第二部分 分布式算法

第六次课

中国科学技术大学计算机系
国家高性能计算中心（合肥）

§ 3.3.3 下界Ω(*n*lg*n*)

现证明对于uniform算法，异步环里任何leader选举算法至少发送**Ω(*n*lg*n*)**个**msgs**。

我们的下界证明是针对leader选举问题的一个变种：

- 选中的leader必定是环上具有最大id的处理器。
- 所有处理器必须知道被选中leader的id，即每处理器终止前，将选中leader的id写入一个特殊变量。

- **基本思想。**

设A是一个能解上述leader选举变种问题的均匀算法，证明存在A的一个允许执行，其中发送了**Ω(*n*lg*n*)个msgs，证明采用构造法。**

§3.3.3 下界Ω(*n*lg*n*)

对于大小为n/2的环构造算法的一个耗费执行(指msg的耗费)，然后将两个大小为n/2的不同环粘贴在一起形成一个大小为n的环，将两个较小环上的耗费执行组合在一起，并迫使θ(n)个附加msg被接收。这种扩展依赖于算法是一致的且对各种规模的环以相同的方式执行

- 调度：前面定义过调度是执行中的事件序列，下面给出能够被粘贴在一起的调度。
- Def3.1 开调度

设σ是一个特定环上算法A的一个调度，若该环中存在一条边e使得在σ中，边e的任意方向上均无msg传递，则σ称为是open，e是σ的一条开边。

§ 3.3.3 下界Ω($n$lg$n$)

Note：开调度未必是容许的调度，即它可能是有限的事件序列，环上的处理器不一定是终止的。

直观上，既然处理器不知道环的大小，我们能将两个较小的开调度粘贴为一个较大环的开调度，其依据是：算法是均匀的。

为简单起见，不放设n为2的整数次幂。

Th3.5 对于每个n及每个标识符集合(大小为n)，存在一个由这些标识符组成的环，该环有一个A的开调度，其中至少接收M(n)个消息，这里：

$$\begin{cases} M(2)=1 & n=2 \\ M(n)=2M(\frac{n}{2})+\frac{1}{2}(\frac{n}{2}-1) & n>2 \end{cases}$$

§ 3.3.3 下界Ω($nlgn$)

显然递归方程的解为M(n)=θ($nlgn$)，他蕴含了异步环选举问题消息复杂度下界。下面用归纳法证明之，其中

$$\begin{cases} \text{引理3.6是归纳基础}(n=2^1) \\ \text{引理3.7是归纳步骤}(n=2^i, i>1) \end{cases}$$

Lemma3.6 对每个由两个标识符构成的集合，存在一个使用这两个标识符的环R，R有A的一个开调度，其中至少有一个msg被接受。（归纳基础）

pf：假定R有两个处理器P_0和P_1，其标识符分别为x和y，不妨设x>y.

§ 3.3.3 下界Ω(*n*lg*n*)

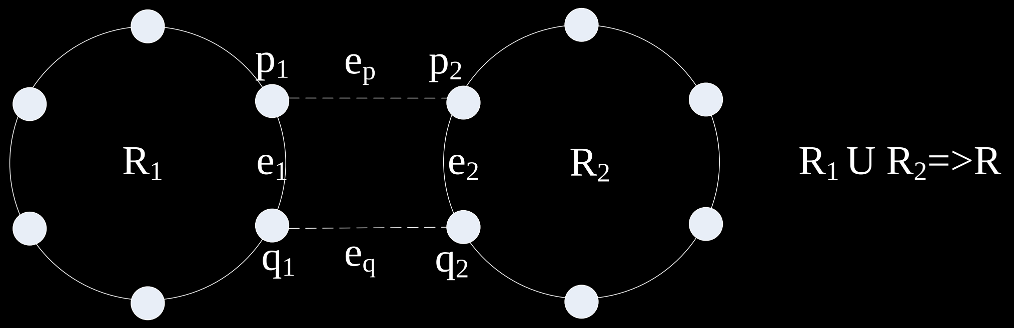
设α是A的一个容许执行，因为A是正确的，在α中，最终P_1定将P_0的标识符写入其中。因此，α中至少须接收一个msg，否则P_1不知道P_0的标识符为x.

设σ是α的调度的最短前缀：它包括第一个接受msg的事件。因为没有接收第一条msg的边是开的，因此σ中只有一个msg被接收且有一条开边，故引理成立。故σ是满足引理的开调度。

Lemma 3.7 选择n>2，假定对每个大小为n/2标识符集合，存在一个使用这些标识符的环，它有A的一个开调度，其中至少接收M(n/2)个msgs(归纳假设)，那么对于n个标识符的每个集合，存在一个使用这些标识符集的环，它有A的一个开调度，其中接收至少2M(n/2)+(n/2-1)/2个msgs(归纳步骤)。

§ 3.3.3 下界$\Omega(n\lg n)$

pf：设S是n个标识符的集合，将S划分为两个集合S_1和S_2，每个大小为n/2，由假设分别存在一个使用S_1和S_2中标识符的环R_1和R_2，它们分别有A的一个开调度σ_1和σ_2，其中均至少接收M(n/2)个msgs，设e_1和e_2分别是σ_1和σ_2的开边，不妨设邻接于e_1和e_2的处理器分别是p_1，q_1和p_2，q_2，将e_1，e_2删去，用e_p链接p_1和p_2，e_q链接q_1和q_2，即可将两个环R_1和R_2粘贴在一起形成环R。



现说明如何在R上构造一个A的开调度σ，其中至少有2M(n/2)+(n/2-1)/2个msg被接收。其想法是先让每个较小环分别执行“耗费”的开调度。

§ 3.3.3 下界$\Omega(n\lg n)$

1) $\sigma_1\sigma_2$构成R上A的一个开调度

考虑从R的初始配置开始发生的事件序列σ_1，因为R_1中的处理器由这些事件并不能区别R_1是一个独立的环还是R的一个子环，它们执行σ_1恰像R_1是独立的那样。考虑环R上后续事件序列σ_2(与上类似)，因为没有msg在e_p和e_q上传递，故R_2中处理器在σ_2中亦不能区别R_2是独立环还是R的子环。

因此，$\sigma_1\sigma_2$是一个调度，其中至少有2M(n/2)个msgs被接收。

2) 现说明如何通过连通e_p和e_q(但不是二者)来迫使算法接收(n/2-1)/2个附加的msgs。

考虑每个形式为$\sigma_1\sigma_2\sigma_3$的有限调度，因为$\sigma_1\sigma_2$中e_p和e_q均为开的，若σ_3中存在一边上至少有(n/2-1)/2个msg被接收，则$\sigma_1\sigma_2\sigma_3$是要找的开调度，引理被证。

假设没有这样的调度，那么存在某个调度$\sigma_1\sigma_2\sigma_3$，它导致相应执行中的一个“静止”配置。(配置：由全体结点状态构成)

一个处理器状态是“静止”的：若从该状态开始的计算事件序列中不send消息，即处理器接收一个msg之前不发送另一msg（即处理器的内部事件不引发send动作）

§ 3.3.3 下界Ω($nlgn$)

一个配置是“静止”的(关于e_p和e_q)：若除开边e_p和e_q外，没有msgs处在传递之中，每个处理器均为静止状态。

不失一般性，假设R中最大id的处理器是在子环R_1中，因为没有msg从R_1传到R_2中，R_2中的处理器不知道leader的id，因此R_2里没有处理器能够在$\sigma_1\sigma_2\sigma_3$结束时终止。(在$\sigma_1\sigma_2\sigma_3$结束时，R_2里无结点终止)

我们断定在每个扩展$\sigma_1\sigma_2\sigma_3$的容许调度里，子环R_2里的每个处理器在终止前必须接收至少一个附加msg，因为R_2里每一处理器只有接收来自R_1的msg才知道leader的id。

上述讨论清楚地蕴含在环R上必须接收Ω(n/2)个msgs，但因为e_p和e_q是连通的，故调度未必是开的，即两边上均可能传递msg。

但若能说明e_p或e_q只有一个是连通的，迫使通过它接收Ω(n/2)个msgs，即可证明。这就是下一断言。

§ 3.3.3 下界Ω($n$lg$n$)

Claim3.8 存在一个有限的调度片断σ_4，其中有 (n/2-1)/2个msgs被接收，$\sigma_1\sigma_2\sigma_3\sigma_4$是一个开调度，其中$e_p$或$e_q$是开的。

Pf: 设 σ_4 使得$\sigma_1\sigma_2\sigma_3$ σ_4 是一个容许调度，因此所有的msgs在e_p和e_q上传递，所有结点终止。

因为R_2里，每个节点在终止前必须收到一个msg，故在A终止前在 σ_4 里至少接收n/2个msgs,设 σ_4' 是 σ_4'' 里接收n/2-1个msg的最短前缀。

考虑 R 里在 σ_4 中所有已接收msg的结点，因为我们是从一个静止位置开始的，其中只有在e_p和e_q上有msg在传输，故这些结点形成了两个连续的结点集合P和Q：

P包含由于连通e_p而被唤醒的结点，故P至少包含p_1和p_2

Q包含由于连通e_q而被唤醒的结点，故Q至少包含q_1和q_2

§ 3.3.3 下界Ω(*n*lg*n*)

因为P∪Q中至少包含n/2-1个结点（由 σ_4' 决定），且又因它们中的结点是连续的，所以P∩Q=Φ。Ｐ和Ｑ这两个集合中有一个集合，其中的结点至少接收 (n/2-1)/2个msg,(因为Ｐ，Ｑ中的结点共接收n/2-1个msg),不失一般性，假定这样的集合是Ｐ。

设σ_4是 σ_4'的子序列，σ_4只包含在Ｐ中结点上发生的事件，因为 σ_4'里Ｐ中节点和Q中结点之间没有通信，故σ_1 σ_2 σ_3 σ_4是一个调度。

因为σ_4里至少有 (n/2-1)/2各msg被接收，且由构造可知，e_q上无msg传递，因此σ_1 σ_2 σ_3 σ_4是一个满足要求的开调度。

§ 3.3.3 下界Ω($n$lg$n$)

总结： Th3.5的证明可分为3步：

1) 在 R1和R2 上构造2个独立的调度，每个接收2M(n/2)各msg: σ_1 σ_2

2) 强迫环进入一个静止配置： σ_1 σ_2 σ_3 （主要由调度片断σ_3 ）

3) 强迫(n/2-1)/2个附加msg被接收，并保持ep或eq是开的: σ_1 σ_2 σ_3 σ_4。

因此我们已构造了一个开调度，其中至少有2M(n/2)+ (n/2-1)/2个msg被接收。

§ 3.4 同步环

研究同步环上leader选举问题的上、下界。

- 上界：提出两个msg复杂性为O(n)的算法，显然，这样的算法的msg复杂性是最优的。但运行时间并非只与环大小n有关，还与算法使用的非普通的id相关。(与id值相关)
- 下界：讨论

1）只基于标识符之间比较的算法

2）时间受限（即若干轮内终止，轮数只依赖于环大小,不依赖于id值）的算法

当算法受限于上述两个条件时，至少需要$\Omega(n\lg n)$个msgs.

§ 3.4.1 上界 O(n)

上节已证明在异步环上leader选举问题的msg复杂度下界为Ω($n\lg n$)，其算法的关键是msg延迟是任意长的。

因为同步环上

1) msg延迟是确定的，故同步模型是否有较好的结果呢？

2) 获取info不仅来自于接收msg,在某轮里的内附件也能获取info

本节提出两个算法，msg复杂性上界为O(n),针对单向环,但是用于双向环

1) 非均匀的：要求环中所有的结点开始于同一轮，标准的同步模型 (需知道n)

2) 均匀的： 结点可开始于不同轮，弱同步模型 (无需知道n)

§ 3.4.1 上界O(n)

1. Non-uniform Alg

- **基本特征**

  选择环中最小id（各id互不相同）的结点作为leader，按Phase运行，每个阶段由n个轮组成。在Phase i (i ≥ 0)，若存在一个id为i的结点，则该结点为leader，并终止算法，因此，最小id的结点被选为leader

  显然，Phase数目取决于n个节点的标志符的取值。

- **具体实现**

  Phase i包括轮：n·i+1, n·i+2, …, n·i+n

  在第i阶段开始，若一个结点的id是i，且它尚未终止，则该节点绕环发送一个msg后作为一个leader终止；若一结点的id不是i，且它收到一个msg，则它转发此msg后作为non-leader终止。

§ 3.4.1 上界O(n)

- 分析

正确性：显然，只有最小的标志符被选中作为leader

Msg复杂性：恰有n个msg被发送，故复杂性为O(n)。注意这n个msg均是在找到leader的那个Phase里发送的。

时间复杂性

依赖于环大小和环上最小标志符，不妨设环大小为n，最小标识符为i，则算法执行轮数为：$n\cdot(i+1)$，不妨设i≥0 //运行时间与环大小及标识符取值相关

缺点

必须知道环大小n和同步开始，下面算法克服了这些限制

①为什么id为i的结点要在phase i发msg

∵各结点互不知道彼此的id值

∴只能在第i phase，结点(id=i)发自己的id

②为什么每个phase要n轮

§ 3.4.1 上界O(n)

2. Uniform Alg

- **特点：**①无须知道环大小，②弱同步模型

  一个处理器可以在任意轮里自发地唤醒自己，也可以是收到另一个处理器的msg后被唤醒

- **基本思想**

  ① 源于不同节点的msg以不同的速度转发

  源于id为i的节点的msg，在每一个接收该msg的节点沿顺时针转发到下一个处理器之前，被延迟2^i-1轮

  ② 为克服非同时启动，须加一个**基本的唤醒阶段**，其中每个自发唤醒的结点绕环发送一个唤醒msg，该msg转发时无延迟

  ③ 若一个结点在算法启动前收到一个唤醒msg，则该结点不参与算法，只是扮演一个relay(转发)角色：即转发或没收msg

§ 3.4.1 上界O(n)

- **要点**：在基本阶段之后，选举leader是在参与结点集中进行的，即只有自发唤醒的结点才有可能当选为leader。
- **具体实现**

  ① **唤醒**：由一个结点发出的唤醒msg包含该结点的id，该msg以每轮一边的正常速率周游，那些接收到唤醒msg之前未启动的结点均被删除(不参与选举)

  ② **延迟**：当来自一个id为i的节点的msg到达一个醒着的节点时，该msg以2^i速率周游，即每个收到该msg的节点将其延迟2^i-1轮后再转发。

Note：　一个msg到达一个醒着的节点之后，它要到达的所有节点均是醒着的。一个msg在被一个醒着的节点接收之前是处在1st阶段 (唤醒msg，非延迟)，在到达一个醒着的节点之后，它就处于2nd阶段，并以2^i速率转发(非唤醒msg，延迟)

§ 3.4.1 上界O(n)

③ 没收规则

a) 一个参与的节点收到一个msg时，若该msg里的id大于当前已看到的最小(包括自己)的id，则没收该msg；

b) 一个转发的节点收到一个msg时，若该msg里的id大于当前已看到的最小(不包括自己)的id，则没收该msg。

§ 3.4.1 上界O(n)

- **算法 Alg3.2 同步leader选举**

```
var waiting : init  Φ;
      asleep  : init true;    //加上relay更好？    : init false;
1: 设R是计算事件中接收msg的集合
2: s:= Φ;// the msg to be sent
3: if asleep then {
4:     asleep:=false;
5:     if R = Φ then { // pi未收到过msg，属于自发唤醒
6:          min:=id;   //参与选举
7:          s:=s+{<id>}; // 准备发送
       }else{ //已收到过msg，但此前未启动，被唤醒故Pi不参与
8:          min:=∞; //选举，置min为∞， 使其变为relay结点
            // relay:=true; ？
       }
   }
```

§ 3.4.1 上界O(n)

```
9:   for each <m> in R do {// 处理完收到的m后相当于从R中删去
10:      if m < min then { // 收到的id较小时通过
11:            become not elected;  // P_i未被选中
               //可用relay控制使转发节点不延迟？
12:            将<m>加入waiting且记住m何时加入; //m加入延迟转发
13:            min:=m;
         } // if m > min then it is swallowed
14:      if m=id then become elected; // P_i被选中
     } //endfor
15:  for each <m> in waiting do
16:      if <m> 是在2^m-1轮之前接收的 then
17:            将<m>从waiting中删去并加入S
18:  send S to left;
```

§ 3.4.1 上界O(n)

- **分析**

下面证明，在第1个结点被唤醒之后的n轮，只剩下第二阶段的msg，只有参与的结点才有可能被选中。

(1) 正确性

对$\forall i \in [0,n-1]$，设id_i是结点p_i的标识符，$\langle id_i \rangle$是源于p_i的msg

Lemma 3.9 在参与的节点中，只有最小id的节点才能收回自己的id。

pf：①选中：设p_i是参与者中具有最小id的结点(Note：至少有1个结点须参与算法)，显然没有节点(无论是否参与)能没收$\langle id_i \rangle$；另一方面，因为在每个节点上$\langle id_i \rangle$至多延迟2^{id_i}轮，故p_i最终收回自己的id；

②唯一：除p_i外，没有别的节点p_j ($j \neq i$) 也收回自己的$\langle id_j \rangle$。

若p_j收回自己的$\langle id_j \rangle$ ，则$\langle id_j \rangle$已通过p_i及其它所有结点，但$id_i < id_j$，因为p_i是一个参与者，它将不会转发id_j，矛盾！

该引理蕴含着：恰有一个结点收回自己的msg，故它是唯一声明自己是leader的结点，即算法正确。

§ 3.4.1 上界O(n)

(2) msg复杂性

在算法的一次容许执行里，发送的msg可分为三个类型：

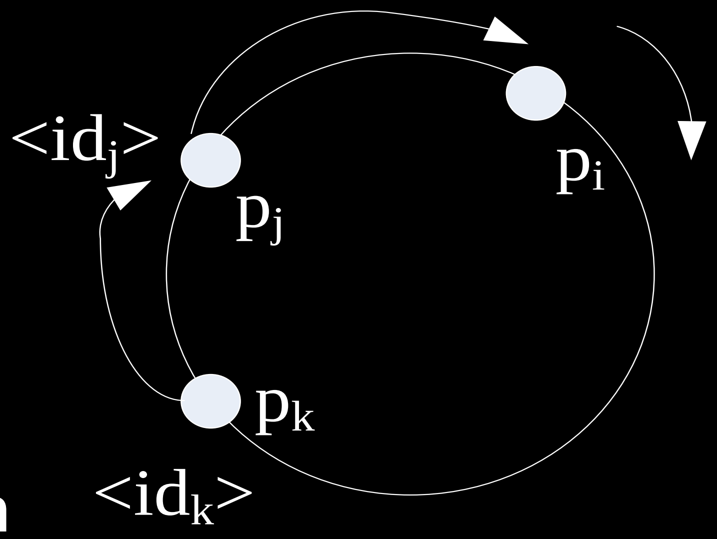
第一类：第一阶段的msg(唤醒msg)

第二类：最终leader的msg进入自己的第二阶段之前发送的第二阶段msg(其它结点发出的)

第三类：最终leader的msg进入自己的第二阶段之后发送的第二阶段msg(包括leader发出的)

Note：一个msg发送时，一开始是作为唤醒msg(非延迟)，当它到达的结点已唤醒时，msg就变为非唤醒msg(第二阶段，延迟msg)

§ 3.4.1 上界O(n)



① 第一类msg总数(第一阶段的msg)

Lemma 3.10 第一类msg总数至多为n

pf： 只要说明每个节点在第一阶段至多转发一个msg即可。

反证： 假设某结点p_i在其第一阶段转发两个msgs：一个来自p_j的$<id_j>$，一个来自p_k的$<id_k>$。不失一般性，p_j比p_k更靠近p_i(沿顺时针方向)。因此，$<id_k>$到达p_i之前先到达p_j。

若$<id_k>$是在p_j自发唤醒及发送$<id_j>$之后到达p_j，则$<id_k>$以2^{id_k}速度作为第二阶段msg继续前进；否则，p_j不是参与结点，不会发送$<id_j>$。

因此，或者$<id_k>$是作为第二阶段msg到达p_i，或者$<id_j>$未被发送，即：p_i最多收到一个第一阶段msg，矛盾！

§ 3.4.1 上界O(n)

② 第二类msg总数 (最终leader发出的msg进入自己的第二阶段之前发送的第二阶段msg)

为了求得第二类msg总数，首先说明第一个开始执行算法的结点启动之后的n轮，所有的msg均在自己的第二阶段中。

设p_i是最早开始执行算法的结点中的某一个，其启动轮数为r。

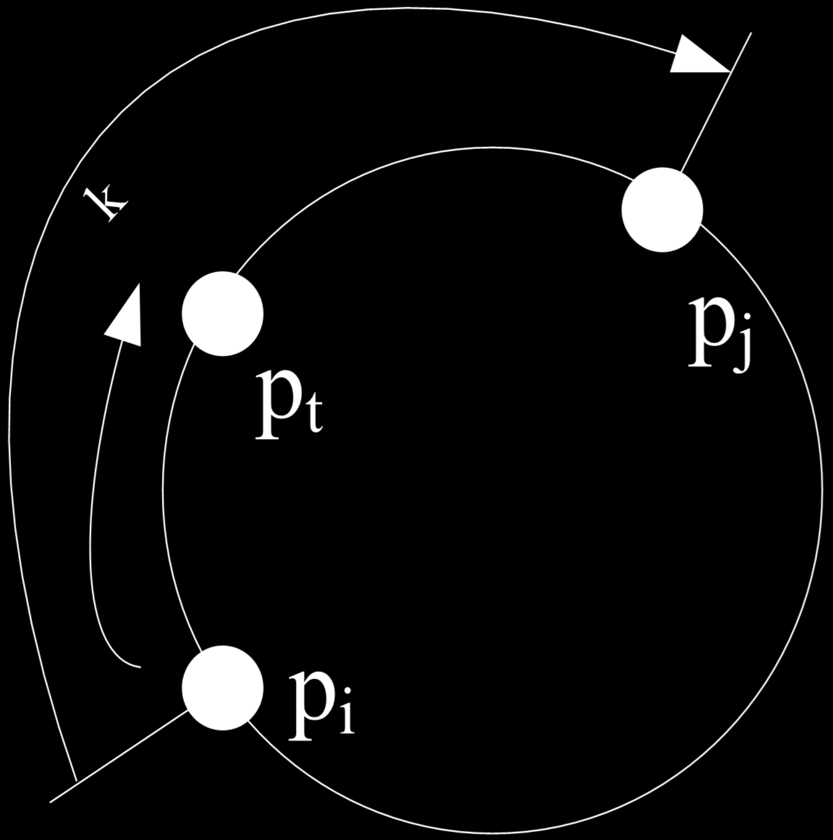
Lemma 3.11 若p_j距离p_i为k(顺时针)，则p_j 接收的第一阶段的msg不迟于第r+k轮。

§ 3.4.1 上界O(n)

pf：对距离k归纳

基础：k=1，因为p_i的左邻居在第r+1轮接收到p_i的msg，故引理成立。

假设：设距离p_i为k-1的结点接收第一阶段的msg不迟于r+k-1轮。



步骤

若距离p_i为k-1的结点p_t(顺时针)接收第一阶段msg时已被唤醒，则p_t已发送了第一阶段msg给邻居p_j；

否则，p_t至迟在第r+k轮转发第一阶段msg到p_j。

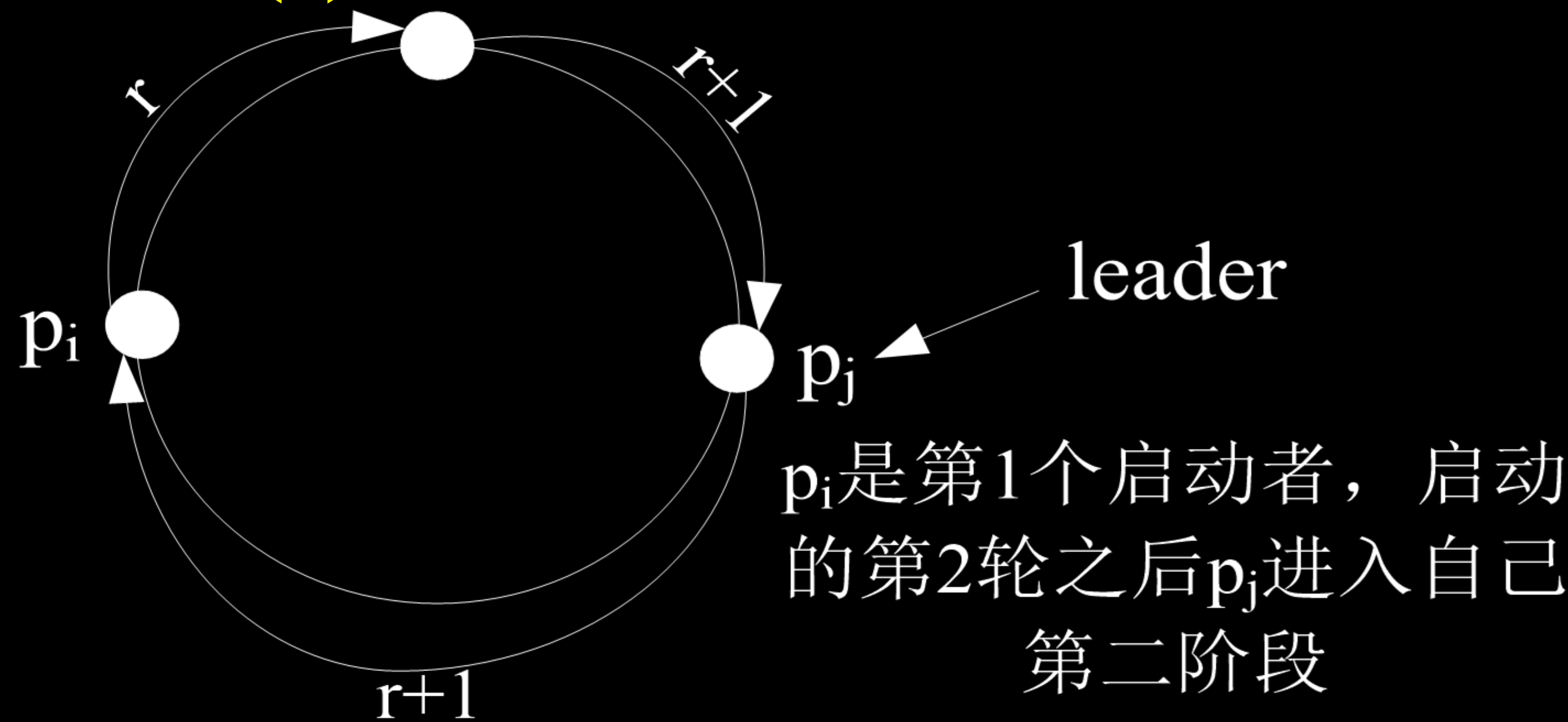
§ 3.4.1 上界O(n)

Lemma 3.12 第二类msg总数至多为n

pf：由引理3.10可知，在每边上至多只发送1个第一阶段的msg，又因为到第r+n轮，每边上已发送了一个第一阶段msg，故到第r+n轮之后，已无第一阶段的msg被发送。

Note：第一阶段msg是唤醒msg，即若在p_i(第一个启动结点)发出唤醒msg绕环一周回到p_i之前已有某结点启动，则该启动结点的msg在未收到p_i的msg之前已将自己的唤醒msg向前转发。

§ 3.4.1 上界O(n)



i) 第二类msg经历的总轮数：

由引理3.11知，最终的leader(不一定是首个启动者)的msg进入自己的第二阶段的时刻是：算法的第1个msg被发送之后至多n轮(前n轮)，故第二类被发送的msg必是在首个启动结点的n轮之中。

ii) 在这n轮中，第二类msg数目。即第二类msg是算法启动的前n轮中非唤醒msg的总数：

§ 3.4.1 上界O(n)

因为msg<i>在其第二阶段中，转发前须延迟2^i-1轮，所以若<i>是第二类msg，则它至多被发送$n/2^i$次。

因为较小id被转发的次数较多，故可这样构造以使msg数目最大：

所有结点均参与选举，标识符均尽可能小：0, 1, …, n-1（顺时针排列）。显然，因为id=0是leader，第二类msg中不包括leader的msg，故第二类msg总数至多是：

$$\sum_{i=1}^{n-1} \frac{n}{2^i} \le n$$

§ 3.4.1 上界O(n)

③第三类msg总数

(即：leader进入自己的第二阶段之后，所有非唤醒msg)

Lemma 3.13 在<id_i>返回p_i之后，没有msg被转发

pf：

设p_i是具有最小id的结点，当一结点转发<id_i>之后，该结点将不再会转发其它msg。

若<id_i>返回p_i，则所有结点均已转发过<id_i>，故再也没有其它msg被转发。

§ 3.4.1 上界O(n)

Lemma 3.14 第三类msg总数至多为2n

pf：设p_i是最终的leader，p_j是某个参与的结点，由引理3.9知，$id_i<id_j$，由引理3.13知，在p_i收回自己的id_i之后，环上不再有msg在传输。

i) 第三类msg经历的总轮数：因为在每个结点上，$<id_i>$至多延迟2^{id_i}轮(在唤醒结点上不延迟，故为至多)，所以$<id_i>$返回p_i至多经过$n\cdot 2^{id_i}$轮。

ii) 在这$n\cdot 2^{id_i}$轮中，第三类msg数目：第三类msg是在这$n\cdot 2^{id_i}$轮中发送的所有第二阶段msg(非唤醒msg)。在这$n\cdot 2^{id_i}$轮中，一个非唤醒的msg$<id_j>$被转发的次数至多为：

$$\frac{1}{2^{id_j}}\cdot n\cdot 2^{id_i} = n\cdot 2^{id_i - id_j}$$

§ 3.4.1 上界O(n)

故有：第三类msg总数至多为(包括leader)

$$\sum_{j=0}^{n-1} \frac{n}{2^{id_j - id_i}}$$

基于引理3.12的同样理由，当所有结点参与选举，及标识符为0, 1, …, n-1时有：

$$\sum_{j=0}^{n-1} \frac{n}{2^{id_j - id_i}} \le \sum_{k=0}^{n-1} \frac{n}{2^k} \le 2n$$

这里，$\forall id_i \in [0, n-1]$

Th3.15 存在一个同步的leader选举算法，其msg复杂度至多为4n

pf：由引理3.10，3.12及3.14立即可得。

§ 3.4.1 上界O(n)

(3) 时间复杂度

由引理3.13知，当leader接收到自己的id时，计算终止。这发生在第一个启动算法的节点之后的$O(n\cdot 2^i)$轮，其中i是leader的标识符。 // 当i=0时，为O(n)轮

//运行时间与环大小及标识符取值相关

(4) 思考

为何非唤醒msg要延迟2^i -1轮？

如何修改算法3.2来改善时间复杂性？

下次继续！