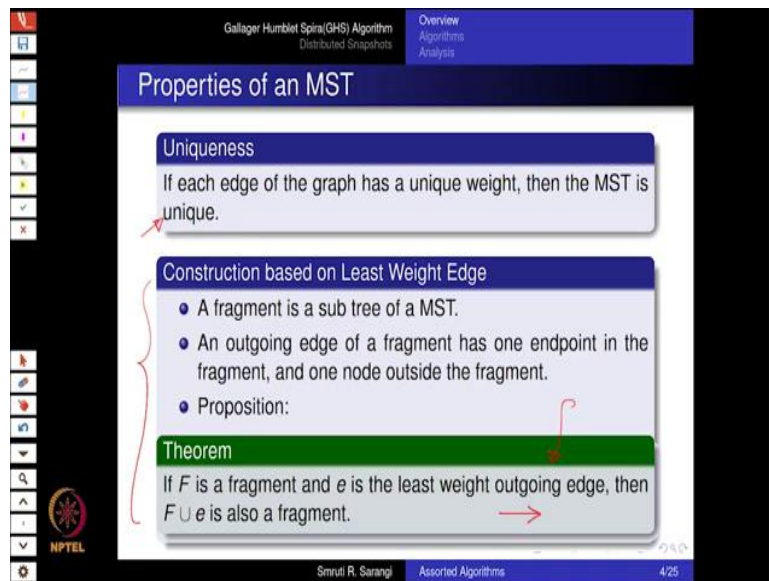
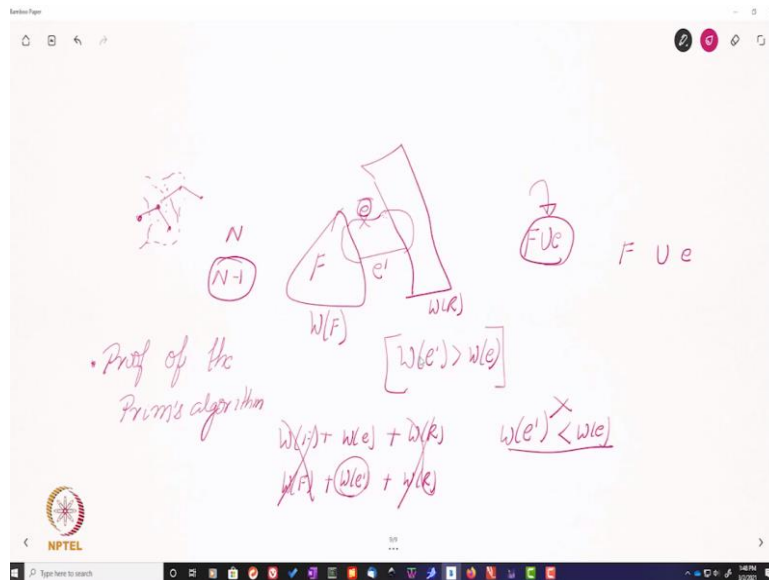
So, if we take a minimum spanning tree so let us say that this is the tree. So, let us take a sub tree of the minimum spanning tree and let us call this a fragment. So, let us refer to this as a fragment. So, then an outgoing edge of a fragment has one endpoint in the fragment and one node outside the fragment. As you can see over here this is the outgoing edge. So, this has one node within the fragment and one node outside the fragment.

So, we basically have a sub tree then we have the rest of the tree over here and there is one edge that connects this fragment with the rest of the tree and so basically we now are looking at some property of this edge. So, for anybody who knows the proof of the Prim's algorithm this theorem will be rather obvious that if F is a fragment and e is the least weight outgoing edge then $F \cup e$ is also a fragment.

What does this mean? That F is a fragment and this is the edge e . so if I consider this to be the fragment and this to be rest of the world and then I draw a line over here. So, there might be multiple edges that go from an edge of the fragment to the rest of the tree. So, this could be one edge, this could be one more this could be one more. So, let them be e, e', e'' .

So, what the theorem is saying just look at it if e is the least weight outgoing edge which means that out of all the edges that connects this fragment to the rest of the tree, if e is the one which has the least weight then the claim is that $F \cup e$ which basically means that I can create a new fragment like this and this will also be a fragment of the MST in the sense that e will be an edge which is the part of MST that makes $F \cup e$ also a fragment of the tree. So, this of course is easy to prove. So, let us look at this once again.

(Refer Slide Time: 06:01)



So, let us say that this is a sub tree, this is a fragment and we have one edge e to the rest of the nodes. And so then currently it is a tree we claim that this is the MST. Let us assume this is not the case well if this is not the case then what would happen then it would mean that there is some other edge e' which is a part of the MST and e is not a part of it. Well, but what you would see is that if let us say now I add e' this will clearly create a cycle.

In this cycle, we know that the $W(e') > W(e)$ and let us say I remove e , we are claiming that this is the MST, but I claim that there is a contradiction the reason is that if add edge e . So, let us assume that the rest of the tree remains the same and its weight is fixed. So, then if I add edge e so let us maybe say that the weight of the fragment is $W(F)$ and weight of the rest of is $W(R)$.

So, let us consider two trees one that has e and one that has e' . So, the tree that has e its weight is $W(F) + W(e) + W(R)$, rest of the tree and the tree that has e' its weight will be again the weight of the fragment plus its own weight because at that point edge e will not be there plus the weight of the rest of the tree.

So, clearly these parts are common and we claim that this is the MST, but this cannot be the case because for this to happen $W(e') < W(e)$ which we clearly know it is not correct because $W(e') >$ and so we clearly know that this is not correct. Hence any out of all the outgoing edges the least weight outgoing edge which is edge e in this case has to be a part of a MST.

And $F \cup e$ will thus become a fragment. So, this is exactly the intuition that is used in the Prim's algorithm to iteratively or I should say recursively increase the size of the tree. So, what we do is we first consider the starting node as a single node then we look at all of its neighbors, take the least weight edge. So, this becomes a new fragment then again we draw another boundary around it then we pick the least weight edge.

Again we draw another boundary around it again we pick the least weight edge maybe this is the one. So, we gradually keep on expanding the boundary and we keep on adding edges, but our criterion always is that for the boundary around the fragment that we have created we just pick the least weight edges and we just keep on adding them and given the fact that we have proved this theorem now the $F \cup e$ maybe I will write it in a slightly better form.

$F \cup e$ is a fragment we are always sure that the edge that we are adding is a part of MST. So, if we continue to grow the tree ultimately we will encompass all the nodes and any N node tree will have $N - 1$ edges. So, when we have $N - 1$ edges we will know we are done and the proof is by induction. Given the fact that at every step we start from an MST for that fragment and adding a new edge still maintains the MST property.

We can prove that when we reach the end which is when we cover all the N vertices with $N - 1$ edges a tree continuous to remain an MST. Of course, if you are able to understand what I said then I would suggest that you do not go forward because you will not be able to understand the rest. You first take a look at the proof of the Prim's algorithm that is the most important.

So, you first take a look at the proof and then you try to understand this theorem over here if this theorem is understood then only you proceed otherwise you do not.

(Refer Slide Time: 10:36)

The screenshot shows a presentation slide titled "GHS Overview" from a course on "Gallager Humblet Spira (GHS) Algorithm" and "Distributed Snapshots". The slide includes handwritten notes in red: "Unique edge weights, Conn" and "fuse". A bulleted list describes the algorithm's process:

- Initially each node is a fragment.
- Gradually nodes fuse together to make larger fragments. A fragment joins another fragment by identifying its least weight outgoing edge.
- The nodes in a fragment run a distributed algorithm to cooperatively locate the least weight outgoing edge.
- Gradually the number of fragments decrease.
- Ultimately there is one fragment, which is the MST.

The slide also features a navigation sidebar on the left, an NPTEL logo, and a footer with the name "Srinull R. Sarangi" and "Assorted Algorithms".

So, the overview of GHS is like this that we want to take Prim's algorithm and create a distributed version of it. So, initially each node is a fragment so initially every single node is a fragment. Gradually what happens is nodes fuse together to make larger and larger and larger fragments something that we also saw in ring based leader election where the windows kind of grow larger, larger, larger and larger.

So, in this case, the fragments fuse together to make larger and larger fragments and the fragments of course joins another fragment why are the previous theorem which is this theorem which is by identifying the least weight outgoing edge. Furthermore, how to find the least weight outgoing edge? Well, all the nodes within a fragment run a distributed algorithm to find the least weight outgoing edge.

Gradually what happens is that the number of fragments this number itself decreases ultimately only one fragments remains which covers the entire set of nodes and of course in this case we are assuming that the graph is connected and then that is the MST. So, one assumption we make is that of course we have unique edge weights that gives us a unique MST that is one. And the other is that the graph is connected. So, these are the two assumptions, but this assumption is made by other algorithms as well nothing special over here.

(Refer Slide Time: 12:21)

Gallager Humblet Spira (GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

Properties of a Fragment

$(F_1) \rightarrow (F_2)$

Properties of a Fragment

- Each fragment has a unique name.
- When two fragments combine, then all the nodes in one fragment change their name to a new name.
- Each fragment has a level.
 - Assume that (F_1) is combining with (F_2) . It can only do so if $level(F_1) \leq level(F_2)$.
 - If $level(F_1) < level(F_2)$ then all the nodes in F_1 take on the name and level of (F_2) .
 - If $level(F_1) = level(F_2)$ then the level of both of the fragments gets incremented by (1) .
 - The nodes of $F_1 \cup F_2$ get assigned a higher level (old level++).

NPTEL

Smruti R. Sarangi Assorted Algorithms 6/25

So, what are the properties of a fragment? So, now we are getting into our distributed algorithm not completely, but we are kind of looking at it from the outside. So, let us give each fragment a unique name, unique ID. So, when two fragments combine then all the nodes in one fragment will change their name to a new name. So, what you see is that if two fragments are combining to create a bigger fragment then of course a new name has to be assigned the same way the two company is merged.

So, what happens is that typically if a large company gobbles up a small company then no name is changed, but if two equal size companies kind of merge then the name kind of reflects both. We will see something similar happening with fragments that assume that fragment F1 is combining with fragment F2. We will see it can only do so if level of $F_1 \leq F_2$.

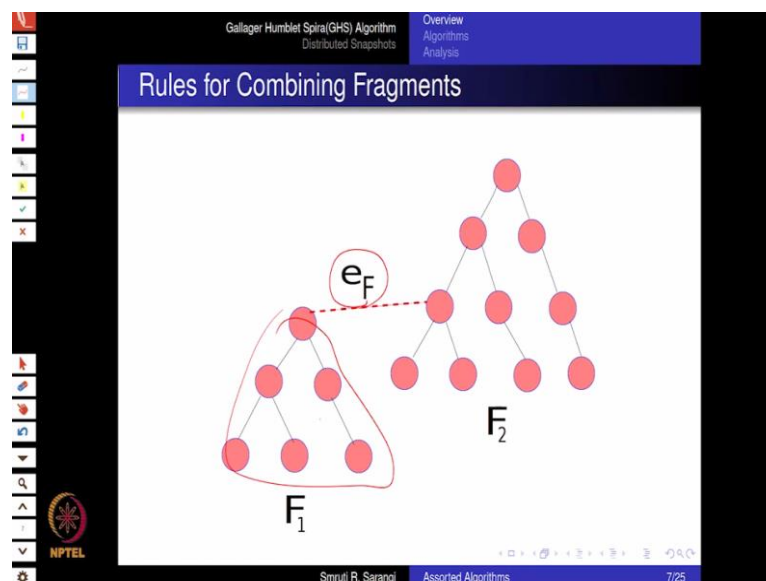
What this means is that if one fragment F1 is joining fragment F 2 it can only do that if F 1 is the smaller guy is smaller or is equal. So, which basically means a bigger fragment cannot gobble up a smaller one, but a smaller one can always approach a bigger one or one of same size asking it to join. If level of F 1 < level of F 2. So, level is somehow indicative of its size we will see how.

If level of F 1 < level of F 2 then all the nodes in F 1 take on the name of level of F 2. So, which basically means if a smaller company joins a big company like a multinational then all the nodes in the smaller company F 1 will take on both the name and the level. So, every fragment has a name and a level and the level is somewhat indicative of its size. So, if level F 1 is less than that then F 1 loses its identity.

So then the nodes of F_1 will take on the names and levels of nodes in F_2 . If however is an important point if you can mind if however level $F_1 = \text{level } F_2$ then the level of both the fragments gets incremented by 1. So, this is important that if two fragments with an equal level are merging then what happens is that the level of the combined fragment increases by 1 gets incremented by 1 that is how the level increases.

Furthermore, we will see what happens with a name so that is also interesting they get a new name and the new name is basically something that kind of combines the names of both. So, we will see in a couple of slides how that happens, but pretty much the levels are equal it is more complicated than when the levels are not equal. The nodes of $F_1 \cup F_2$ get assigned to higher level which is the old level plus, plus.

(Refer Slide Time: 15:36)



So, what we are saying is that look we have a small fragment, we have a big one and we are combining them, the combining edge is here. So, if let us say $F_1 < F_2$ in terms of levels then of course the name and level of F_2 gets transferred over here, but if they has the same level then the levels of both is incremental and a new name is given to both. So, it is not that it becomes one big homogenous fragment.

(Refer Slide Time: 16:10)

Gallager Humblet Spira(GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

Combining Rules

Let (F_1, L_1) be desirous of combining with (F_2, L_2) . e_{F_1} is the least weight outgoing edge of F_1 and it terminates in F_2 .

RULE LT
If $L_1 < L_2$, then we combine the fragments. All the nodes in the new fragment have name F_2 and level, L_2 .

RULE EQ
If $L_1 = L_2$, and $e_{F_1} = e_{F_2}$ The two fragments combine, with all the nodes in the new fragment having:

- The level is $L_1 + 1$
- The name is e_{F_1}

 $F_1 \cup F_2$

RULE WAIT
Wait till any of the above rules apply.

Smriti R. Sarangi Assorted Algorithms 8/25

Gallager Humblet Spira(GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

Rules for Combining Fragments

Smriti R. Sarangi Assorted Algorithms 7/25

So, we will now define two important combining rules and one waiting rule. So, let F_1, L_1 . F_1 is fragment F_1 with level L_1 be desirous of combining with F_2, L_2 where e_{F_1} is the least weight outgoing edge of F_1 and it terminates in F_2 . So, between F_1 and F_2 , e_{F_1} is the least weight outgoing edge from F_1 . So, in consonance with what we have said there are two combining rules one is the less than rule LT rule.

If $L_1 < L_2$ then we combine the fragments all the nodes in a new fragment will have name F_2 and level F_2 . It is like a smaller guy merging with the bigger one which we have discussed in this slide as well. If $L_1 < L_2$ that is what we do that nodes in a new fragment will have the name F_2 and the level L_2 , but if they are equal that is where we said that there is a catch.

If the levels are equal, then we check if the least weight outgoing edges are the same or not. So, this is the catch over here. So, in this case we will combine when we have the same levels subject to the fact that our outgoing edges are the same. If they are not we will not because the two fragments combine with all the nodes subject to this. So, then it is important even if their levels are equal they just do not combine like that only the LT rule if the levels are not equal then only L 1 will combine L 2.

Otherwise, we will see that their outgoing edges have to be same least weight outgoing edges only then we combine and then as we have discussed the final level is $L + 1$ and we also discuss that we will give both the fragments and the nodes within them a new common name and a new common name is basically the name of the edge. So, let us assume that every edge has a unique weight and unique name also and the name could be just a combination of the two node IDs of the edge.

So, it could be that does not matter howsoever the name is derived, but we will essentially the edge will be the common name for the nodes of both the fragments $F1 \cup F2$ and if any of these above rules do not apply we just wait, we wait for them to apply. So, this is basically telling us that what we are doing is that small fragments will always go and merge with bigger ones.

But equal size fragments will have a key condition which is that their least weight outgoing edges need to be the same only then they will actually merge so they will increment their level and the new name will be equal to the edge that connects them the least weight outgoing edge.

(Refer Slide Time: 19:30)

The slide is titled "Variables" and is part of a presentation on the Gallager Humblet Spira (GHS) Algorithm. It lists the following variables and their meanings:

- state** : sleep, find, found
 - sleep** The node is not initialized
 - find** The node is currently helping its fragment search for e_F .
 - found** e_F has been found
- status [q]** basic, branch, reject
 - basic** Edge is unused.
 - branch** Edge is a part of the MST.
 - reject** Edge is not a part of the MST.
- name** Name of the fragment.
- level** Level of the fragment
- parent** Points towards the combining edge.
- bestWt, bestNode, rec, testNode** temporary variables

Hand-drawn diagrams include: a node p with an arrow pointing to q ; two fragments F_1 and F_2 with arrows pointing to a common edge; and a larger fragment F_1 with a smaller fragment F_2 being combined, with an arrow labeled LT pointing to the combining edge.

So, now we will discuss an algorithm it is a fairly long algorithm. So, we will have to discuss the state that we maintain. So, we have three states sleep, find and found. Sleep means the nodes has not been initiated. Find means the node is currently helping its fragment search for e_F , e_F is the least weight outgoing edge. Found means e_F has been found or the least weight outgoing edge has been found. So, that is what this means.

So, here also I am p and the other node is q . So, every node maintains an array called status q . So, status q is basically the status of the edge from p to q . So, status q where I am p and other nodes are q . So, for every q which is my neighbor I will have a status array with a q th entry. So, then it will have three values basic, branch and reject. Basic means the edge is unused, branch means edge is part of MST.

Reject means the edge is definitely not a part of MST basic, branch and reject. Basic means as of now we do not know its status, branch means we know its status and we know it is a part of a MST. Reject means we know its status and we know for sure that is not a part of MST then we have discussed name and level, name of the fragment, level of the fragment, parent so the parent basically says the following that let us say two nodes for the same level combine or let us say even a smaller node combines with a bigger node.

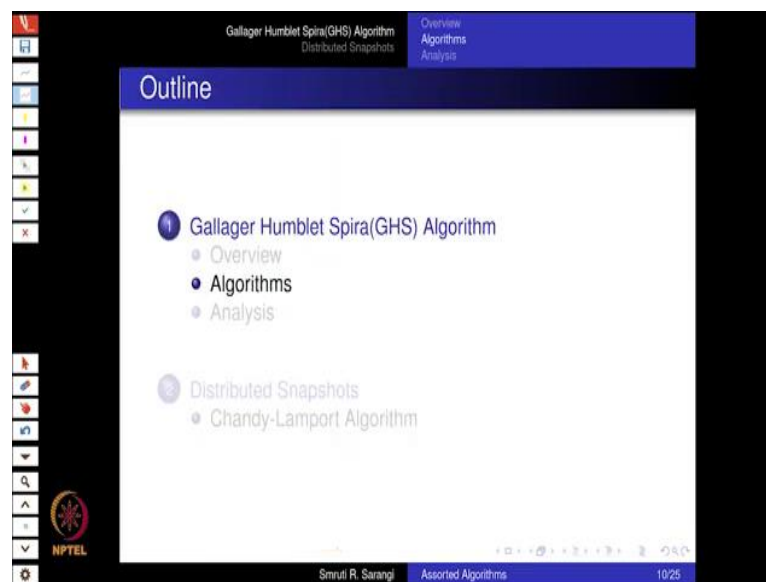
So, there will always be a combining edge. So, let us consider its first combination with the LT rule. So, let us say that this is a small fragment and this is a much bigger fragment. So, in this case if this is the much bigger fragment. Every node over here will basically point to some

other node which points to some other node which will ultimately take it towards the combining edge. So, every node will point towards the combining edge.

So, these are essentially parent pointers which take us towards what is called the combining edge and similarly if we have a combination of two fragments where the level was the same. So, every node here will also have parent pointers towards this combining edge, the common combining edge and every node here also will have a parent pointer that goes towards the common combining edge.

So, we will see why this is the case, but this is how the parent pointers actually work and then of course we have a bunch of temporary variables like best weight, best node, best node and so on which are purely temporary variables.

(Refer Slide Time: 22:41)



Gallager Humblet Spira (GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

Initialization

Current node p . Neighbor q .

Algorithm 1: Initialization

- pq is the least weight edge from p

```

status[q] ← branch
level ← 0
state ← found
rec ← 0
send <connect,0> to q

```

Smriti R. Sarangi | Assorted Algorithms | 11/25

So, now let us discuss the algorithms. So, the algorithms we have many algorithms 8 or 9. So, always the assumption is that the current node is p and the neighbor is q .

So, let us look at the initialization. So, let us say that initialization is when things are starting. So, let us say I am node p and from node p p, q is the least weight branch. So, clearly if this is the initiative I can draw a circle around it p, q intersects this circle and this needless to say is the least weight edge. So, I set the status of this edge as a branch status q as branch. I start from level 0 and my state is found I found a least weight edge.

$rec \leftarrow 0$ means we will see what rec means, but at the moment for this case it is 0 and I send a connect message to q the p will send the connect message to q and this is when I am initializing. So, this is when I am starting I know that the level of q will at least be my level or we will see essentially as far as least weight edge let me send a connect message and let us see what q does.

If q response, I connect otherwise I do not. So, what I send is like I send a connect 0 message we will see in a second what does 0 stands for, but essentially as far as I am concerned if q is my least weight edge I request you to kindly connect with you and I do that by sending a connect message and you can clearly see that in this case p, q will be a branch of MST. So, there is no reason why it should not be.

(Refer Slide Time: 24:46)

Gallager Humblet Spira(GHS) Algorithm Distributed Snapshots Overview Algorithms Analysis

Process connect Message

Algorithm 2: Processing of the connect message

```
1 Receive <connect,L> from q:
  if L < level then
    /* Combine with rule LT */
    status [q] ← branch
    send <initiate,level,name, state > to q
  3 end
  4 else if status [q] = basic then
    5 wait
  6 end
  7 else
    /* Combine with rule EQ */
    8 send <initiate,level+1,pq,find > to q
  9 end
```

Smriti R. Sarangi Assorted Algorithms 12/25

Gallager Humblet Spira(GHS) Algorithm Distributed Snapshots Overview Algorithms Analysis

Process connect Message

Algorithm 2: Processing of the connect message

```
1 Receive <connect,L> from q:
  if L < level then
    /* Combine with rule LT */
    status [q] ← branch
    send <initiate,level,name, state > to q
  3 end
  4 else if status [q] = basic then
    5 wait
  6 end
  7 else
    /* Combine with rule EQ */
    8 send <initiate,level+1,pq,find > to q
  9 end
```

Smriti R. Sarangi Assorted Algorithms 12/25

Gallager Humblet Spira(GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

Initialization

Current node p . Neighbor q .

Algorithm 1: Initialization

- pq is the least weight edge from p

```

status[q] ← branch
level ← 0
state ← found
rec ← 0
send <connect,0> to q

```

NPTEL

Srinati R. Sarangi Assorted Algorithms 11/25

Gallager Humblet Spira(GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

Combining Rules

Let (F_1, L_1) be desirous of combining with (F_2, L_2) . e_{F_1} is the least weight outgoing edge of F_1 and it terminates in F_2 .

RULE LT
If $L_1 < L_2$, then we combine the fragments. All the nodes in the new fragment have name F_2 and level, L_2 .

RULE EQ
If $L_1 = L_2$, and $e_{F_1} = e_{F_2}$. The two fragments combine, with all the nodes in the new fragment having:

- The level is $L_1 + 1$
- The name is e_{F_1}

RULE WAIT
Wait till any of the above rules apply.

NPTEL

Srinati R. Sarangi Assorted Algorithms 8/25

Now, coming to algorithm 2 which is the processing of the connect message. So, when I receive a connect message the message type is connect and L is the level of the sender. So, in this case the level of the sender is 0. So, when a node receives a connect message of course you can say p sent it to q , but what is our convention. Our convention is that I am always node p and the other node is q .

So, as far as I am concerned I am node p , I am getting a connect message from another node which is q . So, this is the convention that we adopt might be confusing, but this is what we do. So, now what I do is that look I have gotten a connect message so I will look at my level and the level of the connector. So, if $L > \text{level}$ which means that the level that is coming along with the message if that is less my level.

No problem this means that a smaller fragment wants to combine with a larger fragment. So, I can happily combine with the rule LT so absolutely no issues. So, I will set the status q to branch and then I will send an initiate message to the smaller fragment this is a large fragment I will send initiate message to the smaller fragment saying that look I have accepted your connect message you as of now you are initiated.

This is your new level, this is your new name, this is your new state and a state is whatever is my state. If currently I am searching for my least weight edge now you are a fragment and you join me so you need to help. So, whatever is my state that is currently your states you take it and you initialize yourself. No problem so this is sent to q . So, just a quick disclaimer before this point.

This p and q business can be confusing because you will argue that look in the previous slide p sent a connect message to q . Now you are saying that I am p and I received a connect message from q how is this possible? Well, in a distributed algorithm you are looking at distributed state machines where every node is an independent computing entity. This is a node it gets a message based on that it updates its state table.

And then it sends messages to other nodes including the one that sent the message to you. So, that is the reason when distributed algorithms are written this might sound tricky and confusing, but I am always node p whoever is doing the action and whoever is sending or receiving a message the entire world outside is always node q . So, this of course is complicated, but if you can appreciate this complexity then appreciating such algorithms will become much, much easier.

So, let us come back to our discussion I will clean the slide. So, the idea here was that I am combining with rule LT primarily because here I have a case where I have a small fragment that is requesting a bigger fragment to connect and the bigger fragment has no issues at all. So, it set status q to branch and sends an initiate message back to the smaller fragment saying that okay look this is my name, level and state.

Henceforth, this is your name, level and state as well, else if this condition is not holding and the status of the node is basic. So, the status of the node basically has not been as far as I am concerned this node has not been explored so the status[q] = basic. So, then what I will do is I will combine with rule EQ.

So, why will rule EQ be useful over here and why not you know so we are automatically seeing that a status of the node is basic. Does it automatically mean that $L > \text{level}$ and do the conditions for rule EQ actually holds. Well, you will see that they actually hold and so you will see that they actually hold and there is no error over here, but it is important to remember this we might come back to this point. So, what we do is that we combine with rule EQ. So, for this we send the initiate message.

So, why EQ will come here at the moment in leap of faith, but we will break that. When we send a initiate message. So, we are saying that they are at the same level. So, well no problem so your new level is $\text{level} + 1$ and your new name is the edge why of which this message is coming the joining edge which is pq , so p on one side and q on the other side and this is how we are joining.

And so, the name of the edge is $p q$ and furthermore, given that we have joined our new state should be equal to find. Find basically because now we have bigger a bigger fragment. So, we further need to expand our fragment which means that we need to find our least weight outgoing edge and grow. So, your state is fine as well as my state also has to become fine. So, this is basically what we do that we send an initiate message to the new fragment. The new fragments start the process of finding.

So, we did make certain assumptions here we have not proven, but let us continue. So, the application of rule LT and rule EQ would be very clear over here. If you want, you can just go back to this slide where we define the EQ and LT rules and as you can see it was not all that complicated. So, here we basically LT was the smaller fragment to the larger fragment and EQ was both the fragments of the same size and they have the same least weight outgoing edge. So, the question that we have kept open is why will the rule EQ be useful over here which we will gradually see why.

(Refer Slide Time: 31:44)

The slide displays the following algorithm:

```
Algorithm 3: Processing of the initiate message
1 Receive <initiate,level',name',state'> from q:
  /* Set the state
2 (level,name, state) ← (level',name',state')
  parent ← q
  /* Propagate the update
3 bestNode ← φ
  bestWt ← ∞
  testNode ← none
4 foreach r ∈ neigh(p): ( status[r] = branch) ∧ (r ≠ q) do
5   send <initiate,level',name',state'> to r
6 end
  /* Find least weight edge
7 if state = find then
8   rec ← 0
   findMin()
9 end
```

Handwritten red annotations on the slide include: 'children' pointing to the loop in step 4, 'MST' pointing to the condition in step 4, and 'rec ← 0' pointing to step 8. There are also some diagrams of nodes and edges.

Now how do you process the initiate message. So, let us say that again node p it is always node p gets an initiate message from node q. So, the message type is initiate we get a level dash, name dash and state dash. So, we set the state no problem, we set our level name and state so level dash, name dash and state dash. So, my level is now level dash, my name is name dash and state is state dash.

Furthermore, given the fact that this is the combining edge I set my parent equal to q. So, if this is p q I set my parent \leftarrow q then what I do is I propagate the update. So, I defined a few local variables best node, best weight and test node we will see what these are. So, for each of my neighbors for each $r \in \text{neigh}(p)$. So, basically for each of my neighbors as long as the $\text{status}[r] = \text{branch}$.

So, essentially I propagate this along the small MST that I have created. So, along with the small little MST within my fragment I forward this message which means that the status has to be branch this means it is part of this MST fragment and furthermore, $\wedge r \neq q$ which means I do not send the message back to my parent I have only sent it to my children. So, this indicates that I am only sending the message to my children and I am sending the initiate message which means that look now your fragment has combined with some other fragment.

So, now we are under the rule of a new fragment or let us say we all have changed our name and level. So, I have already done that now you do it. So, that is the reason we just send an initiate message with a new level name and state to let us say a child r. No problem the child r also does the same thing so on and so forth. What you can clearly see is that all of them

ultimately end up pointing indirectly of course towards the combining edge. Indirectly means by a parents no problem.

Now, what we do is we see what is the state? if the state is equal to find then it means that I am supposed to play my job as a good member of a fragment by finding the least weight outgoing edge. So, I set the rec variable to 0 again an internal state variable and I call the function find min such that we all members of the new fragment can find the least weight outgoing edge.

(Refer Slide Time: 34:41)

Now, problem find min, find min is not hard at all. So, find min by the way is not a message as you can see it is an internal function. So, the internal function is being called over here find min. Find min this is what it says. So, again I am p the other node is q as long as there is $\exists q \in \text{neigh}(p)$ which means I look at all the neighbors of p as long as there is some q which is a neighbor such that $\text{status}[q] = \text{basic}$ which means that as far as p is concerned is an unexplored edge and out of all of its outgoing edges.

So, out of all of its outgoing edges that are basic $w(pq)$ is minimal which means that all of his candidate outgoing edges. So, clearly if the status of an edge is branch or reject it cannot be a candidate outgoing edge. It can only be a candidate outgoing edge if its status is basic which means it is unexplored. So, if it is unexplored out of all of them I find the node q such that p q out of this set is minimal no problem.

Then I say that $q \rightarrow$ test node so then I try to check if it is possible to add q to my fragment and kind of grow q I mean grow the fragment via this pq edge. So, what has happened is that look

two fragments are merged. There has been a common merging edge then initiate messages have been send. So, now let us say every node in at least the new fragment is aware that its boss has change.

If let us say the boss was in the find state then all the nodes in the joint fragment will also be in a find state, so boss means the larger fragment. If both the fragments at the same level then what we will see in a few future algorithms is algorithm 4 in like algorithm 6, 7, 8, 9 is that the same level they will both of them will actually send initiate message to each other and then they will increment the levels in both the fragments and then they will also shift to the find state which means that they will start to find the least weight outgoing edge.

So, every node will try to do its part. So, every node what it will do is it will scan all of its basic edges, neighbors whose status is basic find the minimum one and try to see if a connection can be initiated with it. So, it will do a test, so it will send a test message indicating its level and its name. So, name of course is the name of the fragment not its name the name of the fragment that it belongs to test node, test node in this case is q .

If of course such a q is not found, then it will send test node $\leftarrow \phi$ and report this back that look I did not find any. So, now there are two possible outcomes of find min. One is that you send a test message the other is that you report that look that I did not find any. So, let us see what happens to both. So, how did we reach here. The big picture is that look after fragments merged you cannot stop the process of merging.

So, you cannot stop the process of a fragment growing. So, after you are merged to fragments it is a job of every single node in the merged fragment to look for further expansion that is where we enter the find state. In the find state every node needs to do its part which means find all of its neighbor with a basic status, find the minimal weight neighbor in this set and see if a connection with it can be initiated by sending a test message. If it does not find of course it should report.

(Refer Slide Time: 38:59)

Algorithm 5: Receipt of test Message

```
1 Receive <test,level,name'> from q
2 if level' > level then
3   wait
4 end
5 else if name = name' then
6   /* Internal Edge
7   if status [q] = basic then
8     status [q] ← reject
9   end
10  if q ≠ testNode then
11    send <reject> to q
12  end
13  else (q = test Node)
14    findMin()
15  end
16 else
17  send <accept> to q
```

Handwritten annotations include: "wait" circled in red; "status [q] ← reject" circled in red with a note "(edge to a node in the same fragment)"; "q = test Node" circled in red with a note "q"; and a diagram showing node p and node q with an arrow from q to p.

Algorithm 4: findMin

```
1 findMin:
2 if ∃ q ∈ neigh(p): status [q] = basic, (w(pq) is minimal) then
3   testNode ← q
4   send <test,level,name'> to testNode
5 end
6 else
7   testNode ← ∅
8   report()
9 end
```

Handwritten annotations include: "status [q] = basic, (w(pq) is minimal)" circled in red with a note "rejected"; "send <test,level,name'> to testNode" circled in red; and a diagram showing node p and node q with an arrow from p to q and the word "reject" written below.

So, receiving a test message again same convention I am node p I receive a test message from node q. So, this p, q stuff can be quite confusing in fact the first time that I read it I thought it was pretty challenging, but that said and done now I have gotten used to it you also will be. So, this is basically that I am getting a test message I am p from q and level' and name' is q's name and level.

So, if q's level > my level then by the EQ rule and LT rule anyway you cannot combine it. So, we just wait I just keep the message in an internal buffer and I do not do anything I just sleep on it else if name = name' which means my name and q name is the same which basically means that a message has been send to another node of the same fragment.

Well, then if the status is basic what I do is I mark the status to be reject. So, this clearly cannot be an MST because you cannot have an edge to a node in the same fragment that is now allowed. So, that will create a cycle in a tree. So, we know for sure that look this cannot be a valid tree edge. So, then what do I do this has to be a reject we need to reject this edge because it is to an internal node so I just reject no problem.

Then let us say that if q is not equal to test node which means that as far as I am concerned I might have sent a test message to q and I would have set test node. We just look at this and clearing of the ink. So, whenever I sent a test message to some other node I set that node to the test node as the test node. So, if $q \neq \text{test node}$ which means that I have not sent a message to q saying that would you want to join me because the status of this edge was unexplored.

Then clearly a reject message needs to be sent to q telling you that look we cannot join because we are actually a part of the same fragment, but if let us say $q = \text{test node}$ which means that already a test message has been sent then I should call find min again and then move to some other node because clearly q is not the candidate and q will also mark this edge to me as a rejected edge because it would get my test message.

So, recall that when is the test node set? It is set when a test message is sent. So, the fact that q is a test node it basically means that a test message which is this case. So, this is not equal to and this is equal to. So, in this case that the fact that q is a test node basically means that a test message has been sent to q which means that over due course of time q will mark the edge to me as reject so I did not bother.

As far as we are concerned I know that q is a part of my fragment q knows that I am a part of his fragment. We mutually know each other if it is this case. If we mutually do not know that I should make you explicitly aware of the fact that look q you should not have sent me a test message in the first place because you and me are a part of the same fragment. So, we can never have a connecting edge between us.

Hence, I am rejecting this message. So, as I said regardless of how the message is sent either as a test message or as a reject q will ultimately get to know that it is a part of the same fragment as p which is myself and it will pretty much mark me as invalid. So, given the fact that q is now a rejected node what I need to do is I need to again call find min and if I again call find min what would happen is that in this case the status of q will not be basic anymore.

The status of I mean the other q, the q that we have been talking about in this slide again p and q is slightly confusing, but the good thing about the video is that the same slide can be seen over and over again to get the basic idea. So, in this case since the status[q] = basic this thing does not hold anymore because we just rejected it then some other node has to be picked out of this set.

So, we will have a new minimum again we do the same again we sent a new test message to the new minimum. If it happens to be in my fragment so which means that I am a part of this fragment I sent the message to q. If somehow q also happens to be a part of my fragment unbeknown to me either it would have sent me a test message. So, with that I will get to know that q is actually part of my fragment.

So, I will mark it as invalid or q will sent me a reject message and then I will mark it as invalid and again I will come to this point where I will start testing with some other neighbor of mine that satisfies this criteria and if I do not find the neighbor I will report this fact.

(Refer Slide Time: 45:16)

The screenshot shows a slide titled "Receipt of test Message" with the following algorithm code and handwritten annotations:

```

Algorithm 5: Receipt of test Message
1 Receive <test,level,name> from q
2 if level > level then
3   wait
4 end
5 else if name = name then
6   /* Internal Edge
7   if status[q] = basic then
8     status[q] ← reject
9   end
10  if q ≠ testNode then
11    send <reject> to q
12  end
13  else
14    findMin()
15  end
16 else (names are not the same)
17  send <accept> to q
18 end
  
```

Handwritten annotations in red ink include:

- A bracket on the right side of lines 5-14, with an arrow pointing to the text "reject q".
- A bracket on the right side of lines 10-14, with an arrow pointing to the text "find min".
- A note "(names are not the same)" written in red next to line 16.

So, the key point is that of course if this holds that q has a higher level I wait otherwise we are part of the same fragment then of course essentially what I do is I reject q the branch is rejected and I call find min again because I would like to explore some other neighbor of mine. Otherwise, if the names are unequal names are not the same if the names are not the same then what I do either I sent an accept message to q because there is no reason why I should not.

So, there is absolutely no reason why I should not and the thing is that number one there is no issue with the less than or equality and furthermore it is the least weight edge between the two fragments.

(Refer Slide Time: 46:35)

Receipt of **accept/reject** messages

Algorithm 6: Process accept/reject messages

```

1 Receive <accept> from q:
  testNode ← φ
  if  $w(pq) < bestWt$  then
2   bestWt ←  $w(pq)$ 
   bestNode ← q
3 end
4 report()
Receive <reject> from q:
  if status[q] = basic then
5   status[q] ← reject
6 end
7 findMin()

```

Srnul R. Sarangi | Assorted Algorithms | 16/25

So, after I receive an accept message from q which means that q does not have an objection I set so once let us say p gets an accept message from q which means that q does not have an objection then I set the test node to null and let us say if the $W(pq) < bestWt$ that I have seen then I set the $bestWt \leftarrow W(pq)$ and I set my best node $\leftarrow q$ and I report this fact.

So, what I report() is that look I have found something and as far as I am concerned my best neighbor = q and report mind you are not a message to another node it is just a function call. It is just an internal function call it is not a message and the internal function call we will be able to see these internal variables. If I receive and also what I do if I receive a reject from q, well then I change the status of the edge from basic to reject and I continue with other neighbors.

So, no problem what is the idea the idea is that look my state is find so my job is to find neighbor I start contacting my neighbors. In an ascending order first I look at only the basic edges not branch or reject edges and in an ascending order of weight I start contacting them either they can just hang on to my message and not reply to me. So, that is one option that they have the other is that they can either accept or reject. If they accept it I record this fact if they reject it I move on to some other neighbor of mine let us say this rejects I move on to another neighbor of mine.

(Refer Slide Time: 48:25)

Gallager Humblet Spira (GHS) Algorithm Distributed Snapshots Overview Algorithms Analysis

report Method

```
Algorithm 7: report Method
1 report:
  if (rec = {q; status [q] = branch ^ q ≠ parent }) ^ (testNode = φ) then
  2 state ← found
    send <report, bestWt> to parent
  3 end
```

Handwritten notes: # children, child, found, report, bestWt

Diagram: A node labeled 'Min' with three children.

NPTEL Smruti R. Sarangi Assorted Algorithms 17/25

Gallager Humblet Spira (GHS) Algorithm Distributed Snapshots Overview Algorithms Analysis

Receipt of accept/reject messages

```
Algorithm 6: Process accept/reject messages
1 Receive <accept> from q:
  testNode ← φ
  if w(pq) < bestWt then
  2 bestWt ← w(pq)
    bestNode ← q
  3 end
  4 report() → func. call

Receive <reject> from q:
  if status [q] = basic then
  5 status [q] ← reject
  6 end
  7 findMin()
```

Handwritten notes: φ, w(pq), bestWt, bestNode, q, report(), func. call, status [q], reject, findMin()

Diagram: A node labeled 'q' with three children, and a node labeled 'Min' with three children.

NPTEL Smruti R. Sarangi Assorted Algorithms 16/25

So, what does the report method do? It is a method it is not a message it is a method. So, what does the report method does is that number one it looks at the set, but this is the set of all q so again I am p and the other node is q any other node is q. So, I look at all my neighbors such that the status of the neighbor is branch and it is not a parent. So, it means it is a child. So, both of these things together it means that the other node q is a child because it is a neighbor of mine and it is not a parent.

So, it can only be a child and if rec is equal to this so basically this expression over here the cardinality of this set is essentially the number of children. So, if $(rec = | \{q: status[q] = branch \wedge q \neq parent\} |) \wedge$ the number of children which basically means that similar if you go back to the leader election algorithm in that we have discussed leader election in trees where all the children send their values to their parent and then it kind of propagates up the tree. So, this is basically saying that look if I have received a message from all my children and a (test node = ϕ).

So, when I do I set the test node to null? When I receive an accept message or when I find that none of my neighbors picked a criteria which basically means that no outstanding test message is there. So, let us just look at the test node there when I receive an accept I set it to a null and when else do I set it to null? I set it to null when I run out a neighbor so then also I set it to null otherwise I do not set it to null.

If I have sent a test message and it is outstanding, then it is non null. So, the fact that here I am saying that test node = ϕ which basically means I have finished my job of testing. So, what it means if you go back to the tree based leader election algorithm that was in the previous lecture

of the slide set. So, we had said that every parent essentially finds its own minimum and also aggregates all the minima sent by its children.

Once it is done all once of his children have sent a message which is precisely being captured by this complicated looking mathematical formula over here which just simply puts means that all my children have responded to me and test node = ϕ which again in simple layman terms means that I am done with my job. So, together the if statement means that my children are done with their job and I am done with my job I set the state to found.

And I report as an honest child to my parent that look I am sending you the report message this is the best weight edge that I found. The best weight edge means as far as I am concerned for my sub tree this is the least weight outgoing edge it is a valid least weight outgoing edge and furthermore the node on the other side agrees to connect with you. So, I have an upfront commitment from the other node that is now going to refuse.

(Refer Slide Time: 51:59)

The slide displays the following algorithm with handwritten annotations:

```
Algorithm 8: Process report Message
1 Receive <report( $\omega$ )> from q:
2 if  $q \neq \text{parent}$  then
3   if  $\omega < \text{bestWt}$  then
4      $\text{bestWt} \leftarrow \omega$ 
5      $\text{bestNode} \leftarrow q$ 
6   end
7    $\text{rec} \leftarrow \text{rec} + 1$ 
8    $\text{report}()$ 
9 end
10 else  $q = \text{parent}$ 
11   if  $\text{state} \leftarrow \text{find}$  then
12     wait
13   end
14   else if  $\omega > \text{bestWt}$  then
15      $\text{changeRoot}()$ 
16   end
17   else if  $\omega = \text{bestWt} = \infty$  then
18     end
19 stop
```

Handwritten annotations include red circles around ω , bestWt , bestNode , rec , $\text{report}()$, $q = \text{parent}$, $\text{changeRoot}()$, and end . A diagram shows a node q with children a and b , and a larger tree structure below.

Gallager Humblet Spira (GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

report Method

```

Algorithm 7: report Method
1 report:
  if (rec = |{q: status [q] = branch ∧ q ≠ parent }|) ∧ (testNode = φ) then
2   state ← found
   send <report, bestWt> to parent
3 end

```

NPTEL

Smruti R. Sarangi | Assorted Algorithms | 17/25

So, then how do I process the report message? So, when I get a report message from q so again I am p and I am always p when I get a report message from q in this case q is my child. So, then similar to again the last slide of the leader election algorithm with trees we said that a parent with his own parent and own child so something similar to this is happening over here.

So, if $q \neq \text{parent}$ in the sense if my child is not my parent which of course holds for the root node. If let us say $\text{omega} < \text{bestWt}$ that I have seen that the $\text{bestWt} \leftarrow \text{omega}$ and the best node $\leftarrow q$ it means that as far as I am concerned the child that is sending me the best weight is the best node. So, I will record this fact and furthermore given that my child is sending me a message I will just set $\text{rec} = \text{rec} + 1$ which means that one of my children is responding.

And then I will call the report function, the report function is the same as this which given the fact that we have understood this and so this will basically in this case we will check whether all my children have replied or not and given the fact that the rec variable I just incremented. So, I am assuming all of these variables are global within the scope of a node.

So, since I have just incremented this it is possible that the if condition becomes positive and I enter this. If I do not, then there is no problem I just come back and I just keep waiting. So, this part of the quote basically means that every parent waits for all of its children to report their best edges that they are finding. It computes the overall minimum report that to its parent.

So, this is as I said what would happen in any tree that every sub tree will report its best again the parent will collect everything from its children which again is a root of its own sub tree and it will just propagate that up, up and up and up no problem until you reach the root what is the

root? So, the root in this case is slightly complicated we will come to it. So, otherwise if $q =$ parent in a sense I am receiving a message from my parent which means my parent is sending me the report message.

It is kind of strange, but we will see when that happens. So, then we will see so in this case $q =$ parent. If the state is find, if my state is find I am still finding then I wait. If $\omega < \text{bestWt}$ which means that my best weight is actually the best then I change the rule. Otherwise, if let us say $\omega = \text{bestWt}$ which means me and my parent both are reporting infinity.

This means that we have actually reach the end there are no eligible edges. So, then the MST condition has been met and it is all done end terminal. So, now the structure of the parent is quite important and see if you go back to the connect message so then the parent thing is quite important, but I would like to discuss change root first before going to parent because they are connected.

(Refer Slide Time: 55:55)

The slide displays the following algorithm and diagrams:

```

Algorithm 9: changeRoot() Method
1 changeRoot():
  if status [bestNode] = branch then
2   send changeroot to bestNode
3 end
4 else
  /* Along the Core Edge */
5   status [bestNode] ← branch, send <connect,level> to
   bestNode
6 end
7 Receive changeroot:
  changeRoot()
  
```

Handwritten diagrams include:

- Raymond's tree:** A tree structure with a root node at the top, labeled "Raymond's tree". A node below the root is labeled F_1 . A node further down is labeled "Branch" and F_2 .
- Core Edge:** A diagram showing a node with a core edge connecting to another node, labeled "Core Edge".

The slide also features a navigation sidebar on the left, the NPTEL logo, and a footer with the text "Srnul R. Sarangi Assorted Algorithms 19/25".

Gallager Humblet Spira(GHS) Algorithm Distributed Snapshots Overview Algorithms Analysis


Receipt of report Message

Algorithm 8: Process report Message

```

1 Receive <report, $\omega$ > from  $q$ :
2 if  $q \neq \text{parent}$  then
3   if  $\omega < \text{bestWt}$  then
4      $\text{bestWt} \leftarrow \omega$ 
5      $\text{bestNode} \leftarrow q$ 
6   end
7    $\text{rec} \leftarrow \text{rec} + 1$ 
8   report()
9 end
10 else
11   if  $\text{state} \leftarrow \text{find}$  then
12     wait
13   end
14   else if  $\omega > \text{bestWt}$  then
15     changeRoot()
16   end
17   else if  $\omega = \text{bestWt} = \infty$  then
18     stop
19   end

```



NPTEL Smruti R. Sarangi Assorted Algorithms 18/25

Gallager Humblet Spira(GHS) Algorithm Distributed Snapshots Overview Algorithms Analysis



Process connect Message

Algorithm 2: Processing of the connect message

```

1 Receive <connect, $L$ > from  $q$ :
2 if  $L < \text{level}$  then
3   /* Combine with rule LT */
4    $\text{status}[q] \leftarrow \text{branch}$ 
5   send <initiate,level,name, state > to  $q$ 
6 end
7 else if  $\text{status}[q] = \text{basic}$  then
8   wait
9 end
10 else
11   /* Combine with rule EQ */
12   send <initiate,level+1,pq,find> to  $q$ 
13 end

```

NPTEL Smruti R. Sarangi Assorted Algorithms 12/25

Gallager Humblet Spira(GHS) Algorithm Distributed Snapshots Overview Algorithms Analysis


Receipt of initiate message

Algorithm 3: Processing of the initiate message

```

1 Receive <initiate,level,name,state> from  $q$ :
2 /* Set the state */
3 ( $\text{level}, \text{name}, \text{state}$ )  $\leftarrow$  ( $\text{level}, \text{name}, \text{state}$ )
4  $\text{parent} \leftarrow q$ 
5 /* Propagate the update */
6  $\text{bestNode} \leftarrow \phi$ 
7  $\text{bestWt} \leftarrow \infty$ 
8  $\text{testNode} \leftarrow \text{none}$ 
9 foreach  $r \in \text{neigh}(p): (\text{status}[r] = \text{branch}) \wedge (r \neq q)$  do
10   send <initiate,level,name,state> to  $r$ 
11 end
12 /* Find least weight edge */
13 if  $\text{state} = \text{find}$  then
14    $\text{rec} \leftarrow 0$ 
15   findMin()
16 end

```



NPTEL Smruti R. Sarangi Assorted Algorithms 13/25

So, let us now discuss the last algorithm which is the change root method. So, here what happens is that we have found out in the entire fragment which node is connected to the least weight edge. So, this has been found out why and how? Well, so basically every single node of the sub tree broadcasted its best values to the root and finally the root computed the minimum operation over here as you can see and updated the best weight and best node.

The best node is of course one of its child nodes that has a part to the eventual best node which lies at the edge of his fragment. So, every node just keeps a pointer to its child and just by passing these child pointers we ultimately reach the edge where you have the edge that is the least weight edge of this entire fragment. So, now when you decide to change your root which basically means that you are basically rooted at.

This node that is at the edge so what this essentially means is that if we consider all the nodes within this p regardless of how they were, so, this is of course similar to the Raymond's tree algorithm for mutual exclusion which was there in our lecture set. So, there what happens is we do a change, the change root essentially means that every node updates its parent point to point to the node table.

So, every node over here update its pointer along the path such that so if let us say this was the old root. So, all the nodes pointed to this and the old root over here point to the new root which is over here and this is of course the edge that is pointing to a different fragment this is F1 this is F2 and clearly why are these edges it is possible to reach this node for any node within the fragment because every node within the fragment in any case was pointing to the old root.

So, what will happen is that it will now what we are doing is we are establishing a path from the old root to the new root just by flipping child parent pointers. So, what we are doing is we start from the root we just see the status [best node] ← branch then what we do is we sent change root to best node and so we just keep on doing that ultimately what will happen is we will arrive over here.

Then what will happen is that the status of the best along the core edge what we will do is we will set the status [best node] ← branch in the sense that this point the status of edge e from basic will turn into branch which basically means that now I acknowledge the fact that e is the least weight outgoing edge out of the fragment.

And furthermore, you will send a connect message connect level to best node which means across the edge e to the other fragment. So, what did we do? The summary is that in the entire fragment each of the nodes looked at each of its children that are outside the fragment on undecided edges started from the minimum went up the ascending change if there were any rejects ultimately till the other side gave an acceptance that yes I am willing to join.

And then all of this information was for the work coagulated all the way up to the root and after that point the decision has made what genuinely is the best then subsequently another decision was sent back of course the parent point are slipped and I am not showing that in the slide, but then after that the parent pointers were slipped. So, let us say this edge over here was found to be the minimum.

So, then we set the status of this edge to the branch and then we send the connect message where the connect message will again take us back to algorithm 2.

There what was happening in algorithm 2? What was happening is that we were processing the connect message and there of course if you found the LT condition we connected immediately. So, here if you would recall we have kept something open. So, we had said that the LT condition is not holding so what could happen. If this is not holding it means $L \geq \text{level}$ which is fine.

This means that from the outside the level is coming which is greater than equal to my level. So, now what I do is that I look at a status of the edge. With the status of the edge is basic then I wait which means that so what does this mean? This means that I have not made any decision about this edge. So, even if I have sent a test message on this edge I have technically not made a decision because I have not gotten any accept or reject.

And so, I have not made any decision one sided, but let us see if it is not basic then I come here this is where we have left a question open. What we had said is that what happens here. So what happens here now we know. So, this basically means you reach over here number one if $L \geq \text{level}$. Number two with the status of the edge is either branch or reject.

So, let us look at the reject case first I think reject eliminate. So, if the status of this edge is reject then there is no reason why a connect message should have been again sent by the same edge because the connect message is contingent on the fact that an accept was received, but since a reject has been receive there is no chance that a connect message will be sent on the same edge because it means that both the nodes are a part of the same fragment.

So, reject is not possible. So, this means that the status of the edge has to be a branch which means that from the point of view of both fragments it is there least weight outgoing edge. Furthermore, L cannot be greater than level this is not possible for a simple reason that for any connect to happen it should have gotten an accept first and if I actually look at it if you just look at this line over here whenever a test message is sent in this case if $L > \text{level}$ then it just waits.

So, then what would have happened is that in this case if $L > \text{level}$ then no prior communication would have been initiated in the sense an accept message would not have been sent. This means this branch would not have been chosen and definitely a connect message would not have been sent along this branch. So, even $L > \text{level}$ will not happen.

So, the only choice that we are left is that the status is a branch and furthermore $L = \text{level}$ nothing else is possible. Given that nothing else is possible what we see is that the EQ rule holds in this case. We were not able to save the first time when we looked at this algorithm, but now we can clearly see that the equality rule holds, the status of the edge is a branch.

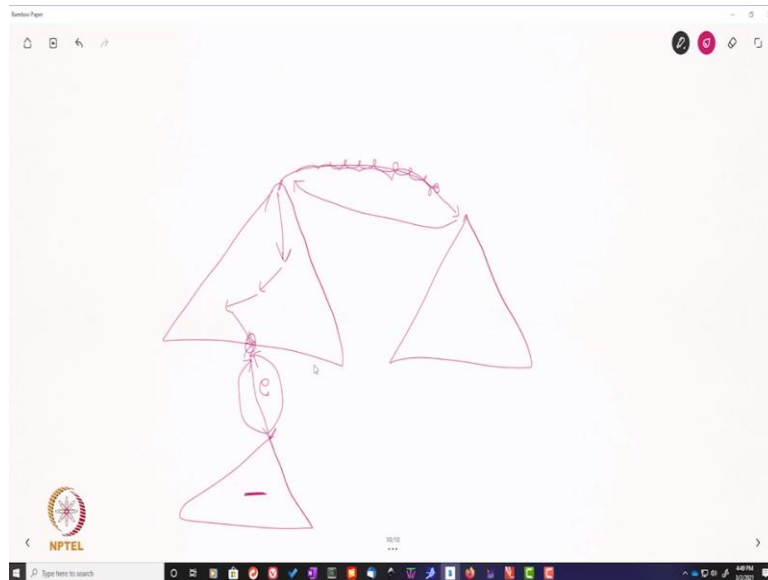
And furthermore the levels are equal because nothing else is possible and that is how we initiate. So, what would happen is for two fragments of the same level if we let us say take a look at their joining edge. So, let us call this node p and p' I do not want to use p and q anymore. So, what would have happened is that p would have sent a connect message to p' and got initiate back and p' would have done exactly the same given that it is a branch for both p p' would have sent a connect message to p and gotten initiate back.

And then what you see is in this thing that you sent the parent you set the parent to the other node. So, you would have a parent relationship around this edge which is also called the core edge that would look something like this. So, this is why I said that as far as all of these nodes are concerned they will direct their parent pointers up here, but here you have this cyclicity around this core edge.

So, we can think of this edge as a parent and the two nodes here pointing to each other in a special kind of manner and all the nodes within are pointing to basically the root and both the roots are connected to each other in this fashion. So, now what happens is that this becomes a bigger fragment, but let us say if this fragment now wants to joins to another fragment then what happens is that let us say it finds a least weight outgoing edge.

So, the least weight outgoing edge can be over here. In this case this cycle over here will break. So, this edge will go away and what will instead remain if I want to draw it in a slightly bigger canvas.

(Refer Slide Time: 1:06:26)



Gallager Humblet Spira (GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

changeRoot()

Algorithm 9: changeRoot() Method

```

1 changeRoot():
  if status [bestNode] = branch then
2   send changeroot to bestNode
3 end
4 else
  /* Along the Core Edge */
5   status [bestNode] ← branch send <connect,level> to
   bestNode
6 end
7 Receive changeroot:
  changeRoot()
  
```

Srinuli R. Sarangi | Assorted Algorithms | 19/25

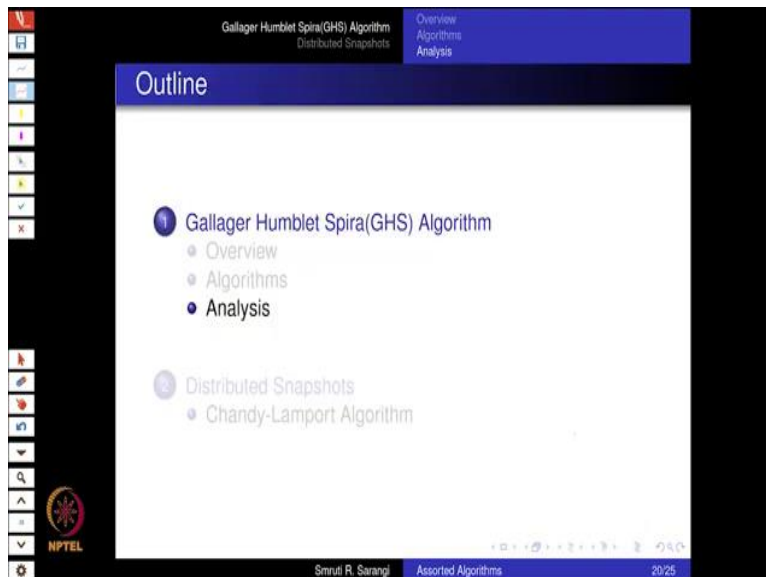
So, what would essentially remain is something like this if let us say these are the two fragments initially what happened is if this is the core edge this is how they were pointed towards each other and if let us say this is the least weight outgoing edge then this parent pointer will break down and then a sequence of parent pointers will be used to come here.

So, this basically means that now as you can see all the nodes in this fragment are pointing here as this is the new root, all the nodes of this fragment are also pointing here because they were pointing to the old roots so now they are pointing over here.

Similarly, if this is joining with another fragment depending upon the levels, depending upon what exactly is the level. You will have a connection that is made and so let us say that this has a lower level than this then of course this pointer like this and you will have a core edge somewhere within this fragment otherwise this edge over here will become the core edge and you will see such kind of a cyclic relationship.

So, this is kind of nice, interesting and elegant yet complex. So, what we have basically done now is that we have looked at these special cases and we have furthermore said that what happens at the end. So, what basically happens at the end is that we set the change root messages and finally a change root happens and so gradually that is the way that your fragments actually expand, your smaller fragments keep joining around core edges. And they keep growing, growing, growing, growing ultimately the entire graph becomes a single fragment.

(Refer Slide Time: 1:08:26)



Gallager Humblet Spira (GHS) Algorithm
Distributed Snapshots

Overview
Algorithms
Analysis

Time Complexity

Proposition 1
There are $O(N \log(N))$ fragment name or level changes.

Message Complexity
Message Complexity: $2E + 5N \log(N)$

- Every node is rejected only once \rightarrow one **test** message and one **reject** message
 - Total $2E$ messages
- At every level, a node sends/receives at most:
 - 1 receives: 1 initiate message
 - 1 receives: 1 accept message
 - 1 sends: 1 report message
 - 1 sends: 1 changeroot/connect message
 - 1 sends: 1 successful test message

5N

NPTEL

Srinuti R. Sarangi Assorted Algorithms 21/25

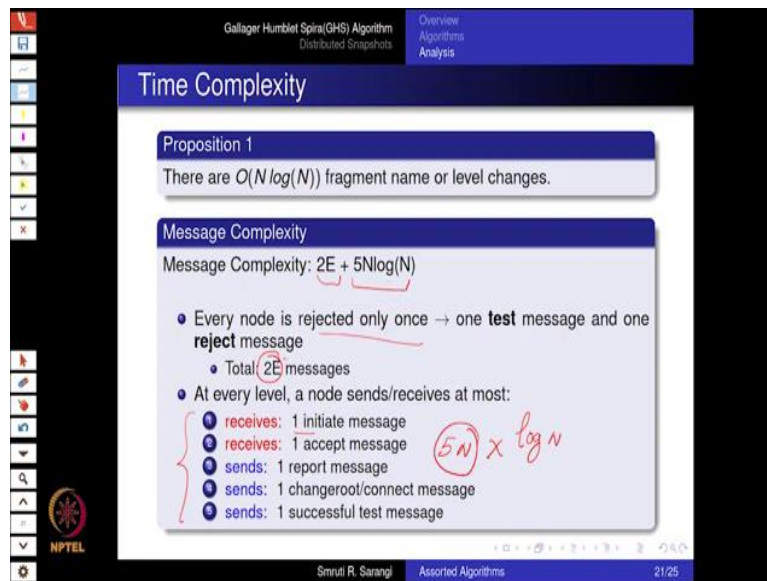
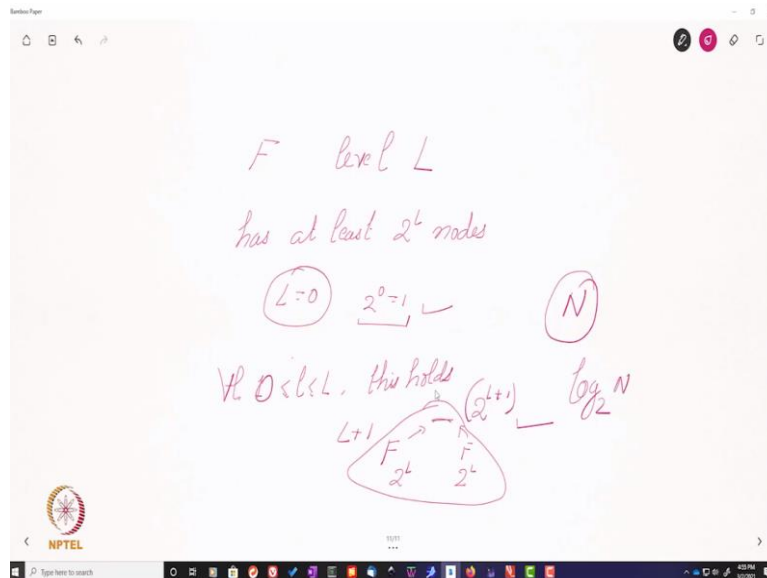
So, now a little bit of an analysis so we claim that there are $O(N \log(N))$ fragment name or level changes total. We further claim that the message complexity is $2E + 5N \log(N)$, E is the number of edges, N is the number of nodes. So, what is the logic well every node is rejected only once correct cannot be rejected more times. One test message and one reject message.

So, every node that is not a part of the tree that is rejected only once and that it is finished. So, this is limited to $2E$ messages fair enough for easy number of edges. At every level a node sends, receives at most these many messages, how many messages. One initiate message to start, one accept message, one report message. So, it receives these two. So, it receives and initiates message and an accept message.

So, initiate and accept is what is receives every node and it sent a report message a change root or connect message depending upon where it is in the tree and a successful test message. So, these are the five kinds of messages that it sends or receives. Furthermore, there is no intersection between the sending set and the receiving set. So, we can say that for every level these are the 5 messages that every nodes receives.

So, let us say there are N nodes then per level we have an exchange of $5N$ messages right here. You are welcome to verify this. So, this is just a question of simple book keeping that is all and so I have just one more thing to add so how many level changes will you have? So, what happens is that anytime a level changes anytime that a level changes we claim that the number of nodes it at least doubles. So, I would like to make a slightly tighter claim over here. So, let us go over here.

(Refer Slide Time: 1:10:40)



So, the claim that I am making is that any fragment with level L with let us say level L has at least 2^L nodes. So, this is trivially true if $L = 0$ it is trivially true because why $2^0 = 1$ and every node by itself has level 0 so this is trivially true. So, now let us consider a mathematical induction based proof. So, let us assume that till level L this holds.

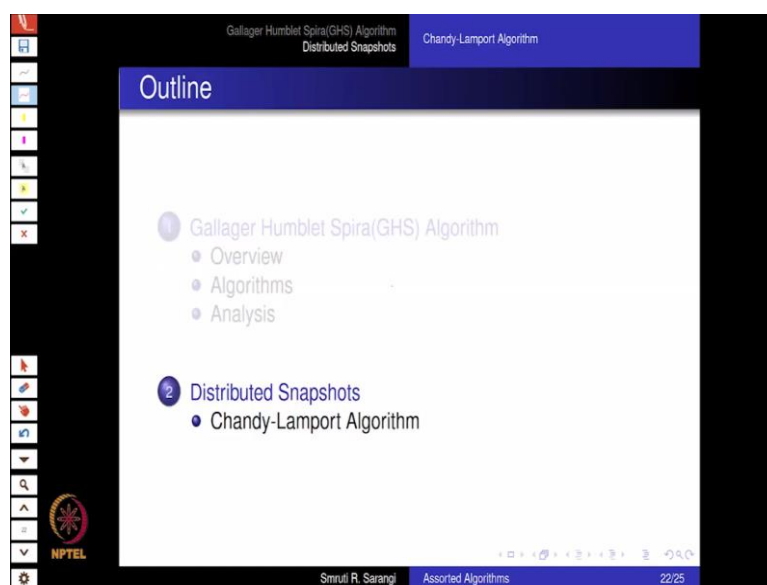
So, basically for all levels for all L or let us say 0 between 0 and L this holds. So, let us now consider level $L + 1$. So, how do you go to level $L + 1$? You go to level $L + 1$ only when two such fragments combine otherwise we remain at the same level and then only is smaller fragments come and keep on joining you which is fine. So, then this induction hypothesis will still hold. So, now if let us say two fragments of the same level are combining then this will have 2^L nodes this will have 2^L nodes.

So, total we will have $2^L + 1$ nodes. So, that is the total. So, again as we can see in the bigger fragment whose level is $L + 1$ the number of nodes that it has again at least 2^{L+1} . So, the induction hypothesis does hold. So, this means that the base case holds and the induction hypothesis holds. Hence by induction every time I increase the level the number of nodes at least double.

So, this further means that with N nodes the maximum number of levels that we are going to have is $\log_2 N$. So, with armed with this information so let us go back. So, given the fact that we will at best have $\log N$ level changes and per level change $5N$ messages are sent. So, the total number of messages are $5N \times \log N$. So, it is basically $2E + 5N \log N$ is the total number of messages that we are looking at.

So, it is essentially two times the number of edges + $5N \log N$ that the number of nodes so that is what we are looking at. So, in terms of complexity this is not that bad at all in the sense we are able to take a very large distributed network and create an MST with $N \log N$ message complexity which is quite good and in the space of distributed algorithms of course this algorithm is complex, but now I hope that most of it is well understood.

(Refer Slide Time: 1:13:52)



So, now we will discuss something called Chandy Lamport Algorithm in one or two slides. It is called a distributed snapshot. So, the idea is that fine I created a large distributed algorithm so what?

(Refer Slide Time: 1:14:06)

Gallager-Humblet-Spira (GHS) Algorithm Distributed Snapshots | Chandy-Lamport Algorithm

Overview

- Every process can take a local snapshot.
- The process does not process any message while taking a snapshot

Consistent Snapshot

If a message receive event is part of a local snapshot, then its send event should also be part of a snapshot.

- The channels are FIFO

NPTEL | Smruti R. Sarangi | Assorted Algorithms | 23/25

So, the idea is that in a large distributed system if let us say we have a large number of processes debugging this entire system is difficult because they are sending so many messages. So, let us say even if we give students a homework to implement the MST the minimum spanning tree algorithm. So, even then also debugging it is hard. So, what we do is that every process takes a local snapshot of its state.

So, snapshot is basically I want to capture a photograph of the entire system such that if anything is wrong with it I can analyze the snapshot and find out what is wrong, but even taking a photograph of a distributed system where there is clock synchrony is hard. So, we are looking at one way of doing it. So, as I said the algorithm is that every process takes a local snapshot.

Furthermore, the process does not process any message so of course these are different processes it does not act on any message while taking a snapshot. So, what we want is we want a consistent snapshot in the entire distributed system such that we can act on it. So, what this basically means is that if there is a sender and there is a receiver and let us say the sender sends a message.

It should never be the case that the receiver is taking a snapshot of its state where it is recording the message received, but in the sender's snapshot the message send is not there that should never be the case. So, if let us say the receive event is there the send event should be there that is the only requirement for consistency. There is no other requirement pursue. So, if let us say that in a distributed system we were to take such kind of a snapshot.

It would at least give us some kind of a photograph which of course is not instantaneous, but might give us enough information to debug and find the source of a problem. So, let us do that so we will use the Chandy Lamport Algorithm which is very simple. The only assumption it makes is that we have FIFO channels FIFO is first in first out channel in the sense A sends a message to B the messages are not re-audit.

(Refer Slide Time: 1:16:24)

Algorithm 10: Chandy Lamport Algorithm

```

1 initialize:
  take local snapshot
  taken ← true
  foreach q ∈ neigh(p) do
2   send <mkr> to q
3 end
4 Receive <mkr> :
  if taken = false then
5   take local snapshot
   taken ← true
   foreach q ∈ neigh(p) do
6     send <mkr> to q
7   end

```

So, the algorithm is like this that we take a local snapshot set taken to true. For each of our neighbors we send a marker to each of the neighbors. So, then what does that happen when a marker is received if taken is equal to false if a snapshot has not been taken then the neighbor takes a local snapshot and it sets taken to true. Again for each of its neighbors it sends the marker. So, one thing that is clear is that let us say if I have a system like this.

So, let us say I take a snapshot I send a marker to these three nodes, each of these nodes then take a snapshot and then they send a marker let us say the marker is send here, here and here, but let us say this marker finds that actually this node has taken a snapshot so the marker is ignored. So, you take a snapshot only once when you get the marker for the first time and after that after the snapshot marker is done you do not do anything.

And you just record the state and that is it I mean either you can stop there or you can wait for another message to ask you to resume. So, that is a separate matter we will get into that slightly later.

(Refer Slide Time: 1:18:04)

Gallager Humblet Spira (GHS) Algorithm
Distributed Snapshots

Chandy-Lamport Algorithm

Analysis

$[a \rightarrow b]$ if a-receive has been logged \Rightarrow send has also been logged
 $\sim b \rightarrow \sim a$

Theorem 1
The algorithm terminates in finite time.

Theorem 2
If a message $(p \rightarrow q)$ is sent after a local snapshot, then it is not a part of the receiver's (q) snapshot.

Consistency

NPTEL

Smriti R. Sarangi Assorted Algorithms 25/25

So, the theorem 1 is the algorithm terminates in finite time why not because you are sending messages ultimately the message will reach everybody and since every node takes a snapshot when it receives the first marker message it will terminate. The most important is theorem 2 that is regardless of the distributed system if I do this what I am saying is that the entire snapshot is consistent.

What does the entire snapshot consist of? The entire snapshot consists of the individual snapshots of all the nodes whatever has been taken that is what the entire snapshot consists of and why do I say it is consistent I claim that if a receive has been logged in the snapshots its corresponding send has also been logged that is my only requirement no other requirement. So, the theorem is that if I message from p to q is sent after a local snapshot.

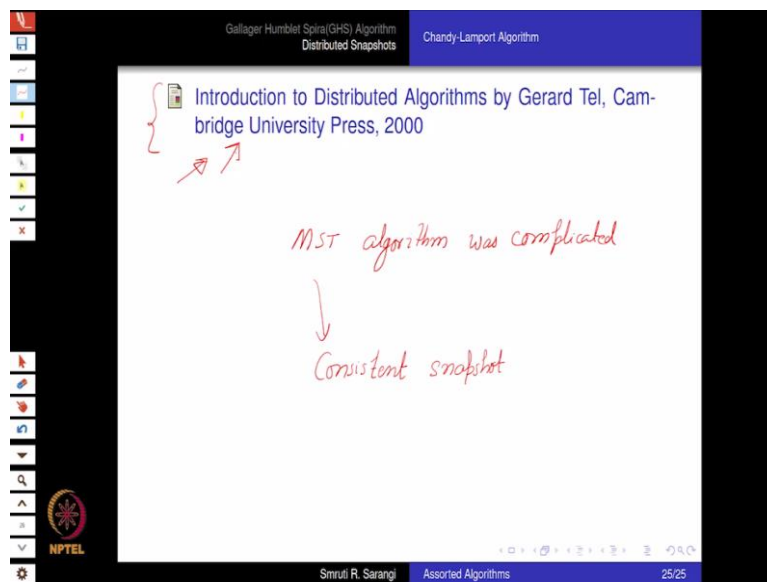
Then it is not a part of the receivers snapshot. So, what I am saying is that let us say there is node p and there is q. So, let us say it takes a local snapshot and then it sends a message. So, in this case we do not stall after taking a snapshot, but we send a message. So, then what I claim is that so in this case the send has not been logged because I take a snapshot first and then I send so I did not logged the send.

So, what I claim is that at the end of the receiver the receive will also not be logged and the answer is very simple, the proof is very, very simple. The proof is that when I take a snapshot I immediately sent a marker message to q. After the marker message I sent my other message this means that q either gets the marker from me or from somebody else before me and takes a local snapshot and only after that does it get the message m by that time q has taken its snapshot.

And it has not recorded the receive consequently this is correct and consistent as per our definition. Given the fact that we did not log a send, we did not log the receive also this was our simple definition of consistency and as you can see this simple algorithm does provide our definition of consistency what is it? If a receive has been logged in the snapshot that is that implies the send has also been logged.

So, what we were actually able to prove was the contra positive of this that if a send has not been logged then the receive has also not been logged which is what we were just able to prove over here and this is the same as this. So, anything contra positive is basically a implies b is essentially the same as not being implies not a. So, this is essentially proved by contra positive. So, this is what we have done and so we have one logarithm of at least recording a consistent snapshot in a distributed system. Why are we introducing this here?

(Refer Slide Time: 1:21:35)



Well, the reason we are introducing this is for a simple reason that the MST algorithm was complicated no doubt. Given that the MST algorithm was complicated when we actually coded on a distributed system you will have many cornered cases and the core is not going to work and then this entire p, q business is going to confuse you. Furthermore, what happens so I will just show you one of the slides which makes our life tough actually.

So, it is essentially slides like these. So, for example, this so where we wait the moment we have a wait if the code is not written correctly you might be waiting forever it might be infinite wait and so these wait messages are essentially what kind of make us quite jittery. So, these are things that we do not like. See here also there is one more wait, so we do not like these things.

So, given the fact that we do not like these things what will happen is that in most cases the code will not complete because the processes will just end up waiting because of some bug somewhere. So, to debug such systems and find out what exactly has gone wrong we can create a nice summary of all the actions that a given node has taken. We will call it the snapshot of the node and use the Chandy Lamport Algorithm to record a consistent snapshot, consistent as per our definition.

And then this snapshot can be written to maybe the disc by all the processes then subsequently this can be analyzed either manually or via script to find out what was the most likely cause of the error that is the reason why this lecture combines a complicated algorithm with a very short and small and cute algorithm to effectively debug a distributed system because debugging a distributed system is hard.

Now coming to the references the book by Gerard Tel has many of these details and so there are many other distributed algorithms as well in this book including a lot of concepts, so you are most welcome to read it and also implement the MST algorithm to get a practical field.