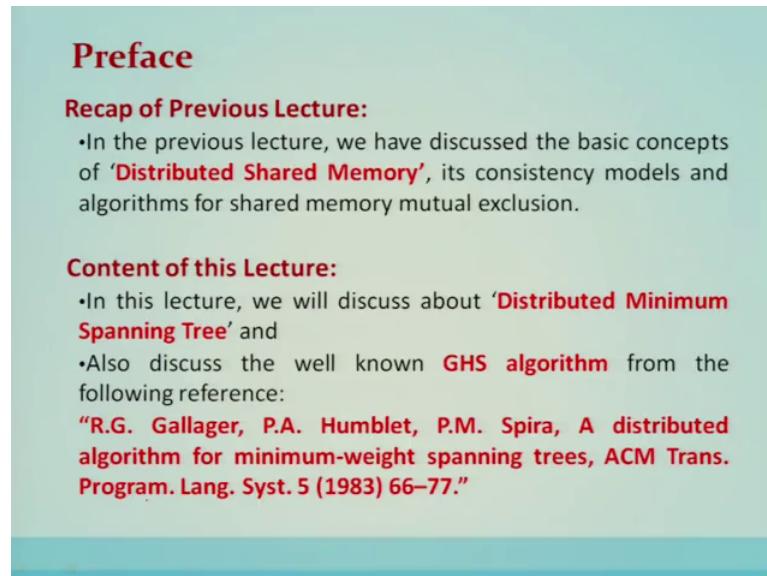


Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 14
Distributed Minimum Spanning Tree

(Refer Slide Time: 00:15)



Preface

Recap of Previous Lecture:

- In the previous lecture, we have discussed the basic concepts of '**Distributed Shared Memory**', its consistency models and algorithms for shared memory mutual exclusion.

Content of this Lecture:

- In this lecture, we will discuss about '**Distributed Minimum Spanning Tree**' and
- Also discuss the well known **GHS algorithm** from the following reference:
"**R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans. Program. Lang. Syst. 5 (1983) 66–77.**"

In previous lecture we have discussed the basic concepts of distributed shared memory, its consistency models and algorithm for shared memory mutual exclusion.

Content of this lecture in this lecture we will discuss about distributed minimum spanning tree and also discuss the well known GHS algorithm from the following reference. Gallager, Humblet, Spira, a distributed algorithm for minimum weight spanning trees in ACM transactions programming language in 1983.

(Refer Slide Time: 00:52)

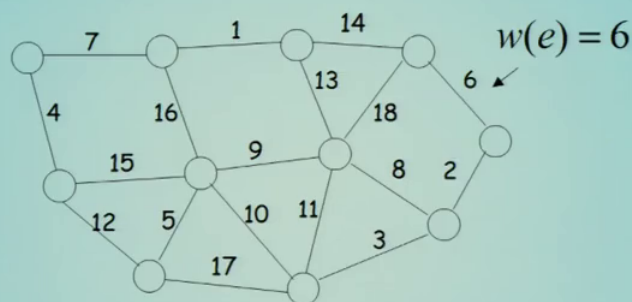
Introduction

- The **'Distributed Minimum Spanning Tree (MST)'** problem involves the construction of a minimum spanning tree by a distributed algorithm, in a network where nodes communicate by message passing.
- One important application of this problem is to **find a tree that can be used for broadcasting.**
- In particular, if the cost for a message to pass through an edge in a graph is significant, a MST can minimize the total cost for a source process to communicate with all the other processes in the network.

Introduction, the distributed minimum spanning tree MST problem involves the construction of a spanning tree by a distributed algorithm of a minimum weight, in a network they are the nodes communicate by passing messages. One important application of this problem is to find a tree that can be used for broadcasting. In particular, the cost of a message to pass through an edge in a graph is significant and MST can minimize the total cost of for a source process to communicate with all other processes in the network.

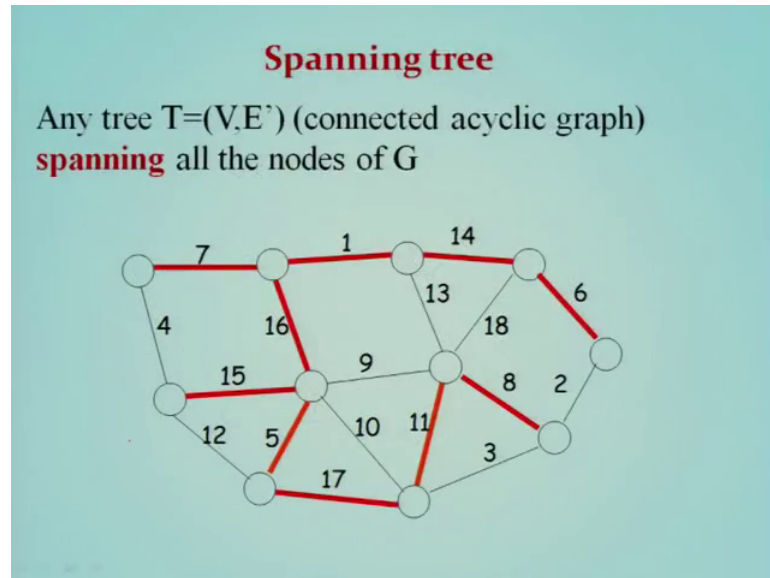
(Refer Slide Time: 01:36)

Weighted Graph $G=(V,E)$, $|V|=n$, $|E|=m$



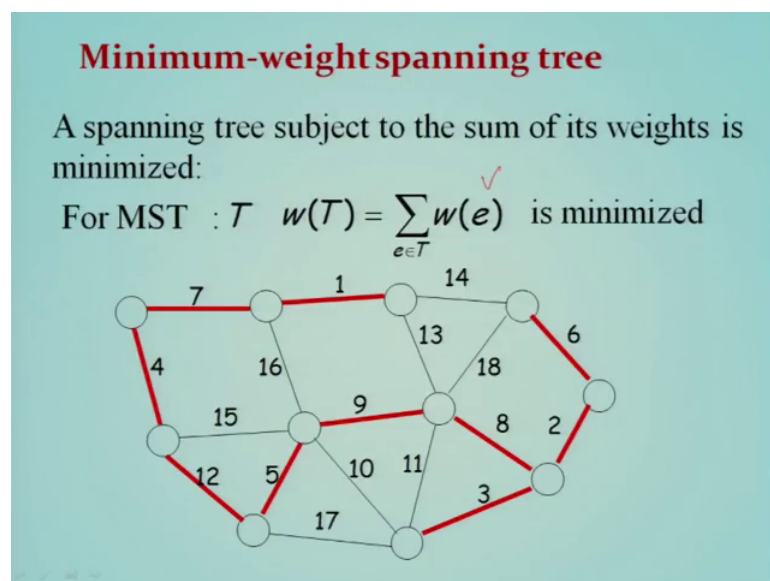
Preliminaries, weighted graph this algorithm requires a weighted graph of n number of vertices and m number of edges where the weights are given to the edges or non negative weights and also the weights are the distinct.

(Refer Slide Time: 02:00)



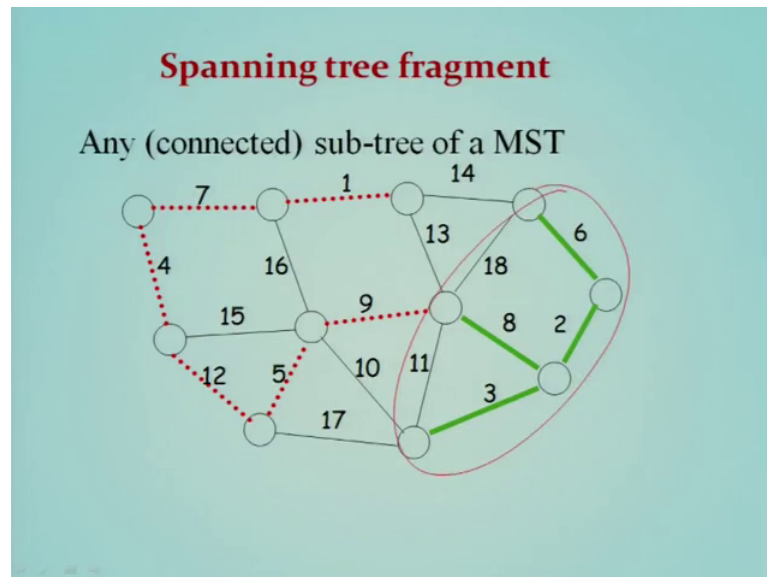
So, the spanning tree is a tree induced in a graph which is connected acyclic graph spanning all the vertices of G. So, we have shown you that the red colored edges forms our tree out of the given graph which we have shown you in the previous slide.

(Refer Slide Time: 02:25)



Now, that spanning tree which is subject to the sum of its weight is minimized is called minimum spanning tree. So, for minimum spanning tree the weight of a tree is nothing, but sum of all the edges of that particular tree and it should be the minimum of all possible spanning trees of the minimum weight is called minimum weight spanning tree.

(Refer Slide Time: 02:55)

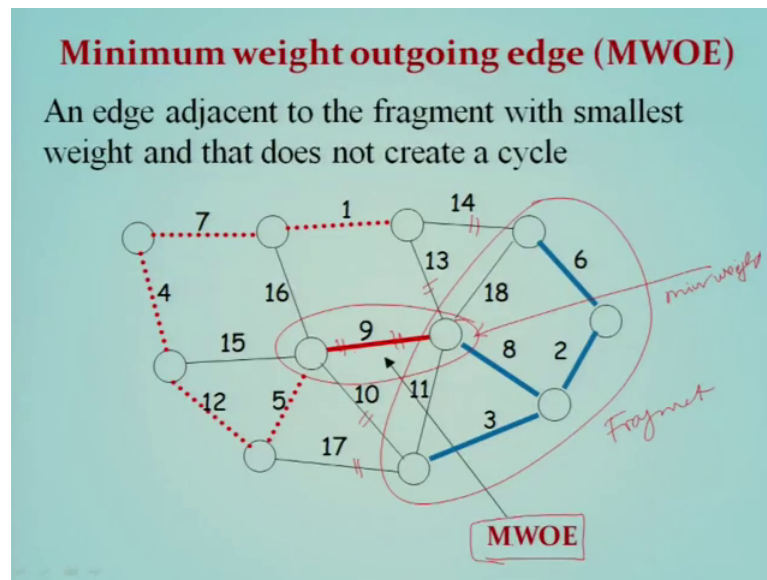


A spanning tree formation or a construction algorithms in the classical system is well known in the form of Prim's algorithm and Kruskal's algorithm. But if the system model is the distributed system where the nodes are communicating through the messages and the messages are transmitted with unpredictable delay in a finite predictable time.

So, in that particular model construction of a minimum spanning tree is going to be a challenging task. So, here we are going to see the algorithm of constructing the minimum spanning tree. So, minimum spanning tree means minimum weighted spanning tree in a distributed system we are going to see. Now we are going to see some of the terminologies which we are going to use in this particular lecture the first of them is a spanning tree fragment. So, any connected sub tree of a minimum spanning tree is called the spanning tree fragment.

So, here you can see here that the sub tree shown as the green edges is called a basically fragment spanning tree fragment.

(Refer Slide Time: 04:27)



Now, the spanning tree fragment will have different edges which are out of that particular tree and connecting to some other fragments are the nodes. So, an edge adjacent to the fragment with the smallest weight that does not create a cycle is called minimum weight outgoing edge. Take for example, that we have seen this as a fragment, and out of this particular fragment we can see that these are the edges which are out of this particular fragment. Among these edges we can see the edge which is given as the red colored edge is of the minimum weight.

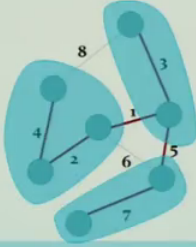
So, hence the definition says that minimum weight outgoing edge. So, the edge nine becomes the minimum weight outgoing edge and which is popularly known as MWOE in this particular algorithm.

(Refer Slide Time: 05:47)

MST Fragment

- Definition – MST *fragment*
 - A connected sub-tree of MST

Example of possible fragments:



Now, MST fragment a connected sub tree of the minimum spanning tree, so you can see in this example there are different fragments are possible.

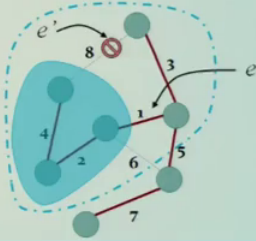
(Refer Slide Time: 05:56)

MST Properties

- MST Property 1

Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.
- *Proof:*

Suppose e is not in MST, but some e' instead.
MST with e forms a cycle.
We obtain a cheaper MST with e instead of e' .



So, minimum spanning tree properties, the first property MST property 1 says that given a fragment of an MST let e be a minimum weight outgoing edge of a fragment then joining e and its adjacent non fragment node to a fragment yields another fragment of an minimum spanning tree. Take this particular example that this is a fragment with one minimum weight outgoing edge which is shown as e . So, here we can see out of this

animation that when then joining e and its adjacent, adjacent non fragment node to the fragment yields another fragment of a MST which is shown over here.

(Refer Slide Time: 06:52)

MST Properties

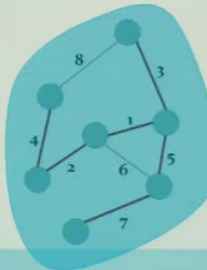
- **MST Property 2**
If all the edges of a connected graph have different weights, then the MST is unique.
- **Proof:**
*Suppose existence of two MSTs.
 Let e be the minimal-weight edge not in both MSTs (wlog e in T).
 T' with e has a cycle
 At least cycle edge e' is not in T .
 Since $w(e) < w(e')$ we conclude that T' with e and without e' is a smaller MST than T' .*

Now, MST property 2 says that if all the edges of a connected graph have different weights then the MST is unique. So, here in this animation we can see that in this particular tree let us take an edge e prime and in this particular spanning tree on that particular graph another spanning tree that is T with another e and in that edge is e . Now we can see that let e be the minimal weight edge not in both MST without loss of generality e is in T . Now, at least with e as a cycle. Now, at least cycle edge e prime is not in T since weight of e is less than weight of e prime we conclude that T prime with e and without e prime is a smaller MST then T prime. So, basically this particular animation proof shows that if all the edges of the connected graph have different weights then MST is a unique.

(Refer Slide Time: 08:06)

MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node (property 2)



The diagram shows a graph with 7 nodes and 8 weighted edges. The edges are labeled 1 through 8. The graph is partially highlighted in a light blue oval, showing a fragment of the tree structure. The nodes are arranged in a roughly circular pattern, with edges connecting them. The weights of the edges are: 1 (between nodes 2 and 3), 2 (between nodes 1 and 2), 3 (between nodes 3 and 4), 4 (between nodes 1 and 4), 5 (between nodes 3 and 5), 6 (between nodes 2 and 5), 7 (between nodes 5 and 6), and 8 (between nodes 1 and 4).

So, with these two properties the idea of MST based on these two properties we can see that we have to start with the fragment of a singleton node as shown over here.

Now, the next step is to enlarge the fragment in any order by using property 1 and combine fragments with a common node. So, that is property number 2. So, in this process; that means, applying iteratively property 1 and 2 we will get the complete coverage and will get the MST; that means, the fragment is continuously going using property 1 and 2 till it covers as one fragment and that is a minimum weighted spanning tree.

(Refer Slide Time: 09:09)

MST in Message Passing Model

- The **message-passing model** is one of the most commonly used models in distributed computing. In this model, **each process is modeled as a node** of a graph. The **communication channel between two processes is an edge** of the graph.
- Two commonly used algorithms for the **classical minimum spanning tree problem are Prim's algorithm and kruskal's algorithm**. However, it is difficult to apply these two algorithms in the distributed message-passing model. The **main challenges** are:
 - i. Both Prim's algorithm and Kruskal's algorithm require **processing one node or vertex at a time**, making it difficult to make them run in parallel.
 - ii. Both Prim's algorithm and Kruskal's algorithm require **processes to know the state of the whole graph**, which is very difficult to discover in the message-passing model.
- Due to these difficulties, new techniques were needed for **distributed MST algorithms** in the message-passing model.

So, minimum spanning tree in a message passing model the message passing model is one of the most commonly used model in the distributed computing in this model each process modeled as a node in a graph the communication channel between two processes is an edge of the graph.

Two commonly used algorithms for classical minimum spanning tree problem are Prim's algorithm and Kruskal's algorithm; however, it is difficult to apply these two algorithms in a distributed message passing model.

The main challenges are both Prim's algorithm and Kruskal's algorithm require processing one node or a vertex at a time, making it difficult to make them run in parallel. Both Prim's algorithm and Kruskal's algorithm require processes to know the state of the whole graph which is very difficult to discover in the message passing model due to these difficulties new techniques were needed for designing the distributed algorithms for minimum spanning tree in this problem setting that is in the minimum in the message passing model.

(Refer Slide Time: 10:17)

GHS Algorithm

- The **GHS algorithm of 'Gallager, Humblet and Spira' (1983)** is one of the best-known algorithms in distributed computing theory.
- GHS is a distributed algorithm based on Kruskal's algorithm** that constructs the minimum-weight spanning tree in a **connected undirected graph with distinct edge weights**.
- A processor exists at each node of the graph, knowing initially only the weights of the adjacent edges. The processors obey the same algorithm and exchange messages with neighbors until the tree is constructed.
- This algorithm can construct the **minimum spanning tree in asynchronous Message-passing model**.

GHS algorithm, GHS algorithm of Gallager Humblet and Spira 1983 is one of the best known algorithms in distributed computing theory. GHS is a distributed algorithm based on Kruskal's algorithm that constructs the minimum weight spanning tree in a connected undirected graph with distinct edge weights. A processor exists at each node of a graph knowing initially only the weights of the adjacent edges the processor will be the same algorithm and exchange messages with the neighbors until tree is constructed. This algorithm can construct the minimum spanning tree in asynchronous message passing model.

(Refer Slide Time: 10:58)

GHS Algorithm: Synchronous vs. Asynchronous

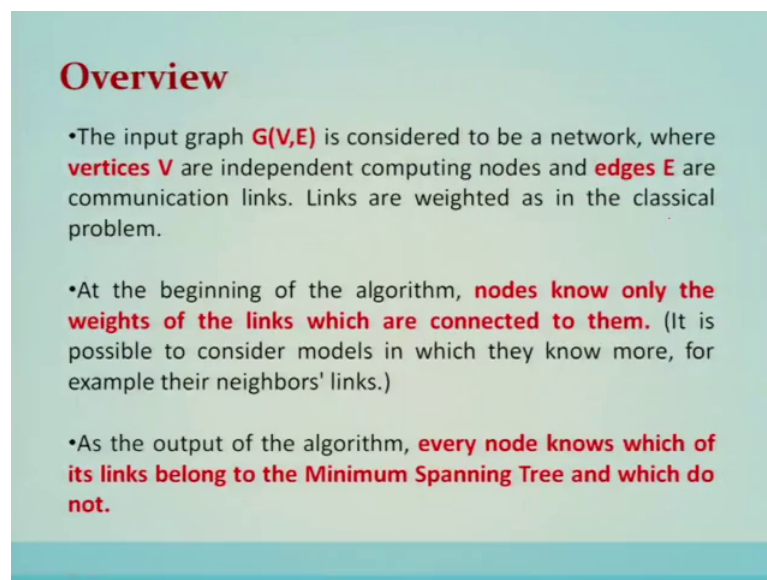
<p>1) Synchronous GHS:</p> <ul style="list-style-type: none">• Works with non-uniform models, distinct weights <p>Steps:</p> <ol style="list-style-type: none">1. Initially, each node is a fragment2. Repeat in parallel: (Synchronous Phase)<ol style="list-style-type: none">i. Each fragment – coordinated by a fragment root node - finds its Minimum Weight Outgoing Edge (MWOE)ii. Merge fragments adjacent to MWOE's3. Until there is only one fragment left	<p>2) Asynchronous GHS:</p> <ul style="list-style-type: none">• Simulates the synchronous version• Works with uniform models, distinct weights <p>Steps:</p> <ol style="list-style-type: none">1. Every fragment F has a level $L(F) \geq 0$: at the beginning, each node is a fragment of level 02. Two types of merges: absorption and join
--	--

Note: We will work with the 'Asynchronous GHS Algorithm'

GHS algorithm can run in synchronous as well as in asynchronous mode of communication and computation, in synchronous GHS it works with non uniform model with a distinct weight steps first initially each node is a fragment repeat in parallel synchronous phase each fragment coordinated by the fragment root node finds its minimum weight outgoing edge merge the fragment adjacent to minimum weight outgoing edge until there is only one fragment.

In asynchronous GHS it simulates the synchronous version works with both uniform and distinct weights uniform models distinct weights. So, steps in asynchronous GHS every fragment f has the level which is greater than 0 or equal to 0 at the beginning each node is a fragment of level 0, two types of merges absorption and join.

(Refer Slide Time: 12:07)



Overview

- The input graph $G(V,E)$ is considered to be a network, where **vertices V** are independent computing nodes and **edges E** are communication links. Links are weighted as in the classical problem.
- At the beginning of the algorithm, **nodes know only the weights of the links which are connected to them.** (It is possible to consider models in which they know more, for example their neighbors' links.)
- As the output of the algorithm, **every node knows which of its links belong to the Minimum Spanning Tree and which do not.**

Overview the input graph is considered to be the network and links are basically given weights is as in the classical problem. Edges represents the communication links at the beginning of the algorithm nodes know only the weights of the links which are connected to them. As the output of the algorithm every nodes knows which of its links belongs to the minimum spanning tree and which do not.

(Refer Slide Time: 12:37)

Preconditions

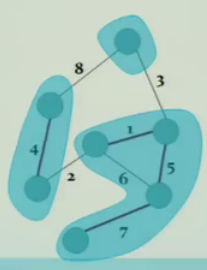
1. The algorithm should run on a **connected undirected graph**.
2. The graph should have **distinct finite weights** assigned to each edge.
3. Each node initially knows the weight for each edge incident to that node.
4. Initially, each node is in a quiescent state and it either spontaneously awakens or is awakened by receipt of any message from another node.
5. Messages can be transmitted independently in both directions on an edge and arrive after an unpredictable but finite delay, without error.
6. Each edge delivers messages in **FIFO order**.

Preconditions the algorithm should run on a connected undirected graph the algorithm should have distinct finite weights assigned to each edge, each node initially knows the weight of weight for each edge incident to that node. Initially each node is in a quiescent state and it is either spontaneously awakens or awakened by receipt of any message from another node. Messages can be transmitted independently in both the directions on an edge and arrive after an unpredictable, but finite delay without error. Each edge delivers message in FIFO order.

(Refer Slide Time: 13:20)

GHS Notation: Fragments

- Fragments
 - Every node starts as a single fragment.

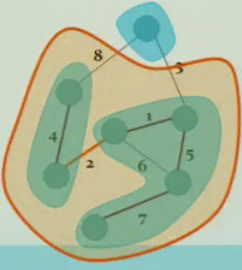


The idea of distributed MST, GHS notations first fragment is every node start with a single fragment.

(Refer Slide Time: 13:25)

GHS Notation: Fragments

- Fragments
 - Each fragment finds its minimum outgoing edge.
 - Then it tries to combine with the adjacent fragment.



(Refer Slide Time: 13:32)

GHS Notation: Levels

- Levels
 - Every fragment has an associated level that
 - has impact on combining fragments.
 - A fragment with a single node is defined to
 - to be at level 0.

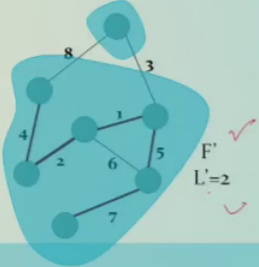
Each fragment finds its minimum outgoing edge then it tries to combine with the adjacent fragment every fragment has an associated level that. That has impact on combining fragments a fragment with a single node is defined to be a level 0. The combination of two fragments depend on the level of fragments.

(Refer Slide Time: 13:42)

GHS Notation: Levels

- Levels
 - The combination of two fragments depends on
 - the levels of fragments.

*If a fragment F wishes to connect to a fragment F' and $L < L'$ then:
 F is absorbed in F' and the resulting fragment is at level L' .*



The diagram shows a network of nodes and edges. Node 1 is the central node. Edges connect nodes 1-2, 1-3, 1-4, 1-5, 1-6, 1-7, and 1-8. Node 8 is highlighted with a blue circle and labeled 'F'. Node 3 is highlighted with a blue circle and labeled 'F'' with a checkmark. The text 'L'=2' is written next to node 3. The entire network is enclosed in a larger blue shape representing the resulting fragment.

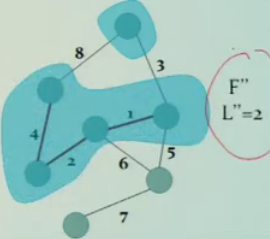
So, if the fragment F wishes to connect to a fragment F' this is F and this is F' and the level of fragment F is L is less than the level of F' then F is absorbed in F' and the resulting fragment is at the level L' that is shown over here. So, the resulting fragment will be F' and its level will be the same as F' 's level that is L' is equal to 2.

(Refer Slide Time: 14:28)

GHS Notation: Levels

- Levels
 - The combination of two fragments depends on
 - the levels of fragments.

*If fragments F and F' have the same minimum outgoing edge and $L = L'$ then:
The fragments combine into a new fragment F'' at level $L'' = L + 1$.*



The diagram shows a network of nodes and edges. Node 1 is the central node. Edges connect nodes 1-2, 1-3, 1-4, 1-5, 1-6, 1-7, and 1-8. Node 8 is highlighted with a blue circle and labeled 'F'. Node 3 is highlighted with a blue circle and labeled 'F'' with a checkmark. The text 'L''=2' is written next to node 3. The entire network is enclosed in a larger blue shape representing the resulting fragment.

Now, if the fragments F and F' have the same minimum outgoing edge and L is equal to L' that is their levels are same. Then the fragments combine into a new

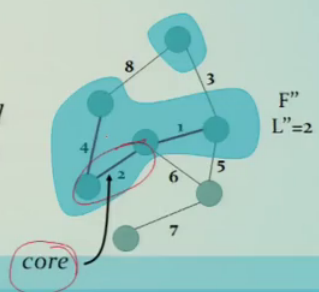
fragment F double prime and the level of this new fragment will be plus 1 of the earlier fragments which were seen as L . So, it will become L plus 1.

(Refer Slide Time: 15:07)

GHS Notation: Levels

- Levels
 - The identity of a fragment is the weight of its *core*.

If fragments F and F' with same level were combined, the combining edge is called the core of the new segment.



Now if the fragments f and f' with the same level were combined then combined edge is called the core of the new segment.

So, here you can see that the core this is the core why because this was instrumental in combining the two fragments F and F' , it becomes the core of the new segment.

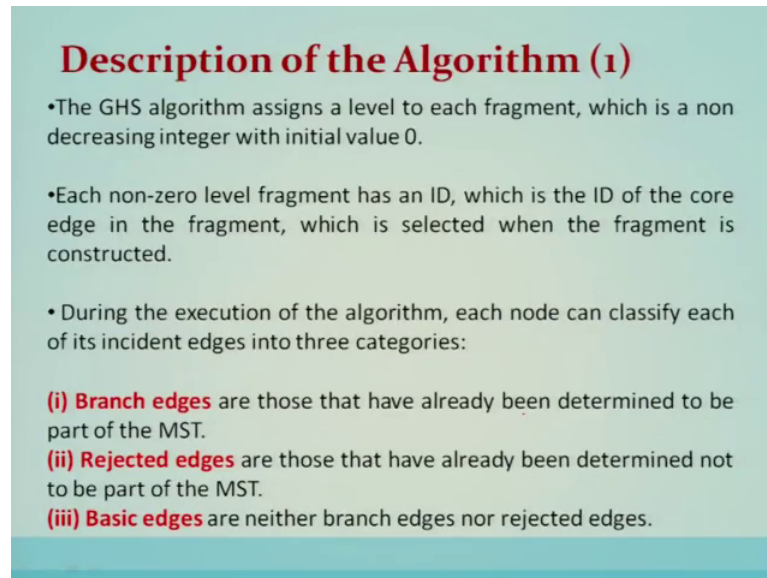
(Refer Slide Time: 15:29)

GHS Notation: Node States

- State
 - Each node has a state
 - Sleeping* - initial state
 - Find* - during fragment's search for a minimal outgoing edge
 - Found* - otherwise (when a minimal outgoing edge was found)

Then in GHS notation next thing is about node states, state each node has a state there are three states in which the node will be at a particular instant of time that is the sleeping state is initial state find state means during the fragment search for a minimal outgoing edge found means otherwise when a minimal outgoing edge was found. So, sleeping find found. So, there are three different node states.

(Refer Slide Time: 16:06)



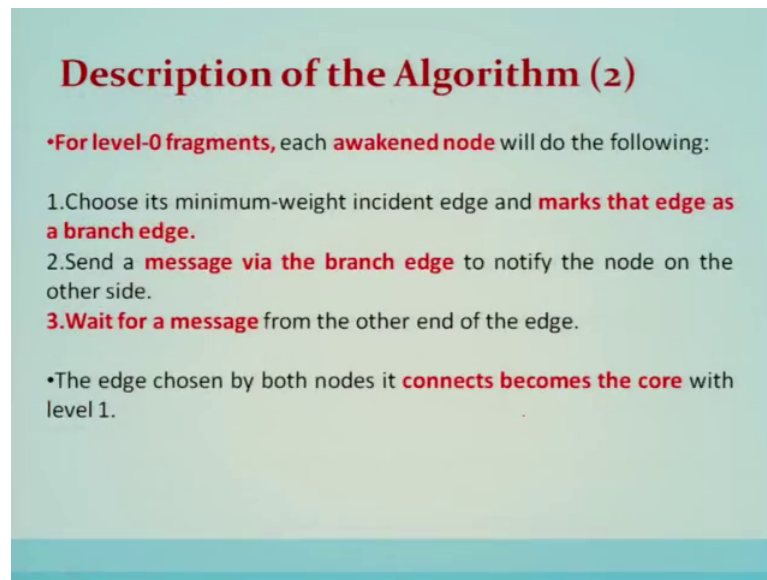
Description of the Algorithm (1)

- The GHS algorithm assigns a level to each fragment, which is a non decreasing integer with initial value 0.
- Each non-zero level fragment has an ID, which is the ID of the core edge in the fragment, which is selected when the fragment is constructed.
- During the execution of the algorithm, each node can classify each of its incident edges into three categories:
 - (i) **Branch edges** are those that have already been determined to be part of the MST.
 - (ii) **Rejected edges** are those that have already been determined not to be part of the MST.
 - (iii) **Basic edges** are neither branch edges nor rejected edges.

The algorithm description of the algorithm that GHS algorithm assigns a level to each fragment which is a non decreasing integer with the initial value 0, each nonzero level fragment has an ID which is the ID of the core edge in the fragment which is selected when the fragments is constructed during the execution of the algorithm each node can classify each of its incident edges into three categories that is the branch edges is the first category are those that have already been determined to be a part of MST.

The second type of edges rejected edge are those edge that have been already determined not to be a part of MST. Third is the basic edge are neither branches nor the rejected edge.

(Refer Slide Time: 16:53)

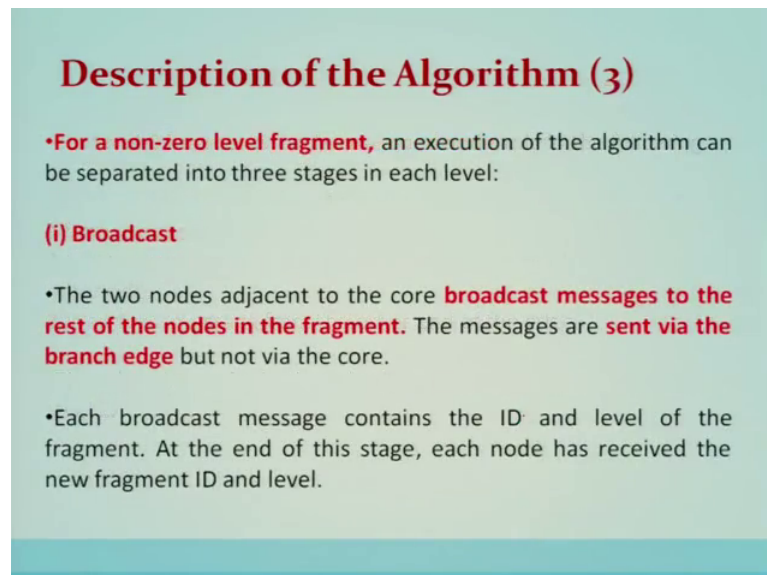


Description of the Algorithm (2)

- For level-0 fragments, each **awakened node** will do the following:
 1. Choose its minimum-weight incident edge and **marks that edge as a branch edge**.
 2. Send a **message via the branch edge** to notify the node on the other side.
 3. **Wait for a message** from the other end of the edge.
- The edge chosen by both nodes it **connects becomes the core** with level 1.

So, the description of algorithm for level 0 fragments each awakened node will do the following. First choose its minimum weight incident edge and marks that edge as the branch edge, send a message via the branches to notify the node on the other side wait for the message from the others end of the edge.

(Refer Slide Time: 17:26)



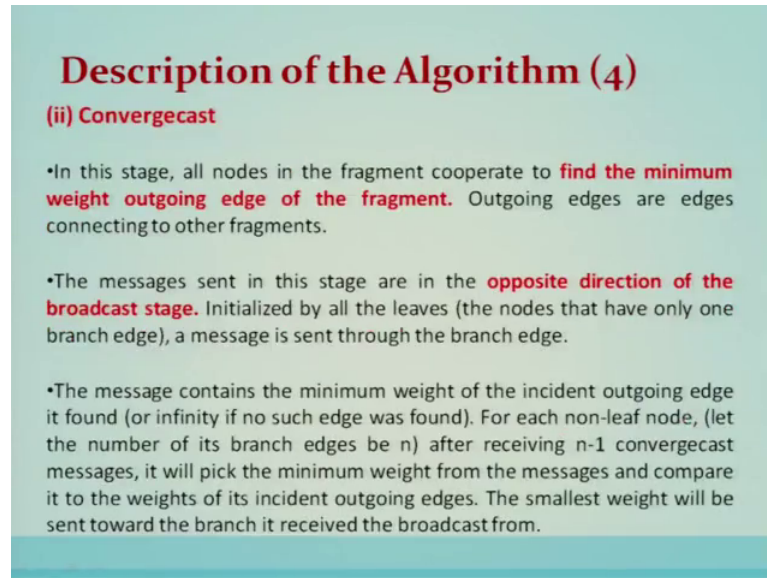
Description of the Algorithm (3)

- For a **non-zero level fragment**, an execution of the algorithm can be separated into three stages in each level:
 - (i) **Broadcast**
 - The two nodes adjacent to the core **broadcast messages to the rest of the nodes in the fragment**. The messages are **sent via the branch edge** but not via the core.
 - Each broadcast message contains the ID and level of the fragment. At the end of this stage, each node has received the new fragment ID and level.

The edge chosen by both the nodes it connects becomes the core with level 1. For nonzero level fragment and execution of the algorithm can be separated into three stages in each level - first is broadcast the two nodes adjacent to the core broadcast messages to

the rest of the nodes in the fragment the messages are sent via the branch. Edge not via the core each broadcast message contains the ID and the level of the fragment at the end of this stage each node has received the new fragment ID and the level.

(Refer Slide Time: 17:58)



Description of the Algorithm (4)

(ii) Convergecast

- In this stage, all nodes in the fragment cooperate to **find the minimum weight outgoing edge of the fragment**. Outgoing edges are edges connecting to other fragments.
- The messages sent in this stage are in the **opposite direction of the broadcast stage**. Initialized by all the leaves (the nodes that have only one branch edge), a message is sent through the branch edge.
- The message contains the minimum weight of the incident outgoing edge it found (or infinity if no such edge was found). For each non-leaf node, (let the number of its branch edges be n) after receiving $n-1$ convergecast messages, it will pick the minimum weight from the messages and compare it to the weights of its incident outgoing edges. The smallest weight will be sent toward the branch it received the broadcast from.

Converge cast at this stage all nodes in the fragment cooperate to find the minimum weight outgoing edge of the fragment; outgoing edges are the edges connecting to other fragment. The messages sent in this stage are in the opposite direction of the broadcast stage initialized by all the leaves the nodes that have only one branch edge a message is sent through the branch edge.

The message contains the minimum weight of the incident outgoing edge it found. For each non leaf node let the number of its branch edges are branch edges be n after receiving n minus 1 converge cast messages it will pick the minimum weight from the messages and compare it to the weight of its incident outgoing edges the smallest weight will be sent towards the branch it received the broadcast from.

(Refer Slide Time: 18:55)

Description of the Algorithm (5)

(iii) Change core

- After the completion of the previous stage, the two nodes connected by the core can **inform each other of the best edges they received**.
- Then they can identify the **minimum outgoing edge from the entire fragment**. A message will be sent from the core to the minimum outgoing edge via a path of branch edges.
- Finally, a message will be sent out via the chosen outgoing edge to request to combine the two fragments that the edge connects.

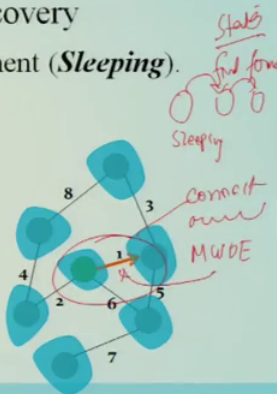
Change core after the completion of the initial stage the two nodes connected by the core can inform each other of the best edge they received then they can identify the minimum outgoing edge from entire fragment. A message will be sent from the core to the minimum outgoing edge via path of the branch edges. Finally, a message will be sent out via choosing via chosen outgoing edge to request to combine the two fragments that the edge connects. So, that was a brief overview of the algorithm.

(Refer Slide Time: 19:29)

The Algorithm: Execution

- **Minimum outgoing edge discovery**
 - Special case of zero-level fragment (**Sleeping**).

*When a node awakes from the state **Sleeping**, it finds a minimum edge connected. Marks it as a **branch** of MST and sends a **Connect** message over this edge. Goes into a **Found** state.*



Now, let us see the execution how these steps are basically incorporated in the execution of the algorithm. So, in this particular execution example the fragment with a minimum outgoing edge discovery is shown that is special case of 0 level fragments they are sleeping when a node awakens from the sleeping state finds a minimum edge connected. Marks it as a branch of the MST and sends a connect message over this edge, goes into a found state.

So, we have seen that the node is making a transition from sleeping to find to or found three states of the process and also send a message that connector over minimum weight outgoing edge which is shown as the red color over here. Now this particular correct message will try to combine the fragments.

(Refer Slide Time: 21:09)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Take a fragment at level L that was just combined out of two level L-1 fragments.

*The weight of the core is the identity of the fragment.
It acts as a root of a fragment tree.*

So, take a fragment at level l that just combined out of the two level L minus 1 fragment the weight of the core is the identity of the fragment it acts as the root of the fragment tree.

(Refer Slide Time: 21:22)

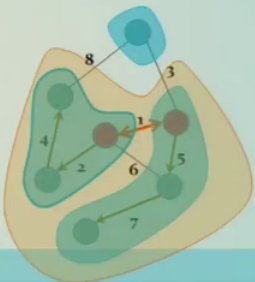
The Algorithm: Execution

- Minimum outgoing edge discovery
 - Take a fragment at level L that was just combined out of two level L-1 fragments.

*Nodes adjacent to core send an **Initiate** message to the borders.*

Relayed by the intermediate nodes in the fragment.

*Puts the nodes in the **Find** state.*



The diagram shows a network fragment with 8 nodes. Node 1 is the core node, highlighted in red. Nodes 2, 3, 4, 5, 6, and 7 are intermediate nodes, highlighted in green. Node 8 is a border node, highlighted in blue. Arrows indicate the flow of an 'Initiate' message from node 1 to nodes 2, 3, 4, 5, 6, and 7.

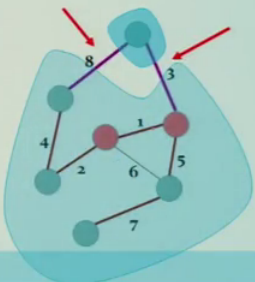
So, the nodes adjacent to the core send an initiate message to the borders, relayed by the intermediate nodes in the fragment puts the node in the find state.

(Refer Slide Time: 21:37)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Edge classification/

Basic - yet to be classified, can be inside fragment or outgoing edges



The diagram shows the same network fragment as the previous slide. Node 1 is the core node, highlighted in red. Nodes 2, 3, 4, 5, 6, and 7 are intermediate nodes, highlighted in green. Node 8 is a border node, highlighted in blue. Red arrows point to the edges connecting node 1 to nodes 2 and 3, and node 8 to nodes 3 and 4, indicating these edges are yet to be classified.

So, the basic edges yet to be classified can be inside the fragment or outgoing edges.

(Refer Slide Time: 21:47)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Edge classification.

Rejected – an inside fragment edge

Rejected will always be inside fragment and branches is an MST edge.

(Refer Slide Time: 21:52)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Edge classification.

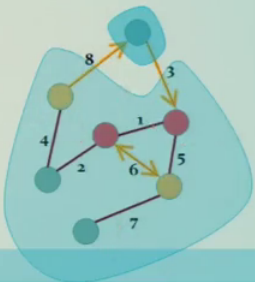
Branch – an MST edge

(Refer Slide Time: 21:56)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the *Initiate* message a node tries to find a minimum outgoing edge.

Sends a *Test* message on *Basic* edges (minimal first)



So, on receiving the initiate message a node tries to find the minimum outgoing edge sends a test message on the basic edge a minimal first.

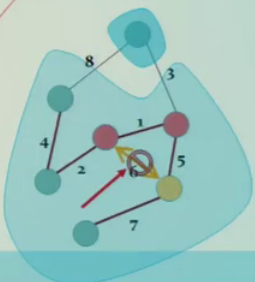
(Refer Slide Time: 22:11)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the *Test* message.

In case of same identity:
send a *Reject* message, the edge is *Rejected*.

In case *Test* was sent in both directions, the edge is *Rejected* automatically without a *Reject* message.



On receiving the test message in case of the same identity send a reject message the edge is rejected same identity means in it is in the same fragment, same fragment the connection will lead to a cycle.

(Refer Slide Time: 21:51)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the *Test* message.

*In case of a self lower level:
Delay the response until the
identity rises sufficiently.*

The diagram shows a network of nodes. A node at level L=0 (blue) is connected to a node at level L=3 (red). A red arrow labeled '8' points from L=3 to L=0, representing a Test message. A yellow arrow labeled '3' points from L=0 to L=3, representing a delayed response. The text 'Response Delayed' is written above the yellow arrow. The diagram shows a network of nodes with levels L=0, L=1, L=2, and L=3.

In case the test was sent in both directions the edge is rejected automatically without a reject message. In case of self lower level delay the response until the identity rises sufficiently.

(Refer Slide Time: 23:03)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the *Test* message.

*In case of a self higher level:
Send an **Accept** message
The edge is accepted as a
candidate.*

The diagram shows a network of nodes. A node at level L=0 (blue) is connected to a node at level L=3 (red). A red arrow labeled '3' points from L=0 to L=3, representing a Test message. A yellow arrow labeled '8' points from L=3 to L=0, representing an Accept message. The diagram shows a network of nodes with levels L=0, L=1, L=2, and L=3.

In case such an accept message the edges accepted as the candidate.

(Refer Slide Time: 23:11)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

Nodes send **Report** messages along the branches of the MST.

If no outgoing edge was found the algorithm is complete.

After sending they go into **Found** mode.

So, the node sent the report messages along the branches of MST, if no outgoing edge was found the algorithm is complete after sending they go in a found state.

(Refer Slide Time: 23:25)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

Every leaf sends the **Report** when resolved its outgoing edge.

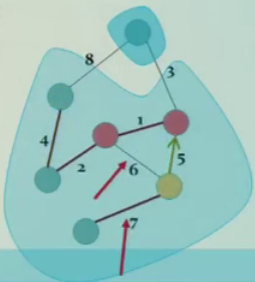
Every leaf sends the report when resolved its outgoing edge.

(Refer Slide Time: 23:33)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

*Every interior sends the **Report** when resolved its outgoing and all its children sent theirs.*



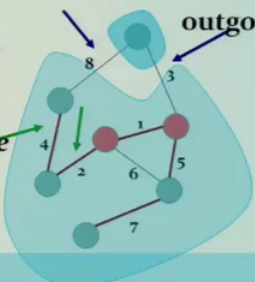
And its children send theirs.

(Refer Slide Time: 23:37)

The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

*Every node remembers the branch to the minimal outgoing edge of its sub-tree, denoted **best-edge**.*



Every node remembers the branch to the minimal outgoing edge of its sub branch denoted the best edge.

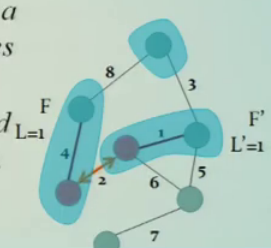
(Refer Slide Time: 24:08)

The Algorithm: Execution

- Final notes
 - Connecting same level fragments.

*Both core adjacent nodes send a **Connect** message, which causes the level to be increased.*

*As a result, core is changed and new **Initiate** messages are sent.*



So, when connecting the same level fragments both core adjacent nodes send a connect message which causes the level to be increased, as a result core is changed and new initiate message are sent.

(Refer Slide Time: 24:29)

The Algorithm: Execution

- Final notes
 - Connecting lower level fragments.

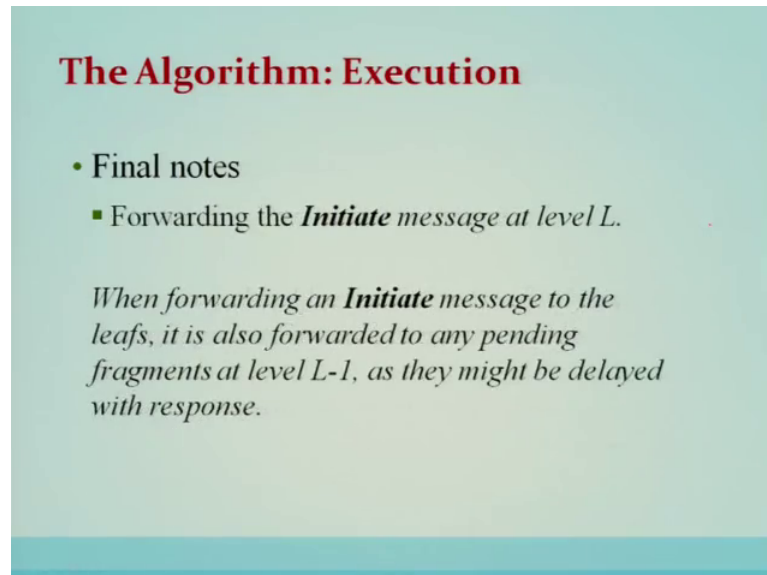
*When lower level fragment F' at node n' joins some fragment F at node n after n sent its **Report**.*

*It means that n already found a lower edge and therefore we can send n' an **Initiate** message with the **Found** state so it doesn't join the search.*

When lower level fragment f prime at a node n prime joins the same fragment at a node n before n sends its report we can send n prime and initiate message with fine listed, so it joins the search. When the lower level fragment F prime at a node n prime joins some fragment f at a node after n sent its report it means that n already found a lower edge and

therefore, we can send n prime an initiate message with the found message. So, it does not join the search.

(Refer Slide Time: 25:07)



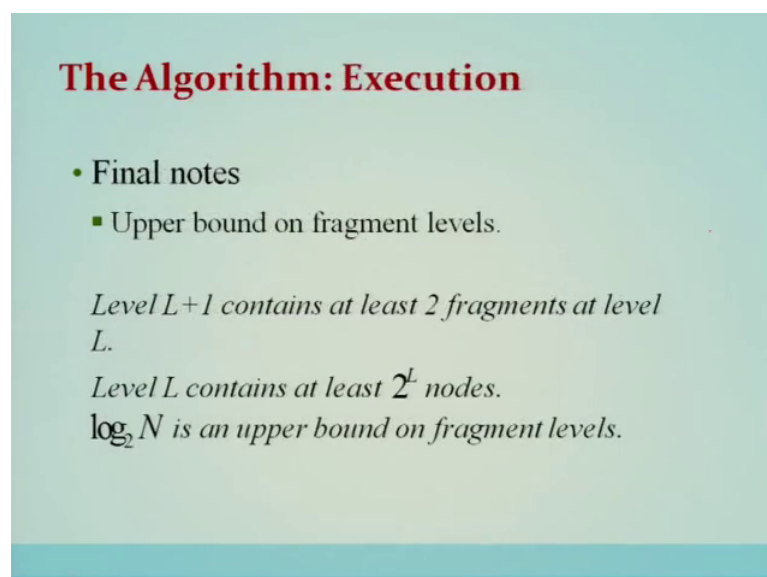
The Algorithm: Execution

- Final notes
 - Forwarding the *Initiate* message at level L .

*When forwarding an **Initiate** message to the leafs, it is also forwarded to any pending fragments at level $L-1$, as they might be delayed with response.*

Forwarding the initiate message at level L when forwarding an initiate message to the leafs, it is also forwarded to any pending fragments at level L minus one as they might be delayed with the response.

(Refer Slide Time: 25:21)



The Algorithm: Execution

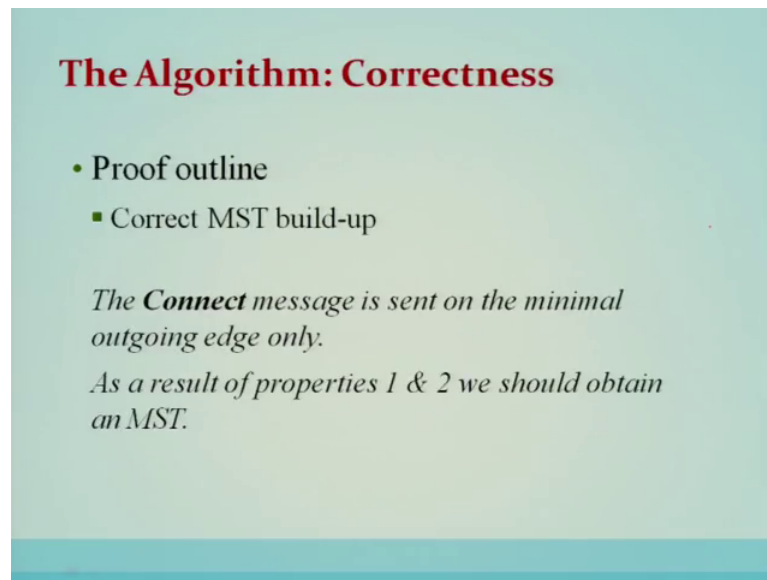
- Final notes
 - Upper bound on fragment levels.

Level $L+1$ contains at least 2 fragments at level L .

Level L contains at least 2^L nodes.

$\log_2 N$ is an upper bound on fragment levels.

(Refer Slide Time: 25:34)



The Algorithm: Correctness

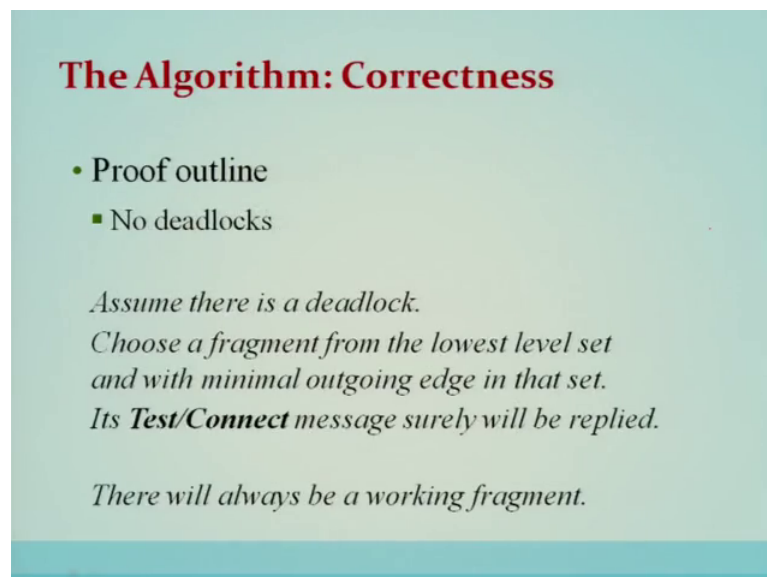
- Proof outline
 - Correct MST build-up

*The **Connect** message is sent on the minimal outgoing edge only.*

As a result of properties 1 & 2 we should obtain an MST.

So, we can see that $\log n$ is an upper bound on the fragment levels and the connect messages send on the minimal outgoing.

(Refer Slide Time: 25:39)



The Algorithm: Correctness

- Proof outline
 - No deadlocks

Assume there is a deadlock.

Choose a fragment from the lowest level set and with minimal outgoing edge in that set.

*Its **Test/Connect** message surely will be replied.*

There will always be a working fragment.

Edges only and there is no deadlock.

(Refer Slide Time: 25:46)

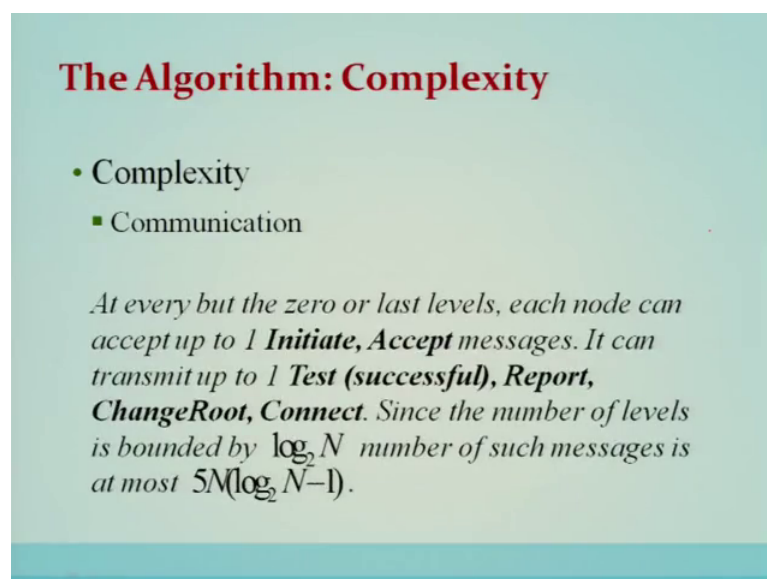


The Algorithm: Complexity

- Complexity
 - Communication

*At most E **Reject** messages (with corresponding E **Test** messages, because each edge can be rejected only once.*

(Refer Slide Time: 25:52)



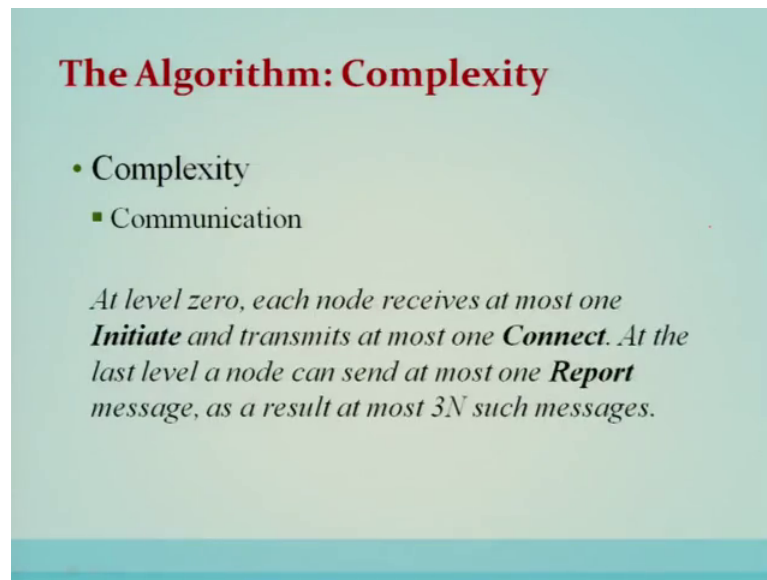
The Algorithm: Complexity

- Complexity
 - Communication

*At every but the zero or last levels, each node can accept up to 1 **Initiate, Accept** messages. It can transmit up to 1 **Test (successful), Report, ChangeRoot, Connect**. Since the number of levels is bounded by $\log_2 N$ number of such messages is at most $5N(\log_2 N - 1)$.*

We have to assume then the communication complexity at each at every, but the 0 or the last levels each node can accept up to 1 initiate accept messages it can transmit up to 1 test report change route connect message. Since the number of levels is bounded by $\log n$ number of such messages is at most $5 n \log n - 5n$.

(Refer Slide Time: 26:19)



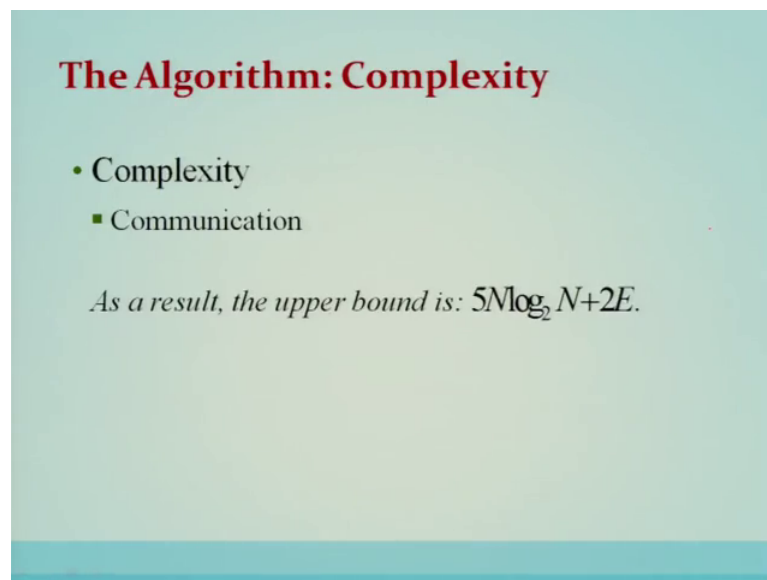
The Algorithm: Complexity

- Complexity
 - Communication

*At level zero, each node receives at most one **Initiate** and transmits at most one **Connect**. At the last level a node can send at most one **Report** message, as a result at most $3N$ such messages.*

At level 0 each node receives at most one initiate and transmit at most one connect at the last level a node can send at most one report message, as a result at most $3N$ messages.

(Refer Slide Time: 26:33)



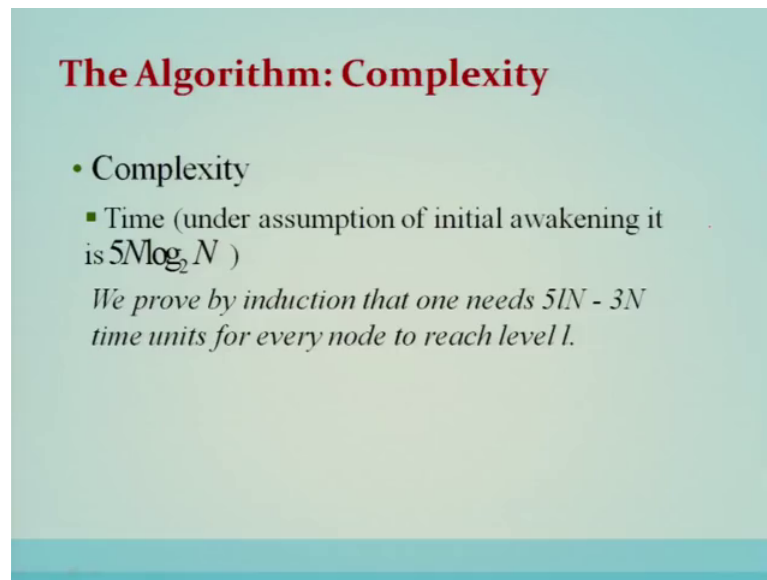
The Algorithm: Complexity

- Complexity
 - Communication

As a result, the upper bound is: $5N \log_2 N + 2E$.

So, as a result the upper bound is $5N \log$ to the base 2 N plus $2E$. So, time under the emission of initial awakening it is $5N \log$ to the base 2 N .

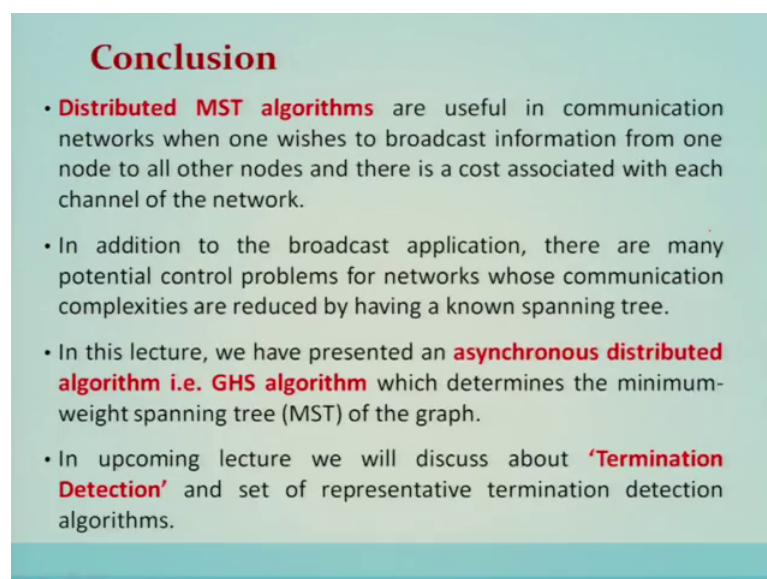
(Refer Slide Time: 26:41)



The Algorithm: Complexity

- Complexity
 - Time (under assumption of initial awakening it is $5N \log_2 N$)
We prove by induction that one needs $5N - 3N$ time units for every node to reach level l .

(Refer Slide Time: 26:51)



Conclusion

- **Distributed MST algorithms** are useful in communication networks when one wishes to broadcast information from one node to all other nodes and there is a cost associated with each channel of the network.
- In addition to the broadcast application, there are many potential control problems for networks whose communication complexities are reduced by having a known spanning tree.
- In this lecture, we have presented an **asynchronous distributed algorithm i.e. GHS algorithm** which determines the minimum-weight spanning tree (MST) of the graph.
- In upcoming lecture we will discuss about **'Termination Detection'** and set of representative termination detection algorithms.

Conclusion, distributed manual spanning tree algorithms are useful in communication network when one wishes to broadcast information from one node to all other nodes and there is a cost associated with each channel of the network. In addition to the broadcast application there are many potential control problems for the network whose communication complexities are reduced by having (Refer Time: 27:13).