Minimal Spanning Tree

CS60002: Distributed Systems

Pallab Dasgupta
Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

Leader Election versus Spanning Tree

- Let C_E be message complexity of the election problem and C_T be the message complexity of finding a spanning tree
- Given a spanning tree, we can run election algorithm on tree to find a leader in O(N) time
  - Thus: $C_E \leq C_T + O(N)$
- Given a leader, we can run the echo algorithm to find a spanning tree with 2|E| messages
  - Thus: $C_T \leq C_E + 2|E|$
- Any comparison election algorithm for arbitrary networks has a (worst case and average case) message complexity of at least $\Omega(|E| + N\log N)$
- Therefore the two problems are of the same order of magnitude

Minimal Spanning Tree

- Let G = (V, E) be a weighted graph, where $\omega(e)$ denotes the weight of edge e.
  - The weight of a spanning tree T of G equals the sum of the weights of the N – 1 edges contained in T
  - T is called a *minimal spanning tree* if no spanning tree has a smaller weight than T.

- *If all edge weights are different, there is only one MST*

The notion of a *fragment*

- A fragment is a subtree of a MST
- If F is a fragment and e is the least-weight outgoing edge of F, then F U {e} is a fragment

- **Prim's Algorithm:**
  - Start with a single fragment and enlarges it in each step with the lowest-weight outgoing edge of the current fragment
- **Kruskal's Algorithm:**
  - Starts with a collection of single-node fragments and merges fragments by adding the lowest-weight outgoing edge of some fragment

Gallager-Humblet-Spira Algorithm

- Distributed algorithm based on Kruskal's algorithm
- <u>Assumptions:</u>
  - Each edge e has a unique edge weight $\omega(e)$
  - All nodes though initially asleep awaken before they start the execution of the algorithm. When a process is woken up by a message, it first executes the local initialization procedure, then processes the message

Gallager-Humblet-Spira Algorithm

- **Algorithm Outline:**
  1) Start with each node as a one-node fragment
  2) The nodes in a fragment cooperate to find the lowest-weight outgoing edge of the fragment
  3) When the lowest-weight outgoing edge of a fragment is known, the fragment will be combined with another fragment by adding the outgoing edge, in cooperation with the other fragment
  4) The algorithm terminates when only one fragment remains

Gallager-Humblet-Spira Algorithm

- **Notations and Definitions:**

1) ***Fragment name.*** To determine whether an edge is an outgoing edge, we need to give each fragment a name.
2) ***Fragment levels.*** Each fragment is assigned a *level*, which is initially 0 for an initial one-node fragment.
3) ***Combining large and small level fragments.*** The smaller level fragment combines into the larger level fragment by adopting the fragment name and level of the larger level fragment. Fragments of the same level combine to form a fragment of a level which is one higher than the two fragments. The new name is the weight of the combining edge, which is called the *core edge* of the new fragment.

Gallager-Humblet-Spira Algorithm

- **Summary of combining strategy:** A fragment F with name FN and level L is denoted as F = (FN, L); let e_F denote the lowest-weight outgoing edge of F.
  - Rule A. If e_F leads to a fragment F′ = (FN′, L′) with L < L′, F combined into F′, after which the new fragment has name FN′ and level F′. These new values are sent to all processes in F
  - Rule B. If e_F leads to a fragment F′ = (FN′, L′) with L = L′ and $e_{F'} = e_F$, the two fragments combine into a new fragment with level L+1 and name $\omega(e_F)$. These new values are sent to all processes in F and F′.
  - Rule C. In all other cases fragment F must wait until rule A or B applies

Gallager-Humblet-Spira Algorithm

- **Node and link status:**
  - $stach_p[q]$: Node p maintains the status of the edge pq.
    - The status is *branch* if the edge is known to be in the MST, *reject* if the edge is known not to be in the MST, and *basic* otherwise.
  - $father_p$: For each process *p* in the fragment, $father_p$ is the edge leading to the core edge of the fragment.
  - $state_p$: State of node *p* is *find* if *p* is currently engaged in the fragment's search for the lowest-weight outgoing edge and *found* otherwise. Initially it is in state *sleep*.

GHS Algorithm: Part-1

var *$state_p$* : (*sleep, find, found*) ;

$statch_p[q]$: (*basic, branch, reject*) for each q $\in$ *$Neigh_p$* ;

$name_p$, *$bestwt_p$* : real ;

$level_p$: integer ;

$testch_p$, *$bestch_p$*, *$father_p$* : *$Neigh_p$* ;

rec_p : integer;

(1) As the first action of each process, the algorithm must be initialized:

begin let *pq* be the channel of *p* with smallest weight ;

$statch_p[q]$:= *branch* ; *$level_p$* := 0 ;

$state_p$:= *found* ; *rec_p* := 0 ;

send ⟨connect, 0⟩ to *q*

end

GHS Algorithm: Part-1

(2) Upon receipt of ⟨connect, L⟩ from *q*:

begin if $L < level_p$ then (* Combine with rule A *)

begin $statch_p[q] := branch$;

send ⟨initiate, $level_p$, $name_p$, $state_p$⟩ to q

end

else if $statch_p[q] = basic$

then (* Rule C *) process the message later

else (* Rule B *)

send ⟨initiate, $level_p + 1$, $\omega(pq)$, $find$⟩ to q

end

GHS Algorithm: Part-1

(3) Upon receipt of ⟨initiate, L, F, S⟩ from *q*:

begin $level_p$:= *L* ; $name_p$:= *F* ; $state_p$:= *S* ; $father_p$:= *q* ;

$bestch_p$:= *udef* ; $bestwt_p := \infty$;

forall $r \in Neigh_p : statch_p[r] = branch \wedge r \neq q$ do

send ⟨initiate, *L, F, S*⟩ to *r* ;

if $state_p$ = *find* then begin rec_p := 0; *test* end

end

Testing the edges

- To find its lowest-weight outgoing edge, node *p* inspects its outgoing edges in increasing order of weight.
- To inspect edge *pq*, *p* sends a ⟨test, $level_p$, $name_p$⟩ message to *q* and waits for an answer
  - A ⟨*reject*⟩ message is sent by process *q* if *q* finds that *p*'s fragment name is the same as *q*'s fragment name. On receiving the ⟨*reject*⟩ message, *p* continues its local search.
  - If the fragment name differs *q* sends an ⟨*accept*⟩ message.
  - The processing of a ⟨*test, L, F*⟩ message is deferred by *q* if L > $level_q$ because *p* and *q* may actually belong to the same fragment, but the ⟨*initiate, L, F, S*⟩ message has not yet reached *q*

A simple optimization

- To inspect edge *pq*, *p* sends a ⟨test, $level_p$, $name_p$⟩ message to *q* and waits for an answer
  - A ⟨*reject*⟩ message is sent by process *q* if *q* finds that *p*'s fragment name is the same as *q*'s fragment name.
  - If the edge *pq* was just used by *q* to send a ⟨*test, L, F*⟩ message then *p* will know (in a symmetrical way) that the edge *pq* is to be rejected. In this case, the ⟨*reject*⟩ message need not be sent by *q*.

GHS Algorithm: Part-2

(4) **procedure** *test*

begin if $\exists q \in Neigh_p : stach_p[q] = basic$ **then**

begin $testch_p := q$ **with**

$stach_p[q] = basic$ **and** $\omega(pq)$ **minimal ;**

send $\langle$**test,** $level_p$, $name_p\rangle$ **to** $testch_p$

end

else begin $testch_p := udef$; *report* **end**

end

GHS Algorithm: Part-2

(5) Upon receipt of ⟨test, L, F⟩ from q:

begin if $L > level_p$ then (* Answer must wait *)

process the message later

else if $F = name_p$ then (* internal edge *)

begin if $stach_p[q] = basic$ then $stach_p[q] := reject$;

if $q \neq testch_p$

then send ⟨reject⟩ to q

else *test*

end

else send ⟨accept⟩ to q

end

GHS Algorithm: Part-2

(6) Upon receipt of ⟨accept⟩ from *q*:

begin $testch_p$:= *udef* ;

if $\omega(pq) < best_p$

then begin $bestwt_p := \omega(pq)$;

$bestch_p := q$

end ;

report

end

(7) Upon receipt of ⟨reject⟩ from *q*:

begin if $stach_p[q]$ = *basic* then $stach_p[q]$:= *reject* ;

test

end

Reporting the lowest-weight outgoing edge

- The lowest-weight outgoing edge is reported for each subtree using ⟨report, ω⟩ messages
  - Node *p* counts the number of ⟨report, ω⟩ messages it receives, using the variable rec_p.
- The ⟨report, ω⟩ messages are sent in the direction of the core edge by each process.
  - The messages of the two core nodes cross on the edge; both receive the message from their father
  - When this happens, the algorithm terminates if no outgoing edge was reported. Otherwise a ⟨connect, *L*⟩ message must be sent through this edge.

Reorientation of the tree

- **If an outgoing edge was reported, the lowest-weight outgoing edge is found by following the *bestch* pointer in each node, starting from the core node on whose side the best edge was reported**
- **A ⟨connect, *L*⟩ message must be sent through this edge, and all *father* pointers in the fragment must point in this direction**
  - **This is done by sending a ⟨changeroot⟩ message.**
  - **When the ⟨changeroot⟩ message arrives at the node incident to the lowest-weight outgoing edge, this node sends a ⟨connect, *L*⟩ message via the lowest-weight outgoing edge.**

GHS Algorithm: Part-3

(8) procedure *report*:

begin if $rec_p = \#\{\, q : stach_p[q] = branch \wedge q \neq father_p \,\}$

and $testch_p = udef$ then

begin $state_p := found$;

send ⟨report, $bestwt_p$⟩ to $father_p$

end

end

GHS Algorithm: Part-3

(9) Upon receipt of ⟨report, ω⟩ from q:

begin if $q \neq father_p$

then (* reply for initiate message *)

begin if $\omega < bestwt_p$ then

begin $bestwt_p := \omega$; $bestch_p := q$ end ;

$rec_p := rec_p + 1$; *report*

end

else (* *pq* is the core edge *)

if $state_p = find$

then process this message later

else if $\omega > bestwt_p$

then *changeroot*

else if $\omega = bestwt_p = \infty$ then stop

end

GHS Algorithm: Part-3

(10) procedure *changeroot*:

begin if $stach_p[bestch_p]$ = *branch*

then send ⟨changeroot⟩ to $bestch_p$

else begin send ⟨connect, $level_p$⟩ to $bestch_p$;

$stach_p[bestch_p]$:= *branch*

end

end

(11) Upon receipt of ⟨changeroot⟩ :

begin *changeroot* end

Complexity

The Gallager-Humblet-Spira algorithm computes the minimal spanning tree using at most 5N logN + 2|E| messages

– **Each edge is rejected at most once and this requires two messages (test and reject). This accounts for at most 2|E| messages.**

– **At any level, a node receives at most one initiate and one accept message, and sends at most one report, one changeroot *or* connect message, and one test message not leading to a rejection. For logN levels, this accounts for a total of 5N logN messages.**