第二部分　分布式算法

第三次课

中国科学技术大学计算机系
国家高性能计算中心（合肥）

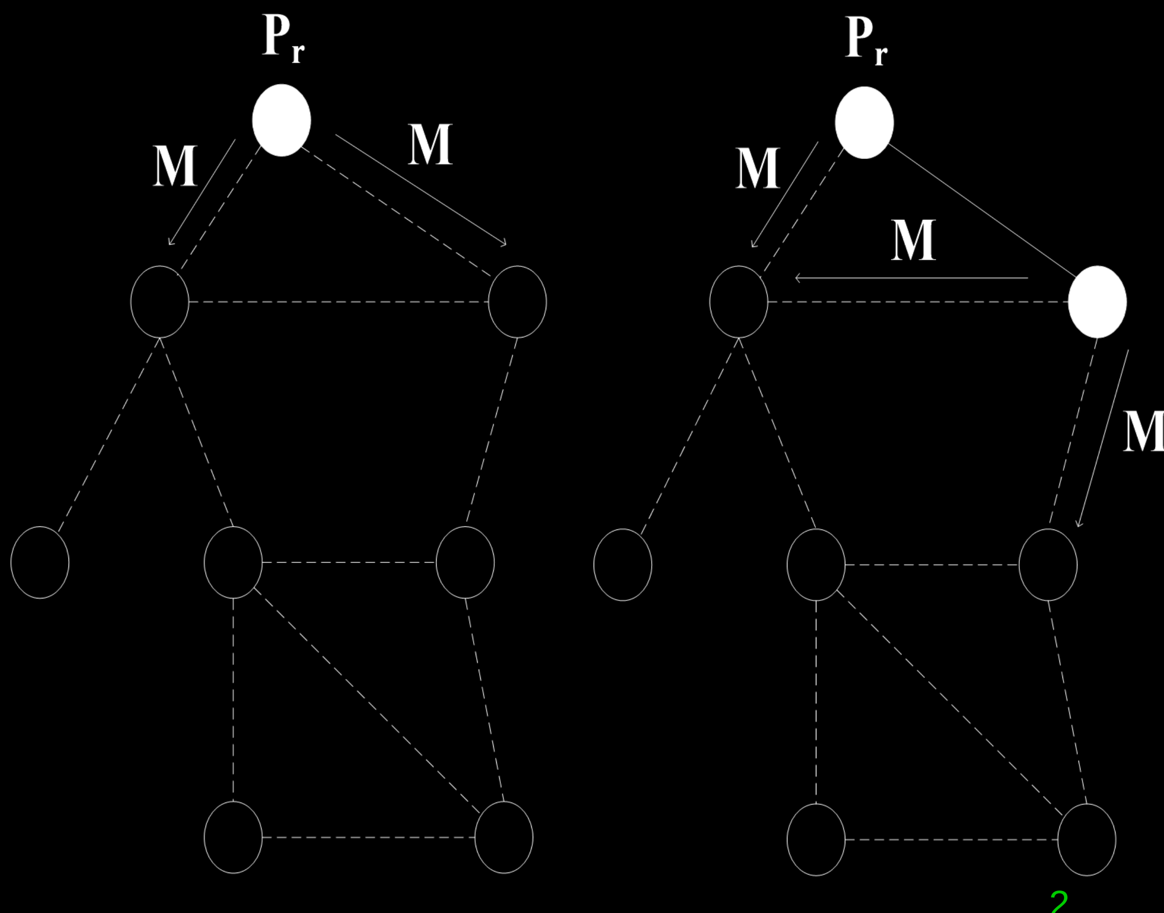
§ 2.3 构造生成树

上节算法均假设通信网的生成树已知。本节介绍生成树的构造问题。

1.Flooding算法(淹没)

- **算法**

  设 p_r 是特殊处理器。从 p_r 开始，发送M到其所有邻居。当 p_i 第1次收到消息M（不妨设此msg来自于邻居 p_j）时，p_i 发送M到除 p_j 外的所有邻居。



(a)　　(b)

§ 2.3 构造生成树

- **msg复杂性**
  因为每个结点在任一信道上发送M不会多于1次，所以每个信道上M至多被发送两次(使用该信道的每个处理器各1次)。
  在最坏情况下：M除第1次接收的那些信道外，所有其他信道上M被传送2次。因此，有可能要发送2m-(n-1)个msgs。这里m是系统中信道总数，它可能多达n(n-1)/2。
- **时间复杂性**：O(D)　D—网直径

2.构造生成树

对于flooding稍事修改即可得到求生成树的方法。

§ 2.3 构造生成树

①基本思想

- 首先，p_r发送M给所有邻居，p_r为根
- 当p_i从某邻居p_j收到的M是第1个来自邻居的msg时，p_j是p_i的双亲；若p_i首次收到的M同时来自多个邻居，则用一个comp事件处理自上一comp事件以来的所有已收到的msgs，故此时，p_i可在这些邻居中任选一个邻居p_j做双亲。
- 当p_i确定双亲是p_j时，发送<parent>给p_j，并向此后收到发来M的处理器发送<reject>msg

§ 2.3 构造生成树

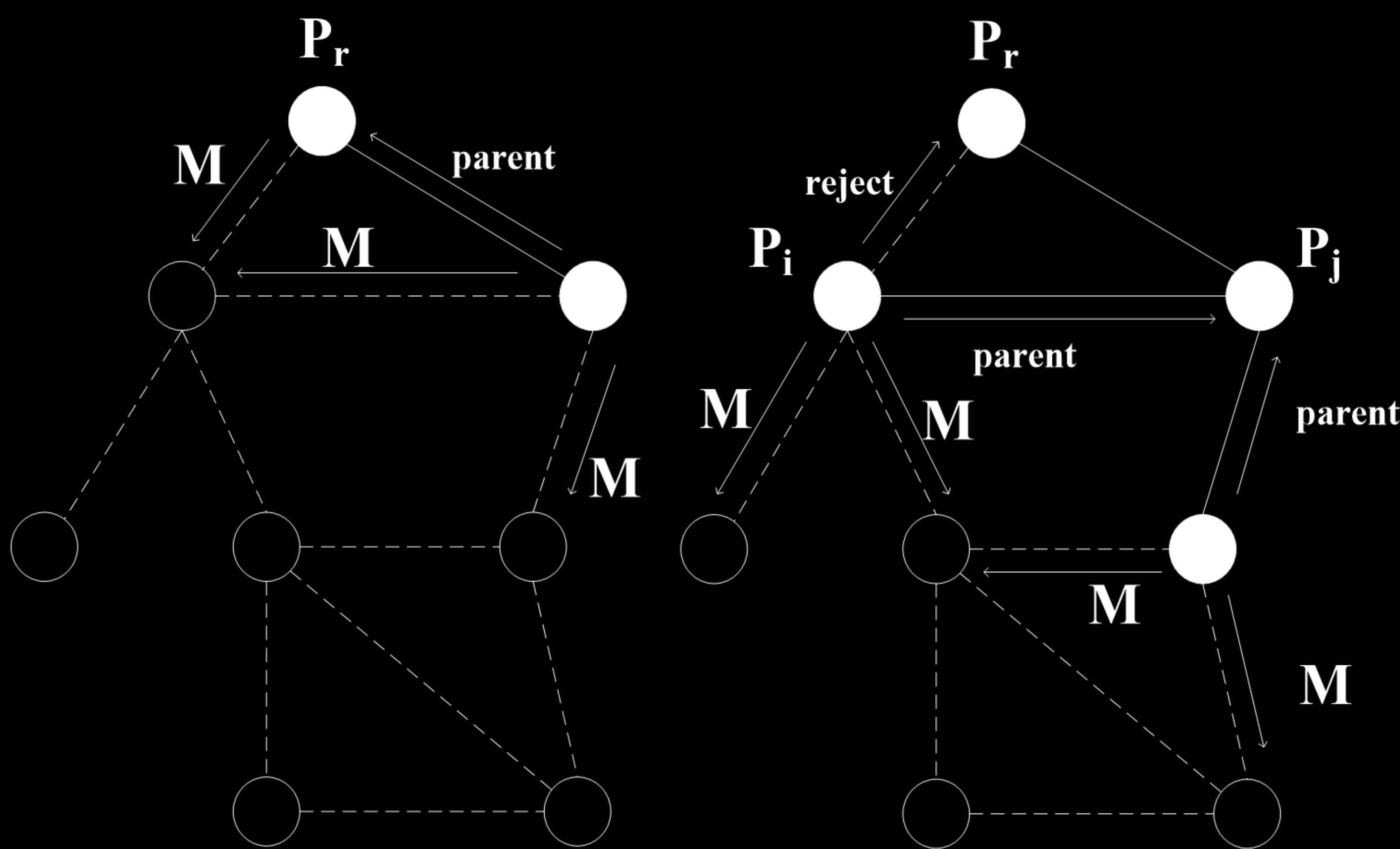
①基本思想

- 因为p_i收到p_j的M是第1个M，就不可能已收到其他结点的M，当然可能同时收到(说明p_i与这些邻居间不是父子关系，或说它们不是生成树中的边)；同时p_i将M转发给其余邻居，这些邻居尚未发M给p_i，或虽然已发M给p_i，但p_i尚未收到。
- p_i向那些尚未发M给p_i(或已发M但尚未到达p_i)的邻居转发M之后，等待这些邻居发回响应msg：<parent>或<reject>。那些回应<parent>的邻居是p_i的孩子。
- 当 p_i 发 出 M 的 所 有 接 收 者 均 已 回 应 (<parent> 或 <reject>)，则p_i终止。将parent和children边保留即为生成树。

§ 2.3 构造生成树

②图示

p_i若认为p_j是其双亲，则p_i向p_r发出M，而p_r仍会向p_i发reject，但因为此前p_r向p_i发出过M，故p_i收到M时仍会向p_r发reject。(可以改进？)



§2.3 构造生成树

③算法： Alg2.2 构造生成树（code for p_i $0 \le i \le n-1$）

初值：parent=nil；集合children和other均为φ

upon receiving no message:
    if i=r and parent=nil then { //根尚未发送M
        send M to all neighbors;
        parent:=i;} //根的双亲置为自己
upon receiving M from neighbor p_j:
    if parent=nil then {//p_i此前未收到过M，M是p_i收到的第1个msg
        parent:=j;
        send <parent> to p_j; //p_j是p_i的双亲
        send M to all neighbors except p_j;
    }else //p_j不可能是p_i的双亲，p_i收到的M不是第1个msg
        send<reject> to p_j;
upon receiving <parent> from neighbor p_j:
    children:=children∪{ j }; //p_j是p_i的孩子，将j加入孩子集
    if children∪other 包含了除parent外的所有邻居 then terminate;
upon receiving <reject> from neighbor p_j:
    other:=other∪{ j }; //将j加入other，通过非树边发送的msg。
    if children∪other包含了除p_i的双亲外的所有邻居 then terminate

§2.3 构造生成树

④分析

Lemma2.6 在异步模型的每个容许执行中，算法2.2构造一棵根为p_r的生成树。(正确性)

Pf：算法代码告诉我们两个重要事实

a) 一旦处理器设置了parent变量，它绝不改变，即它只有一个双亲

b) 处理器的孩子集合决不会减小。

因此，最终由parent和children确定的图结构G是静止的，且parent和children变量在不同结点上是一致的，即若p_j是p_i的孩子，则p_i是p_j的双亲。

下述证明结果图G是根为p_r的有向生成树。

§2.3 构造生成树

- **为何从根能到达每一结点？(连通)**

  反证：假设某结点在G中从p_r不可达。因网络是连通的，若存在两个相邻的结点p_i和p_j使得p_j在G中是从p_r可达的(以下简称p_j可达)，但p_i不可达。因为G里一结点从p_r可达当且仅当它曾设置过自己的parent变量(Ex 证明)，所以p_i的parent变量在整个执行中仍为nil，而p_j在某点上已设置过自己的parent变量，于是p_j发送M到p_i(line9)，因该执行是容许的，此msg必定最终被p_i接收，使p_i将自己的parent变量设置为j。矛盾！

§ 2.3 构造生成树

- **为何无环？(无环)**

  假设有一环，$p_{i1},\ldots p_{ik}p_{i1}$，若$p_i$是$p_j$的孩子，则$p_i$在$p_j$第1次收到M之后第1次收到M。因每个处理器在该环上是下一处理器的双亲，这就意味着p_{i1}在p_{i1}第1次接收M之前第1次接收M。矛盾！

- **复杂性**

  显然，此方法与淹没算法相比，增加了msg复杂性，但只是一个常数因子。在异步通信模型里，易看到在时刻t，消息M到达所有与p_r距离小于等于t的结点。因此有：

Th2.7 对于具有m条边和直径D的网络，给定一特殊结点，存在一个msg复杂性为O(m)，时间复杂性为O(D)的异步算法找到该网络的一棵生成树。

§ 2.3 构造生成树

Alg2.2在同步模型下仍可工作。其分析类似于异步情形。然而，与异步不同的是，它所构造的生成树一定是一棵广度优先搜索(BFS)树。

Lemma2.8 在同步模型下，Alg2.2的每次容许执行均构造一棵根为p_r的BFS树。

Pf：对轮t进行归纳。即要证明：在第t轮开始时刻
①根据parent变量构造的图G是一棵包括所有与p_r距离至多为t-1结点的BFS树；
②而传输中的消息M仅来自于与p_r距离恰为t-1的结点。
由此构造的树是一棵根为p_r的BFS

§ 2.3 构造生成树

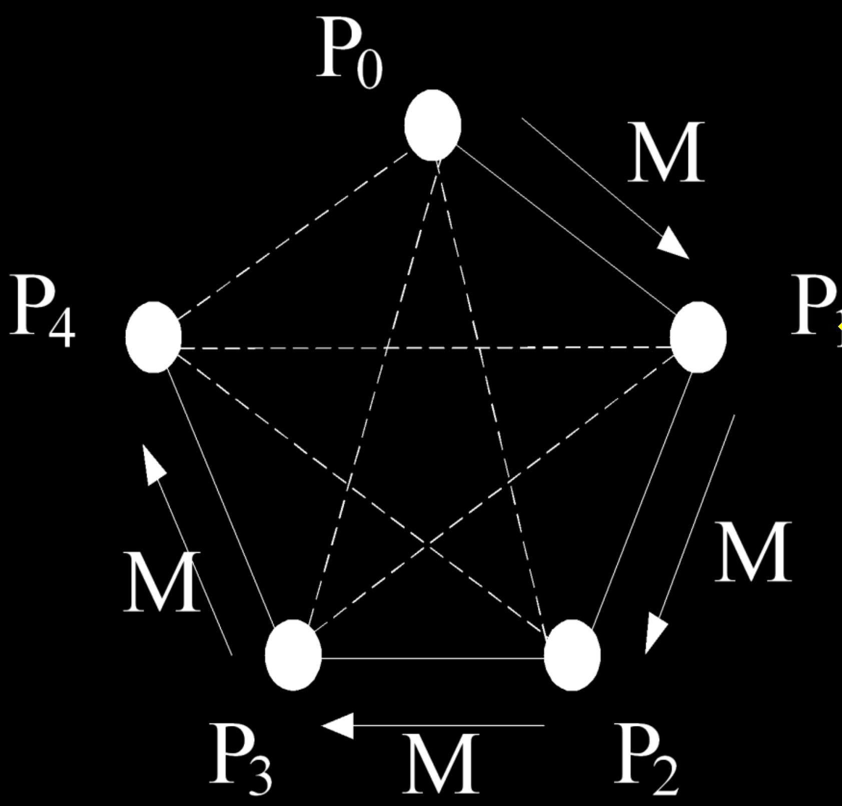
当t=1时，所有parent初值为nil，M从p_r传出。

假设引理对第t-1≥1轮为真，在该轮里，从距离 t-2的结点传出的M被接收，任何接收M的结点与p_r的距离不超过t-1(恰为t-1或更短)，那些parent值非空的接收结点显然与p_r的距离不超过t-2，他们既不改变parent的值也不转发M；而与p_r距离为t-1的结点在t-1轮里收到M，因为它们的parent为nil，故将其置为合适的双亲并转发M。距离p_r大于t-1的结点不会收到M，因此也不会转发M。因此有如下定理：

Th2.9 对于具有m条边直径为D的网络，给定一个特殊结点，存在一个同步算法在msg复杂性为O(m)，时间复杂性为O(D)内找到一棵BFS树。

§ 2.3 构造生成树

- **异步系统里，Alg2.2能构造BFS树？**
  例如，考虑5个顶点的完全连通图



- P_0为根，假定M消息按P_0到p_1，P_1到P_2，P_2到P_3，P_3到P_4的次序快速传播，而M在其它路径上传播较慢。结果生成树是从P_0到P_4的链，它不是BFS树
- 虽然此图直径D=1，生成树的高度d=4，但是算法的运行时间仍然为O(D)而不是O(d)。理解：P_0到P_4的M在1个时间内到达，即P_0->P_1->P_2->P_3->P_4的时间之和不超过1。

§ 2.3 构造生成树

■ 信息的请求和收集

将算法2.2(求生成树)和汇集算法组合即可完成。组合算法的时间复杂性在同步和异步模型中不同,设网是完全图

❖ 求生成树算法：同步和异步均为 $\begin{cases}\text{消息复杂性}O(m)\\ \text{时间复杂性}O(D)\end{cases}$

❖ 汇集算法：同步和异步均有 $\begin{cases}\text{msg } n-1\\ \text{time } d \text{ //生成树高}\end{cases}$

❖ 组合算法

①同步：组合算法的msg复杂性O(m+n)；BFS树中，d=1， d≤D，故时间复杂性O(D+d)=O(D)=O(1)。

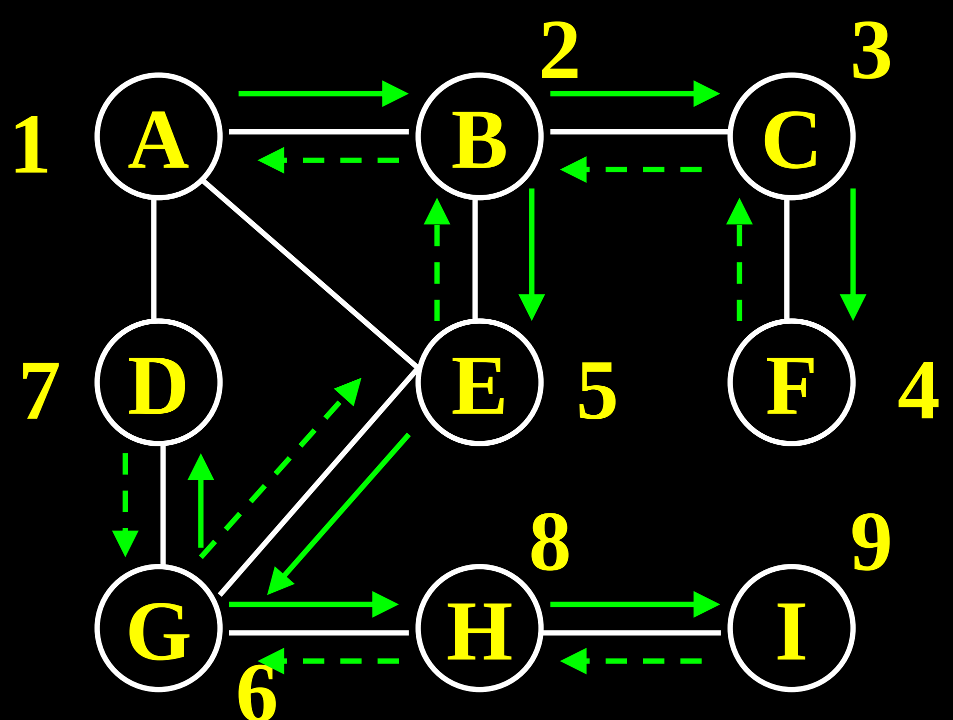
②异步：组合算法的msg复杂性O(m+n)；生成树高d=n-1，所以时间复杂性O(D+d)=O(d)=O(n)。

1-time复杂性的组合算法T(n)=O(D)。

§ 2.4 构造DFS生成树(指定根)

回忆无向图的深度优先搜索问题：

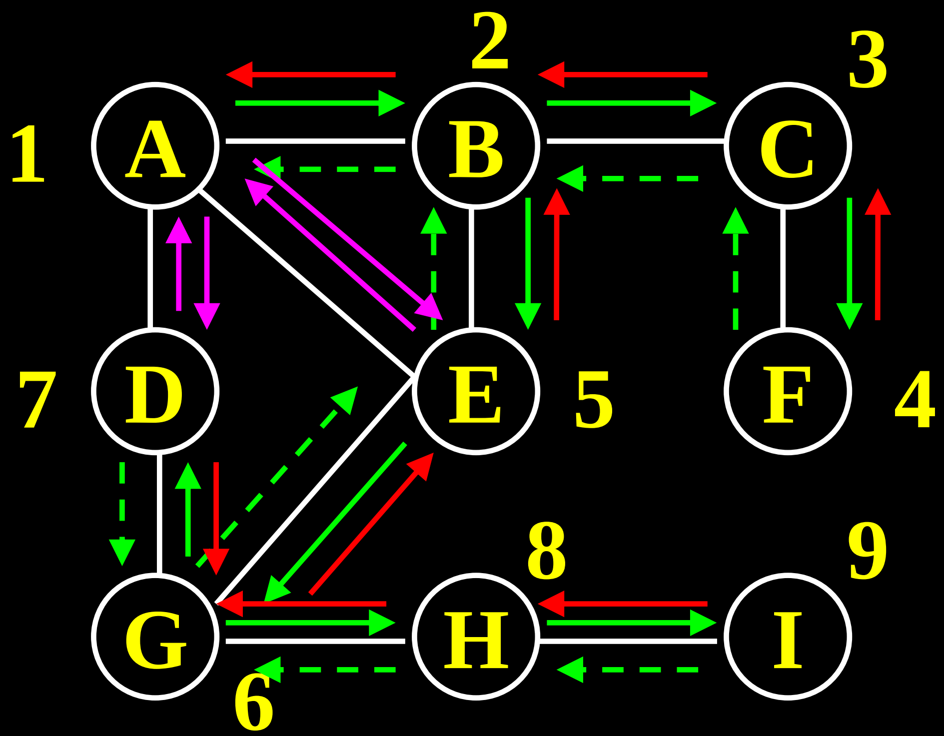
- **方法：**
  - 从任意顶点开始访问，例如A
  - 然后访问它的一个没有访问过的邻接点，例如B
  - 若无未访问过的邻接点,则后退寻找，直至全部被访问为止



§ 2.4 构造DFS生成树(指定根)

基本思想：

- 设定P_r为指定的根节点，P_r从还未向其发送消息<m>的邻接节点中任选一个节点发送消息<m>。
- 当P_i从P_j收到的消息<m>是第一个来自于邻接节点的消息时，P_j为P_i的双亲，向其发送<parent>消息，并向此后向自己发送消息的邻居发送<reject>消息。
- P_i从还未发送过消息的邻居中任选一个，发送消息<m>，然后等待<parent>或者<reject>消息，并将回应<parent>的节点加到自己的孩子集合中。
- 当P_i向所有的邻居都转发过消息后，P_i终止。



§ 2.4 构造DFS生成树(指定根)

构造DFS树时每次加一个结点，而不像Alg2.2那样，试图在树上同时增加同一层的所有结点。

Alg2.3 构造DFS生成树，为P_r为根

Code for processor P_i, $0 \le i \le n-1$

```
var     parent:        init nil;
        children:      init φ ;
        unexplored: init all the neighbors of P_i
                                    //未访问过的邻居集
1: upon receiving no msg:
2:    if (i=r) and (parent=nil) then {  //当P_i为根且未发送M时
3:       parent := i;  //将parent置为自身的标号
4:       ∀ P_j ∈ unexplored;
5:       将P_j从unexplored中删去; //若P_r是孤立结点,4-6应稍作修改
6:       send M to P_j;
     }//endif
```

§ 2.4 构造DFS生成树(指定根)

upon receiving M from neighbor P_j:

    if parent=nil then { //P_i此前未收到M

        parent := j; //P_j是P_i的父亲

        从unexplored中删P_j

        if unexplored ≠ φ then {

            $\forall P_k \in$ unexplored;

            将P_k从unexplored中删去;

            send M to P_k;

        } else send <parent> to parent;

                //当P_i的邻居均已访问过，返回到父亲

    }else send <reject> to P_j; //当P_i已访问过时

§ 2.4 构造DFS生成树(指定根)

upon receiving <parent> or <reject> from neighbor P_j:

if received <parent> then add j to children;

//P_j是P_i的孩子

if unexplored = ϕ then { //P_i的邻居均已访问

if parent $\neq$ i then send <parent> to parent;

//P_i非根，返回至双亲

terminate; //以P_i为根的DFS子树已构造好！

}else { //选择P_i的未访问过的邻居访问之

$\forall P_k \in$ unexplored;

将P_k从unexplored中删去；

send M to P_k;

}

§2.4 构造DFS生成树(指定根)

引理2.10 在异步模型里的每个容许执行，Alg2.3构造一棵以P_r为根的DFS树。证明留作练习。

Th2.11 对于一个具有m条边，n个结点的网络，以及给定的特殊顶点，存在一个时间复杂性和消息复杂性均为O(m)的异步算法找到一棵DFS树。

Pf：每个结点在其邻接边上至多发送M一次，每个结点至多生成一个msg(<reject>或<parent>)作为对每个邻接边上收到的M的响应。因此Alg2.3至多发送4m个消息(其实大部分没有4倍)，即算法的msg复杂性为O(m)。

时间复杂性证明留作练习。

如何改进使msg的复杂性不是4m？

注意：上述算法msg复杂性较好，但时间复杂性太差。可降至O(n)。

§2.5 不指定根时构造DFS生成树

算法2.2和2.3构造连通网络的生成树时，必需存在一个特殊的结点作为启动者(Leader)。当这样的特殊结点不存在时，如何构造网络的一棵生成树？但本节算法须假定：各结点的标识符唯一，不妨设是自然数，§3.2仍需此假定。

不指定根构造DFS生成树，和后面的领导者选举问题一样，都是破对称问题。

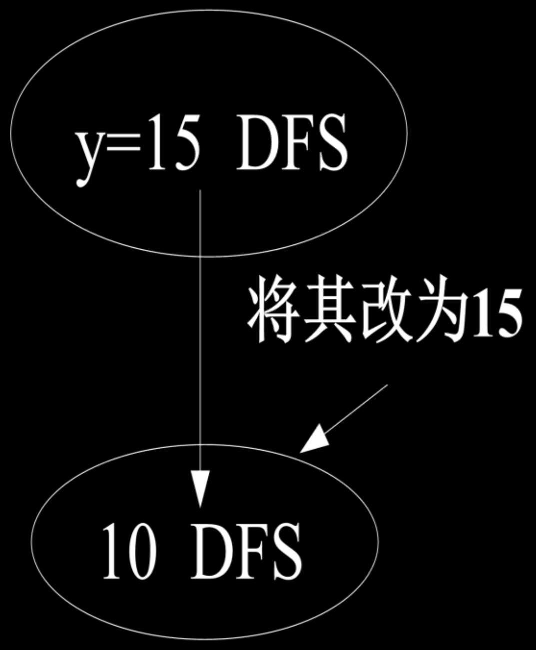
1. 基本思想

- 每个结点均可自发唤醒，试图构造一棵以自己为根的DFS生成树。若两棵DFS树试图链接同一节点(未必同时)时，该节点将加入根的id较大的DFS树。
- 为了实现上述思想，须做：
  - 每个结点设置一个leader变量，其初值为0，当P_i唤醒自己时，$leader_i=id_i$；

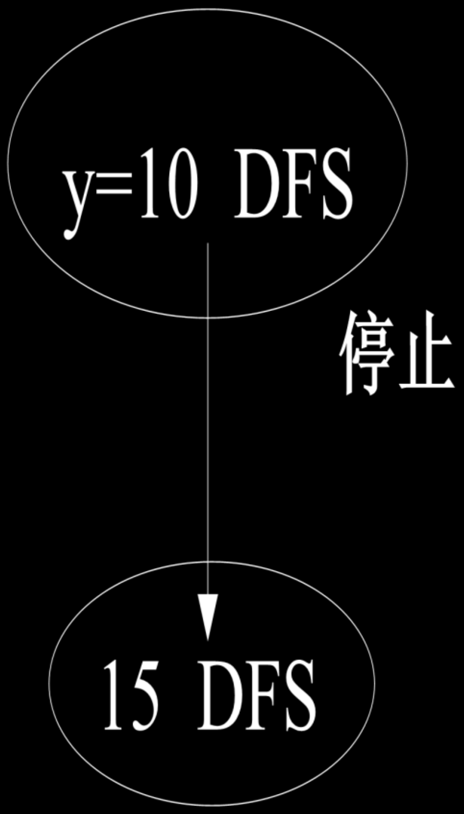
§ 2.5 不指定根时构造DFS生成树

- 当一结点自发唤醒时，它将自己的id(leader)发送给某一邻居；
- 当一结点P_i收到来自邻居P_j的标识符y时，P_i比较y和$leader_i$：

①若$y>leader_i$，则y可能是具有最大标识符结点的DFS子树的标记。因此，将$leader_i$置为y，并令P_j是P_i的双亲。从P_i开始继续生成标记为y的DFS树。Note：要修改原P_i所在的DFS子树中所有结点的leader。



§ 2.5 不指定根时构造DFS生成树



② 若y<$leader_i$，则标记为y的DFS树中最大id(y)小于目前所看到的最大标识符。此时无须发送msg，停止构造标记为y的DFS。等待最终某个更大的id的leader消息到达标记为y的树中结点时，再将该节点连接到树中。(至少标记为$leader_i$的msg会到达标记为y的树)

③ 若y=$leader_i$，则P_i已属于标记y的DFS树中。

§2.5 不指定根时构造DFS生成树

2. 算法

Alg2.4 构造生成树，不指定根

Code for Processor P_i $0 \le i \le n-1$

Var parent: init nil;
leader: init 0;
children: int φ;
unexplored: init all neighbors of P_i;

1: upon receiving no msg: //wake up spontaneously
2: if parent = nil then {
//若非空，则P_i在某子树上，则P_i失去竞选机会
3: leader := id; parent := i;//试图以自己为根构造树
4: $\forall P_j \in$ unexplored;
5: 将P_j从unexplored中删去；
6：send <leader> to p_j;
}

§ 2.5 不指定根时构造DFS生成树

想像：有m个人竞选领袖，id是他自身的素质分，不想竞争人的 id不参与比较。

竞争规则：将自己的id(如讲演片)传递给一个熟悉的人，由他再传给另一人(一次只能送一人。)

upon receiving <new-id> from neighbor P_j:

    if leader<new-id then {   //将P_i所在树合并到P_j所在树中
        leader := new-id; parent := j;
        //令P_i的双亲为P_j，可能是修改，而非对nil赋值
        //并不一定能停止较差的竞选者传播msg
        unexplored := all the neighbors of P_i except P_j;
        //重置未访问的邻居集
        if unexplored ≠φ then {
            //因为new-id大，使原P_i所在DFS树修改各自id
                $P_k \in$ unexplored;
            将P_k从unexplored中删去；

§ 2.5 不指定根时构造DFS生成树

```
          send <leader> to P_K;
      }else send <parent> to parent; // unexplored =φ
   }else // leader≥new-id
     if leader=new-id then send <already> to P_j;
   //表示自己已经传出过此录像带，无需重传。已在同一树中
   //若leader>new-id，则new-id所在DFS停止构造
   //以前收到的竞选者优于new-id，不传送，使之停止传播。

upon receiving <parent> or <already> from neighbor P_j:
     if received <parent> then add j to children;
     if unexplored=φ then {  //无尚未访问的邻居
         if parent≠i then send <parent> to parent  //返回
         else terminates as root of the DFS tree;  //根终止
     }else {  //有尚未访问的邻居
```

§ 2.5 不指定根时构造DFS生成树

$\forall P_k \in$ unexplored;
将P_k从unexplored中删去;
send <leader> to P_k;
}

3. 分析:

- 只有生成树的根显式地终止，其它结点没有终止，始终在等待msg。但可修改此算法，使用Alg2.1从根结点发送终止msg
- 正确性

  该算法比前面的算法更复杂，这里只给出粗略的证明。

  设P_m是所有自发唤醒结点中标识符最大者，其标识符为id_m。消息id_m总是被传播，而一旦一个结点收到id_m，则该节点(P_m除外)上所有msgs被忽略。因为消息id_m的处理和Alg2.3求DFS树一致，因此产生的parent和children变量的设置是正确的。因此有:

§2.5 不指定根时构造DFS生成树

Lemma2.12 设P_m是所有自发唤醒结点中具有最大标识符的结点。在异步模型的每次容许执行里，算法2.4构造根为P_m的一棵DFS树。

Note：因为在容许执行中，网络里的所有自发唤醒结点中最大标识符结点最终会自发启动，故建立的DFS树的根是P_m

可通过广播算法从P_m发出终止msg，即使不广播，所有非P_m结点最终也会因为收到P_m的标识符而停止。因此，不可能构造一棵根不是P_m的生成树。

Lemma2.13 在异步模型的每个容许执行里，只有一个处理器终止作为一生成树的根。

§ 2.5 不指定根时构造DFS生成树

- 复杂性

定理：对于一个具有m条边和n个节点的网络，自发启动的节点共有p个，其中ID值最大者的启动时间为t，则算法的消息复杂度为O(pn^2)，时间复杂度为O(t+m)。

消息复杂性：简单地分析，最坏情况下，每个处理器均试图以自己为根构造一棵DFS树。因此，Alg2.4的msg复杂性至多是Alg2.3的n倍：O(m*n)

时间复杂性：类似于Alg2.3的msg复杂性O(m)。

§2.5 不指定根时构造DFS生成树

Ex.

2.1 分析在同步和异步模型下，convergecast算法的时间复杂性。

2.2 证明在引理2.6中，一个处理器在图G中是从P_r可达的，当且仅当它的parent变量曾被赋过值

2.3 证明Alg2.3构造一棵以P_r为根的DFS树。

2.4 证明Alg2.3的时间复杂性为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)，并证明。

下次继续！