算法设计与分析

> SA25011049 李宇哲

T1 EX2.1

> 2.1 分析在同步和异步模型下，convergecast算法的时间复杂性。

同步模型的时间复杂度

假设树高为 h，节点数为n，同步算法需要 h round才能从叶子节点传到根节点，因此时间复杂度为 O(h)

异步模型的时间复杂度

最好情况 O(h)

每个非根节点发1条消息 = O(n)条 - 如果消息串行传递，每条消息延迟1单位时间 - 最坏时间 = O(n)，由于2关键路径长度为 h，因此最坏时间复杂度为 $O(nh)$

T2 EX2.3

> 2.3 证明Alg2.3构造一棵以Pr为根的DFS树。
>
> ```
> Code for processor Pi, 0≤i ≤ n-1
> var parent: init nil;
> children: init ϕ;
> unexplored: init all the neighbors of Pi
> //未访问过的邻居集
> 1: upon receiving no msg:
> 2: if (i=r) and (parent=nil) then { //当Pi为根且未发送M时
> 3: parent := i; //将parent置为自身的标号
> 4: Pj ∈ unexplored;
> 5: 将Pj从unexplored中删去; //若Pr是孤立结点,4-6应稍作修改
> 6: send M to Pj;
> }//endif
> 7: upon receiving M from neighbor Pj:
> 8: if parent=nil then { //Pi此前未收到M
> 9: parent := j; //Pj是Pi的父亲
> 10: 从unexplored中删Pj
> 11: if unexplored ≠ϕthen {
> 12: Pk ∈ unexplored;
> 13: 将Pk从unexplored中删去;
> 14: send M to Pk;
> 15: } else send <parent> to parent;
> //当Pi的邻居均已访问过，返回到父亲
> 16: }else send <reject> to Pj; //当Pi已访问过时
> 17: upon receiving <parent> or <reject> from neighbor Pj:
> 18: if received <parent> then add j to children;
> //Pj是Pi的孩子
> 19: if unexplored = ϕ then { //Pi的邻居均已访问
> 20: if parent ≠ i then send <parent> to parent;
> //Pi非根，返回至双亲
> 21: terminate; //以Pi为根的DFS子树已构造好!
> ```

```
22: }else { //选择Pi的未访问过的邻居访问之
23: Pk ∈ unexplored;
24: 将Pk从unexplored中删去;
25: send M to Pk;
}
```

要证明算法终止后成功构造了一颗以 P_r 为根的 DFS 生成树，需要证明：

所有结点连通并且无环，满足深度优先的性质，同时算法经过有限次迭代终止。

先证明连通性：

假设存在某节点 p_i在G中从p_r不可达，存在两个相邻结点 p_i和p_j，p_j是从p_r可达的，但p_i不可达。即p_j设置过parent变量，且$p_i \in unexplored(p_j)$，而 p_j每次向邻居发送消息会收到 <parent> or <reject>，直到 unexplored 为空，这意味着 p_i一定会收到从 p_j 发来的消息，使得 p_i 的 parent 不为nil，即p_i可达，矛盾。

所以算法终止时所有结点都是 p_r可达的

无环：类似引理证明。假设存在一个环，若 p_j 是 p_i的parent，在 p_j在 p_i第一次收到M之前就第一次收到M，重复这个过程n次，存在逻辑矛盾，没有第一个收到M的节点，因此不存在环。

上述证明了生成的是一个有向无环图，下面证明满足深度优先的性质。

由算法，每个节点依次选择一个未访问邻居，进行下一步搜索，并不会并行访问多个邻居。直到当前结点回溯收到<parent>后，才访问下一个邻居，这满足深度优先的性质。下面进行形式化证明：

若 T 是 网络 N 上以 p_r 为根的生成树。对于 T 上任意两点 x，y，如果边 (x,y)在网络 N 中，要么 x 是 y在 T 中祖先，要么 y 是 x 在 T 中的祖先，则 T 是 N 上以 p_r 为根的 DFS 树。

假设存在 两个不同节点 p_i, p_j，(p_i, p_j) 在网络 N 汇总，但两者不互为祖孙关系。不放假设 p_j 先向 parent 发送<parent>，若 p_i 没有接收过消息，则 p_i第一次接收消息要么沿(p_j, p_i)，要么从其他节点从 p_j转发，则 p_j是 p_i的祖先，与假设矛盾。

因而满足 DFS 性质。

T3 EX2.4

> 2.4 证明Alg2.3的时间复杂性为O(m)。

算法2.3存在性质，任意时刻系统中最多只有一个处理器在主动发送消息，因为一个节点之后再收到子结点的返回消息后才会探索下一个邻居。因此消息的因果顺序是完全串行的。

同步模型下：每一轮仅允许一个处理器发送消息，总过产生 $O(m)$条消息，因此需要 $O(m)$ 轮通信，则时间复杂性为 O(m)

异步模型下，任意时刻至多有一个处理器在发送消息，总共O(m)条消息直到需要 O(m)的时间顺序完成

T4 EX2.5

> 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。

新算法主要是要避免逐边重复通信，不是串行的逐个邻居访问。

先定义变量

```
parent[i]: 初始为 nil;
children[i]: 初始为空;
neighbors[i]：节点 i 的邻居集合;
expected_reply[i]: 节点i需要等待多少个子节点的回应
reply_count[i]: 节点i已经从多少个子节点收到done/reject消息
```

定义两种message：

- reject：表示这条边不是树边
- done：表示当前结点对应子树的DFS搜索结束。

```
// initial
if i == r then
    parent[i] := 1
    expected_reply[i] := size(neighbors[i])
    for each Pj in neighbors[i] do
        send(M) to P(j)
    if expected_reply[i] == 0 then
        terminate
// receive M from Pj
upon receive M from Pj do
    if parent[i] == nil then
        parent[i] := Pj
        remove Pj from neighbors[i]
        expected_reply[i] := size(neighbors[i])
        for each Pk in neighbors[i] do
            send M to Pk
        if expected_reply[i] == 0 then
            send done to parent[i]
     else
         send reject to Pj
// receive done from a child
upon receive done from Pj do
    add Pj to children[i]
    reply_count[i] := reply_count[i] + 1
    if reply_count[i] == expected[i] then
        if i != r then
            send done to parent[i]
        else
            terminate
//receive reject
upon receive recject from Pj do
    reply_count[i] := reply_count[i] + 1
    if reply_count[i] == expected_reply[i] then
        if i != r then
            send done to parent[i]
        else
            terminate
```

每个节点在第一次收到 ⟨M⟩ 消息时唯一确定父节点, 节点只向未访问的邻居广播 ⟨M⟩，确保不会形成环，最终所有节点被访问，各节点的 `parent` 指针组成一棵以根 rrr 为根的连通无环树。

每个节点收到 ⟨M⟩ 后立即向所有邻居并发发送消息；因此每一轮传播完成一层 DFS 树的扩展。DFS树高 $d \leq n - 1$，因此递归深度 $\leq O(n)$，因此时间复杂度为 O(n)