Assorted Algorithms

Minimum Spanning Trees, Snapshots

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

Outline

Properties of an MST

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.

GHS Overview

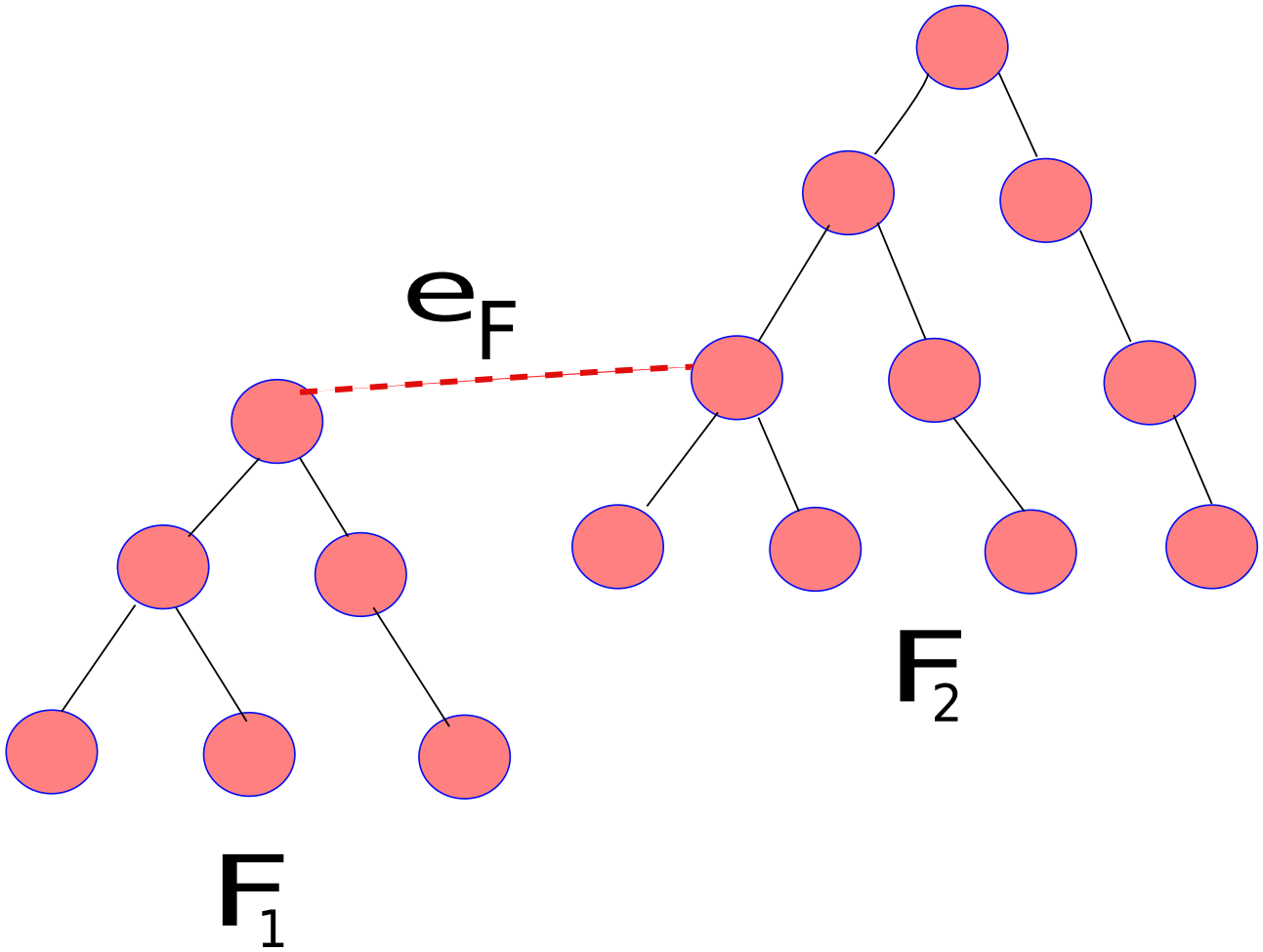
- Initially each node is a fragment.
- Gradually nodes fuse together to make larger fragments. A fragment joins another fragment by identifying its least weight outgoing edge.
- The nodes in a fragment run a distributed algorithm to co-operatively locate the least weight outgoing edge.
- Gradually the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

Properties of a Fragment

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++).

Rules for Combining Fragments



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.

Variables

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

Outline

Initialization

Current node p. Neighbor q.

Algorithm 1: Initialization

1 pq is the least weight edge from p
status[q] $\leftarrow$ branch
level $\leftarrow$ 0
state $\leftarrow$ found
rec $\leftarrow$ 0
send <**connect**,0> to q

Process **connect** Message

Algorithm 2: Processing of the **connect** message

```
1 Receive <connect,L> from q:
  if L < level then
      /* Combine with rule LT                                    */
2     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
```

Receipt of **initiate** message

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
```

findMin

Algorithm 4: *findMin*

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

Receipt of **test** Message

Algorithm 5: Receipt of **test** Message

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

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()
```

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
```

Receipt of **report** Message

Algorithm 8: Process *report* Message

```
1  Receive <report,ω> from q:
   if q ≠ parent then
2      if ω < bestWt then
3          bestWt ← ω
           bestNode ← q
4      end
5      rec ← rec + 1
       report()
6  end
7  else
8      if state ← find then
9          wait
10     end
11     else if ω > bestWt then
12         changeRoot()
13     end
14     else if ω = bestWt = ∞ then
15         stop
```

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()
```

Outline

Time Complexity

Proposition 1

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

Message Complexity

Message Complexity: 2E + 5Nlog(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
  2. receives: 1 accept message
  3. sends: 1 report message
  4. sends: 1 changeroot/connect message
  5. sends: 1 successful test message

Outline

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

Algorithm

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
```

Analysis

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.

Introduction to Distributed Algorithms by Gerard Tel, Cambridge University Press, 2000