算法第二次作业

李博杰 SA14011026

2.1 分析在同步和异步模型下，convergecast算法的时间复杂性。

解：（1）同步模型：由于 convergecast 算法按照生成树来汇聚最大值，每个节点仅向其父节点发送一条消息，共有 n-1 条消息。最坏情况下算法的每一轮中只有一条消息传递（当生成树退化成链表时），共有 n-1 轮，故时间复杂度为 O(n-1)。

（2）异步模型：每个节点到其距离最远的叶子节点的路径长度决定了其收到消息的时间。最坏情况下路径长度为 n-1（当生成树退化成链表时），故时间复杂度为 O(n-1)。

2.2 证明在引理2.6中，一个处理器在图G中是从Pr可达的，当且仅当它曾设置过自己的parent变量。

证明：（1）必要性：处理器A在图G中从Pr可达，则它给parent变量赋值。当A为根时，它在算法第4行给parent赋值。

当A不为根时，对A到Pr的最短路径的跳数用数学归纳法：

跳数为1时，Pr在第三行给它的所有邻居发消息，处理器A在算法第5行会收到消息，并在第7行给parent赋值；

假设跳数为n-1的处理器都在第7行给parent变量赋过值，则由于是容许执行，它会在第9行给除parent以外的所有邻居发消息。跳数为n的处理器A至少有一个跳数为n-1的邻居，且该邻居的parent不是A（否则A的跳数<=n-2），故能在第5行收到消息并在第7行给parent赋值。

（2）充分性：处理器A曾设置过自己的parent变量，则A在图G中从Pr可达。当A为根时，A显然从自身可达。

当A不为根时，其只可能在第7行给parent赋值，由于是容许执行，第5行也必然执行过了，也就是A收到过消息M。由于消息M只从Pr发送出来，从Pr到A之间可达。

2.3 证明Alg2.3构造一棵以Pr为根的DFS树。

证明：首先证明引理：Alg2.3执行过程中，恰有一条消息在图中传递。

证：初始状态只有根节点发送一条消息；由算法，任一非根节点收到一条消息后，都恰好向一个节点发出一条消息。根节点收到一条消息时，或者恰好发出一条消息，或者算法终止。因此图中不可能存在两条消息在同时传递。

其次证明Alg2.3的有穷性，即Alg2.3必定能够终止。图中的每个节点至多向其每个相邻节点发送一次M消息；每个节点至多向其相邻节点发送一条parent消息或reject消息（parent和reject不会同时出现）。设图中有m条边，每条边上两个方向至多各有两条消息，故消息总数至多为4m。由引理，这些消息发送时间至多为4t，因此算法能够在有限时间内终止。

回到原命题。若证生成的是DFS树，只需证其连通性、无环性、深度优先性。

（1）连通性。假设存在相邻的节点A，B，算法结束时，A在DFS树中，B不在DFS树。由于A、B相邻，B初始时在A的unexplored集合里。由算法的有穷性，A向任意邻居C发出任意一条M消息，都会从C收到parent或reject消息，触发算法第17、25行；A收到M消息时会触发算法第11、14行。故算法结束时，unexplored集合为空，其中必有一次从unexplored集合中取出的是B，在14或25行向B发出M消息。B收到M消息后应当加入DFS树，矛盾。

（2）无环性。设生成的图中有一个环P1, P2, … Pi, P1。不失一般性，设P1是该环中最早接收到M消息的节点，且其M消息首先传给了P2。则P1是P2的parent……Pi是P1的parent。由于Pi是P1的parent，故Pi必向P1发送了M消息。由于在此之前，P1已经收到过M消息并将该消息传递给P2，根据算法Pi应当拒绝该M消息，矛盾。

（3）深度优先性。只需证明在子节点与兄弟节点都未访问时，先加入DFS树的一定是子节点。由引理知，任意时刻只有一个节点活跃，在向其一个邻居发送消息。由算法知，节点会向非parent的一个邻居节点（即一个子节点）发送消息，此时兄弟节点不可能接收到消息。

2.4 证明Alg2.3的时间复杂性为O(m)。

证明：（1）同步模型

由2.3题的引理知，每一轮中恰有一条消息在传输。又由2.3题的算法有穷性证明知，对m条边的图，至多发送4m条消息，故时间复杂度为O(4m)=O(m)。

（2）异步模型

由于每个时刻恰有一条消息在传输，时间复杂度与消息复杂度一致，均为O(4m)=O(m)。

2.5 修改Alg2.3获得一新算法，使构造DFS树的时间复杂性为O(n)。

解：当A给B发送消息M的时候，A向除A的parent和B以外的邻居发送消息X。每个节点维护一个收到消息X的邻居节点集合，以后不向集合内的这些元素发送M。

这样消息M只会在生成树的边上传递。不然，假设消息M在生成树以外的边B=>A上传递了，也就是发送消息时（时刻记为T）A、B都已经在生成树上。由于B=>A在生成树以外，故A不是B的parent，也就是必存在结点C，A是C的parent。A给C在T时刻之前发送消息M时，也会给B发送消息X。因此B在T时刻前收到了来自A的消息X，又在T时刻向A发送消息M，矛盾。

故新算法的消息复杂度为O(n)，由于时间复杂度与消息复杂度一致（2.4题），时间复杂度为O(n)。

3.1 证明同步环系统中不存在匿名的、一致的领导者选举算法。

证明：用数学归纳法证明每轮各处理器的状态机相同。

初始状态：在匿名系统中，每个处理器的初始状态机相同。

设在第n轮各处理器的状态相同，则第n+1轮每个处理器均发出相同的message。由于环拓扑结构的对称性，第n+1轮每个处理器均接收到相同的message。由于是匿名系统，从相同状态、相同message转移到的下一状态也相同。故第n+1轮各处理器的状态相同。

因此，匿名同步环系统中要么所有处理器同时成为leader，要么所有处理器都不是leader，不存在选出1个leader的算法。

注：上述证明中没有用到一致性条件。

3.2 证明异步环系统中不存在匿名的领导者选举算法。

这个结论我认为是不正确的。

假定异步环系统中消息处理延迟忽略不计，且各边上消息传输的延迟固定不变、两两不等。

初始状态：每个处理器初始化其timer为0，并向左边发一条消息M。

处理器收到消息M时，将收到消息M的时间戳作为自己的ID。这时，每个处理器的ID就是其右侧的边的消息传输延迟。也就是每个处理器有了唯一ID（尽管不是同时拥有唯一ID），结束匿名状态。

之后可以仿照非匿名环的leader选举算法。只要收到的id比自己大，就一直向左传。如果自己的id转了一圈回来了，就说明自己的id是最大的，自我选举为leader。显然，该算法能够在有限时间内结束，对n个节点的环，消息复杂度为O(n\*n)。

完整的异步匿名环上leader选举算法：

 1 var id = null;

 2 var leader = false;

 3 begin {

 4 StartTimer();

 5 send\_left(PROBE);

 6 }

 7 upon receive PROBE {

 8 id = CurrTimer();

 9 send\_left(TIME, id);

 10 }

 11 upon receive TIME {

 12 if (TIME.id > id)

 13 send\_left(TIME, id);

 14 else if (TIME.id == id)

 15 leader = true;

 16 }

上课的时候讲的是“每个处理器尽管接收消息的时间可能不同，接收到的消息序列也相同，因此最终的状态一定相同”，这点一直存在疑问。即使不允许使用定时器（环上的节点不能知道自己收到消息的时间），也可以根据左右两侧收到消息的先后顺序来判断。

例如，下列简单算法（不是leader选举算法）：

 1 var win = null;

 2 begin {

 3 send\_left(M);

 4 send\_right(M);

 5 }

 6 upon receive M {

 7 if (receive from left) {

 8 win = true;

 9 }

 10 if (receive from right) {

 11 win = false;

 12 }

 13 halt;

 14 }

对4个节点的异步环，设这四个节点为1、2、3、4（为描述方便，算法是匿名的），12、23、34、41之间的消息延迟分别为0.1, 0.2, 0.3, 0.2。

上述算法的意思是每个节点向两侧发消息；如果左侧先收到就赢，如果右侧先收到就输。上述四个节点中，2、3会输，4、1会赢，而且都进入停机状态。也就是最终4个处理器的状态是不一致的。

也许可以根据左右两侧消息收到的先后，构造出不依赖定时器的异步匿名环leader选举算法。

例如下面的算法可以使得有且仅有一个节点最后die=false，其他都是die=true。但算法无法终止（我还想不出来如何让这个最后没死的节点知道自己是leader）。（约定左右两边同时收到消息时，右边来的消息先处理）

 1 var serial = 0;

 2 var die = false;

 3 begin {

 4 send\_left (M, serial=0);

 5 send\_right(M, serial=0);

 6 }

 7 upon receive M {

 8 if (M.serial < serial)

 9 ignore;

 10 else if (M.serial == serial) {

 11 if (receive from left)

 12 die = true;

 13 else if (receive from right && not die) {

 14 ++serial;

 15 send\_left (M, serial);

 16 send\_right(M, serial);

 17 }

 18 }

 19 else { // M.serial > serial, forward message

 20 if (receive from left)

 21 send\_right(M, M.serial);

 22 else

 23 send\_left (M, M.serial);

 24 }

 25 }

3.9 若将环Rrev划分为长度为j(j是2的方幂)的连续片段，则所有这些片段是次序等价的。

证明：对一个整数P(0≤P≤n−1),可以表示为：

$$P=\sum\_{i=1}^{m}a\_{i}∙2^{i-1}$$

其中m=lg n,则有rev(P)=$\sum\_{i=1}^{m}a\_{i}∙2^{m-1}$。

设P、Q在同一个片段上，P1、Q1在同一片段上，且设这两个片段时相邻的，由模运算的加法可得：P1=P+ l；Q1=Q + l。l表示片段的长度，l=2k。

又因为：

$$Q=\sum\_{i=1}^{m}b\_{i}∙2^{i-1}$$

且P、Q在同一个片段上，有

|P-Q|<l=2k

所以存在r(0≤r≤k),满足 ar ≠ br。否则，|P−Q|≥l。这与P、Q在同一个片段上矛盾。

设s=min⁡{r},则根据rev(P),rev(Q)的表示方法可得：

sign(rev(P)-rev(Q))=sign (as-bs)

而

$$P1=P+1=\sum\_{i=1}^{m}a\_{i}∙2^{i-1}+2^{k}$$

$$Q1=Q+1=\sum\_{i=1}^{m}b\_{i}∙2^{i-1}+2^{k}$$

显然，P与P1的前k位相同，Q与Q1的前k位相同。由0≤s≤k得

sign(rev(P1)-rev(Q1))=sign (as-bs)

这两个相邻片段是序等价的，根据等价的传递关系，可得所有的片段都是次序等价。