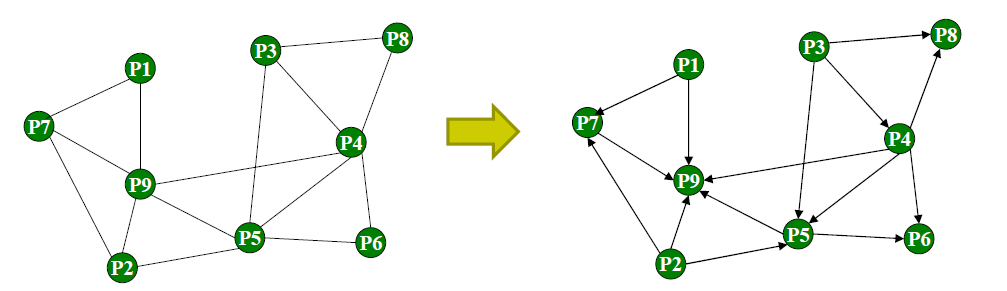
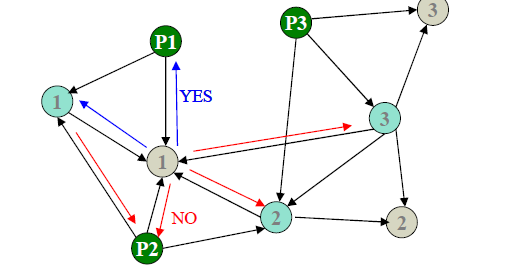
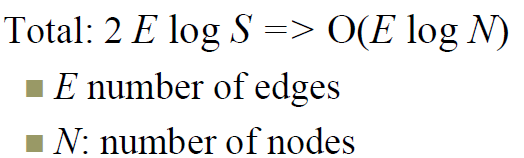
* Yo-Yo algorithm (有兩個階段)
  + pre-processing (Setup)
    - Create a directed acyclic graph (DAG) by orienting edge direction based on node id
    - 
    - 作法
      * 每個 node 都會跟鄰居交換pid，所以最後會有所有鄰居的pid
      * 之後由 pid 小的指向 pid 大的 node，形成directed graph
        + 這個 graph 其實是 acyclic (directed acyclic graph) 因為pid 是由小指向大，如果要形成cycle就代表需要在中間將pid大的指向pid小的node，這樣就不符合我們的定義了
    - Source node: 所有 edge都是out-edges，代表它id最小
    - Sink node: 所有edge都是in-edges，代表他比鄰居的pid都還要大
    - internal node: 非source、sink的node
  + a sequence of iterations of YO- & -YO
    - To eliminate candidate source nodes (只有out-edges，所以代表他比鄰居的pid都還要小，有機會當leader) until only one survives
      * 每個source node都是候選者(因為這個graph是從PID小的指向PID大的，PID最小的會當作leader)，比出最後存活的當成leader
    - phase 1: YO-
      * 這階段是從source node開始，目的是將source node的id送到sink node，讓它比較誰最小
      * 1. Source node送自己的id給所有out-neighbours.
      * 2. internal node等所有的in-neighbours給他id，然後找出最小的id再給它自己所有的out-neighbours.
      * 3. Sink node等待所有的in-neighbours給它id，然後找出最小的id，並進入第二階段
    - phase 2: -YO
      * 這階段從sink node開始，目的是為了減少候補讓source node變成sink或internal node
      * 4. Sink node通知所有in-neighbours，如果pid最小就通知YES，如果不是最小就通知NO
      * 5. Internal node會等待所有out-neighbours的投票結果，如果所有都是YES，就代表這個Internal node當初選擇最小的id勝選了(internal node就像是推舉一個人出去比賽)，所以寄送YES給當初這個最小id的in-neighbours，其他id的in-neighbours就寄送NO。
        + 如果投票結果有任意一個NO，就代表這區推舉的id輸給某個人，所以寄送NO給所有in-neighbours
      * 6. Source node等待從所有out-neighbours寄來投票結果，如果有一個NO這個source就失去資格，所有都YES就活下來，參加下一輪iteration
        + 在下一輪iteration開始前要重新修改Graph，將失去資格的source node去除候選人的資格，並且要記得不能形成cycle
        + 這件事情其實很簡單，只要將NO的edges進行翻轉，即可得到一個去除失去資格的source node後的DAG
        + 這個步驟可分為兩個部分

7. Node x 寄送NO給in-neighbour「y」，就翻轉它的Edge，所以現在y變成x的out- neighbor

8. Node y 從out- neighbor「x」收到NO，翻轉Edge，Node x變成y的in-neighbour

這個跑完後source node就會變成sink node，原本的sind node則可能變成internal node

* + - 
    - 簡單來說
      * 每個source node都會傳自己的PID給鄰近的node，讓它當作仲裁者，看誰的PID比較小，來決定誰是可以存活下來
      * 存活下來的人會收到Yes，輸的人收到No，一直迭代下去直到選出leader
  + 如何知道演算法要停止了?(找出leader了?)
    - add some meta-rules, called Pruning
      * 去除useless(不會影響迭代結果)的node、edge
      * 1. 去除leaf sink node (只有一個in- neighbour的sink node)
        + 因為它只有一個鄰居，它永遠只能投YES，沒有意義，所以可以去除
      * 2. 如果一個node收到多個in- neighbour給予相同source的id，就會只保留一個，要求其他in- neighbour去除他們的edge、node
        + 因為收到重複source的id沒有助於投票，只要保留其中一個就夠了
      * Note: 這兩個步驟做完後都可能會產生新的leaf sink node，如果有就要再次清除
    - 何時進行pruning?
      * during the -Yo phase:
      * pruning is performed when voting.
    - 這樣怎麼判斷演算法終止條件?
      * 如果DAG剩下一個source node，再pruning的第二個步驟就會一直修剪(因為大家都會拿到同一個source)，修剪到最後就會只剩下source node一個node，這時候演算法就可以終止了(Ex: 判斷沒有任何neighbor)
  + Safety
    - 最後只會有一個PID最小的當leader
  + Liveness
    - 如果PID最小的掛了，就會由PID第二小的當上leader
  + message complexity
    - 
      * 比前面多了一個E
    - 可以透過pruning減少其複雜度
  + Time complexity
    - 老師說回去自己想
    - 我查到好像還是O(E log N)，但看不太懂原因