04_04_Minimum Spanning Trees

1. Problem Model

Given a weighted undirected graph $G = (V, E)$

- **Spanning Tree:** $T = (V_T, E_T)$ is a tree with $V_T = V_G$ and $E_T \supset E_G$

Problem:

create a Minimum-weight Spanning Tree $(MST)G' = (V, E')$:

- G’ is a tree (i.e., no cycles)
- G’ has minimum weight (sum of weights of edges)

1.1. Application

broadcasting with minimum cost

1.2. Another way to broadcasting message (Flooding Algorithm)

- The node sends the message **with a hop counter** initialized to the diameter of the graph along each of the edges it is connected to.
- When a node receives the message for the first time,
    - it **decrements the hop counter**

- **sends** the message with the modified hop counter, if this is still positive, on every edge it is connected to, except the edge along which it received the message.
- Messages with a hop counter of 0 are discarded.

1.3. Main difficulty

compute G' in a distributed way

1.4. Weight Assumption

The requirement of different weights in G is not very important if we assume that in G all nodes have different integer identities. If in a graph G edges of equal weights would occur, we can assign every edge e connecting nodes n1 and n2 with identies i1,i2 with i1 <i2 the triple (w(e),i1,i2) as its weight, and use the lexicographic ordering on these weights.

2. Mathematical Interval Lemma

2.1. Pre Lemma

eash sub-graph with n-1 edges but without a circle is a tree

2.2. Lemma 1

A weighted connected undirected graph in which all **weights are different has a unique MST**

Proof:

1. if two different, must a cycle
2. move least weight edge, a more small MST

2.2. Lemma 2

2.2.1. Terminology

Fragment:

A **fragment** of G is a **subtree** of its (unique) MS

Minimum-weight Outgoing Edge(MOE):

The edge e of G is the Minimumweight Outgoing Edge (MOE) of fragment F, if:

- $e \notin F$
- exactly one of the two nodes connected by e is in F
- e has minimum weight among the edges in G with above two properties

2.2.2. Lemma

- let **F** be a **fragment** of G,
- let **e** be its **MOE**,
- let **v** be the node connected by e that is not in F.

Then **F** ∪ **{e}** is a fragment of G.

This lemma means we can **extends** a fragment thorw MOE

3. GHS algorithm

Gallager’s, Humblet’s, and Spira’s algorithm for the MST of an undirected weighted graph with different edge weights

Preparation(Structure)

- Every **node maintains** for each of its **adjacent edges** a **state SE(),** for each edge, it can be **one** of the follow
    - **?_in_MST**: not yet known whether the edge is in the MST
    - **in_MST**: edge is part of the MST
    - **not_in_MST**: edge is not part of the MST
- Every **node** maintains its **own state**
    - **sleeping**: initial state
    - **find**: cooperating to find the MOE of the current fragment
    - **found**: the MOE of the current fragment has been found

  A sleeping node can be **woke up** after receiving a message, once not sleeping anymore, a node **alternates between find and found**
- Every node maintains the following variables:
    - LN: level number of the current fragment it is part of
    - FN: name of the current fragment it is part of
    - SN: state of the node(find/found)
    - in-branch: edge towards core (sense of direction)
    - test-edge: edge checked whether other end in same fragment
    - best-edge: local direction of candidate MOE

- best-weight: weight of current candidate MOE
- find-count number of report messages expected

Core IDEA

- Build up the MST from fragments,
  - start from the set of **single-node fragments**.
  - different nodes/fragments can work as the same time
  - the nodes in a fragment **cooperate to find the fragement's MOE**
- Each fragment has a **level**
  - single-node fragments having **level 0**
  - Every fragment except for level-0 fragments has a unique edge that is called its **core**
  - Every fragment has a fragment **name**,which is the **weight of its core**
- The way to **absorbing fragment (connecting fragment F of level l to fragment F' of level l')**:
  - if **l=l'** and **they have same MOE e,**
    - **joining** F and F’ **along e** yields a new fragment **F” of level L+1**
    - the **link e** because the **core** of F''
    - the weight of e is the **name** of F''
    - **Means**: a fragment of level k has at least 2^k nodes
  - if **l<l'**, if MOE of F is e, if **e is connected to F':**
    - joining F and F' along e, yields a new Fragment F'' of level l'
    - the **core** of F'' is the core of F'
    - the **name** of F''is the name of F'
    - vice versa
  - if (**l<l' or l'<l** but the MOE of smaller one **do not connected** to larger one) or (**l=l'** and **MOEs are not the same**):
    - do not coincide, **wait** until previous two conditions is satified
- When a fragment F' is absorbed at node N to fragment F':
  - if N **has not finished searching** for the candidate MOE in its subtree, it **includes F’ in the search** with an initiate message
  - if **N has already finished searching** for the candidate MOE in its subtree, it sends to F’ an initiate message **only to notify F’ of the current fragment level and name**
  - in the second case, that means

- After an MST has been created, the node of the core of the final result with the **largest id has been elected** (also can be used to **choose a leader**)

Message Types

- **initiate**: start finding the MOE (wave of message from core outwards)
- **test**: check an edge for being a cnadidate MOE
- **accept**: positive answer to test message
- **reject**: negative answer to test message
- **report**: report an candiate MOE found (wave of messages towards core)
- **chang-root**: modify sense of direction towards new core
- **connect**: requesting the connection of two fragments

Detailed Procedure

- In order to find the gragment's MOE:
    - In order to do so, the two nodes connected by the fragment's core broadcast an **initiate** message carrying the **fragment's name** and **level** in their "own" parts of the fragment
    - As a node belongs to different, ever larger fragments in the course of the execution of the algorithm, it **will be involved in MOE**
      finding multiple times
- When a node **receives an initiate message**
    - it forwards an **initiate** message along every edge in state in MST
        - except for the edge along which the initiate message was received,
        - **records the number** of such messages forwarded
    - The node then tries to **select a candidate MOE** among its own adjacent edges
        - checking the **edges with status ?_MST** in the order of increasing weight.
        - An edge is checked by sending a **test** message along it, **containing the fragment's name and level**.(find out whether the node at the other end of the edge is in the same or a different fragment)
- When a node **receives a test message**
    - from a **lower-level fragment** or a fragment of the same level with **a different fragment name**:
        - replies an **accept** message
    - the message has the **same fragement name:**
        - replies with **reject** message

- When a node **receives an accept message:**

    - records the edge as a **potential MOE**
    - because the **test** message is sent in order from small weight to high, so the first accept message means the **MOE**
- When a node **receives a reject message:**
    - sets the state of the **edge to not_in_MST**
- The **potential MOEs and their weights** are **sent back** in the direction of the core with **report** messages
    - Nodes **wait until** they have received the same number of such messages as the number of initiate messages they have forwarded (IX) before reporting the optimal candidate MOEs in the subtrees of which they are the roots.
- When the core nodes have **received all report messages** they expect, and **have exchanged the best candidates in their own subtrees**
    - the MOE of the fragment is known
    - send a **change-root** message from the core to the node that is connected to the MOE
- when the node connected to MOE of its fragment receive **change-root**:
    - sends a **connect message over the MOE**
- when a node receives **connect:**
    - from lower level:
        - abosorb it and send an **initiate** message back
            - if this node **has already finished searching** for the candidate MOE in its subtree, the last argument of the initiate message is **found**
            - if this node **has not finished searching**, the last argument of the initiate message is set to **find**
    - Otherwise:
        - if the edge is not in the node's fragment's MST: appended the connect message into the queue
        - else: merge (means they are same value and has the same MOE)

Implementation

I. Spontaneously starting the algorithm

```
when (SN=sleeping) do
   wakeup()
```

II. Procedure wakeup()

```
j ← adjacent edge of minimum weight
SE(j) ← in_MST; LN ← 0; SN ← found
find-count ← 0
send(connect;0) on edge j
```

III. Receiving a connect message

```
upon receipt of (connect;L) on edge j do
   if (SN=sleeping) then wakeup()
   if (L<LN) then
      SE(j) ← in_MST
      send(initiate;LN,FN,SN) on edge j
      if (SN=find) then
         find-count ← find-count + 1
   else
      if (SE(j)=?_in_MST) then
         append message to message queue
      else
         send(initiate;LN+1,w(j),find) on edge j
```

IV. Receiving an `initiate` message

```
upon receipt of (initiate;L,F,S) on edge j do
   LN ← L; FN ← F; SN ← S
   in-branch ← j
   best-edge ← NIL; best-wt ← ∞
   for all (adjacent edges i, i ≠ j, SE(i) = in_MST do
      send(initiate;L,F,S) on edge i
      if (S=find) then
         find-count ← find-count + 1
   if (S=find) then test()
```

V. Procedure `test()`

```
if (there are adjacent edges in state ?_in_MST) then
   test-edge ← edge in state ?_in_MST of minimum weight
   send(test;LN,FN) on test-edge
else
   test-edge ← nil
   report()
```

VI. Receiving a `test` message

```
upon receipt of (test;L,F) on edge j do
   if (SN=sleeping) then wakeup()
   if (L>LN) then append message to message queue
   else
      if (F ≠ FN) then
         send(accept) on edge j
      else
         if (SE(j)=?_in_MST) then
            SE(j) = not_in_MST
         if (test-edge ≠ j) then
            send(reject) on edge j
         else test()
```

VII. Receiving a `reject` message

```
upon receipt of (reject) on edge j do
   if (SE(j)=?_in_MST) then
      SE(j) ← not_in_MST
   test()
```

VIII. Receiving an `accept` message

```
upon receipt of (accept) on edge j do
   test-edge ← NIL
   if (w(j)<best-wt) then
      best-edge ← j
      best-wt ← w(j)
   report()
```

IX. Procedure `report()`

```
if ((find-count=0) and (test-edge=NIL)) then
   SN ← found
   send(report;best-wt) on in-branch
```

X. Receiving a `report` message

```
upon receipt of (report;w) on edge j do
    if (j ≠ in-branch) then
        find-count ← find-count-1
        if (w<best-wt) then
            best-wt ← w
            best-edge ← j
        report()
    else
        if (SN=find) then
            append message to message queue
        else
            if (w > best-wt) then
                change-root()
            else
                if (w=best-wt=∞) then HALT
```

XI. Procedure `change-root`

```
if (SE(best-edge) = in_MST) then
    send(change-root) on best-edge
else
    send(connect;LN) on best-edge
    SE(best-edge) = in_MST
```

XII. Receiving a `change-root` message

```
upon receipt of change-root do
    change-root()
```

Complexity

Message complexity: GHS algorithm in a graph with N nodes and E edges: 2E + 5N log N