생성트리와 강결합요소의 갱신을 위한 분산알고리즘

박 정 호† · 박 윤 용† · 최 성 희†

요 약

생성트리와 같은 문제를 해결하는데 필요한 정보가 네트워크상의 프로세서에 분산되어 있는 상황에서 그들 정보를 교환하면서 그 문제를 해결하는 알고리즘을 분산알고리즘(Distributed Algorithm)이라고 한다.

생성트리와 강결합요소가 이미 구성되어 있는 비동기식 네트워크상에서 네트워크 형상이 변할 경우, 이로인해 구성되어 있던 생성트리와 강결합요소를 갱신해야 하는 경우가 발생한다. 본 논문에서는 이러한 경우 생성트리와 강결합요소를 효율적으로 갱신하는 메시지복잡도 O(n'log n' + (n' + s + t)), 이상시간복잡도 O(n'log n')의 분산 알고리즘을 제안한다. 여기서 n'는 토폴로지 변화후의 네트워크의 프로세서수, s는 추가 링크수를 나타낸다. 또 t는 삭제 링크를 포함하는 강결합요소에 포함되어 있는 전체 링크수를 나타낸다.

A Distributed Algorithm to Update Spanning Tree and Strongly-Connected Components

Jung-Ho Park† · Yoon-Yong Park† · Sung-Hee Choi†

ABSTRACT

Considers the problem to update the spanning tree and strongly-connected components in response to topology change of th network. This paper proposes a distributed algorithm that solves such a problem after several processors and links are added and deleted. Its message complexity and its ideal-time complexity are O(n'log n' + (n' + s + t)) and O(n'log n') respectively, where n' is the number of processors in the network after the topology change, s is the number of added links, and t is the total number of links in the strongly connected component (of the network before the topology change) including the deleted links.

1. 서 론

네트워크상에서 프로세서와 링크의 추가 및 삭제가 빈번히 발생함으로 인해 네트워크 형상은 일정하지 않고 동적으로 변화하는 것이라고 생각할 수 있다. 동적으로 네트워크 형상이 변화하는 네트워크 환경에 있어서는 그 토폴로지 정보를 하나의 프로세서가 통괄해서 일괄적으로 관리하는 것보다 각 프로세서가 자기에 관한 토폴로지 정보만을 분담해서 관리하는 것이 좋다. 이와 같이 어떤 문제를 해결하는데 필요한 정보가 네트워크상의 프로세서에 분산되어 있는 상황에서 그들 정보를 교환하면서 그 문제를 해결하는 알고리즘을 분산알고리즘(Distributed Algorithm)이라고 한다[1]~[8].

분산알고리즘이 실행되는 네트워크 환경에서 메시지수가 코스트의 대부분을 차지하고, 또 메시지의 전

※ 본 연구는 정보통신부에서 시행하는 대학기초연구 지원사업에서 일부를 지원 받았음.
† 종신회원 : 선문대학교 컴퓨터정보학부 교수
논문접수 : 1998년 10월 2일, 심사완료 : 1998년 12월 2일

송 시간이 분산알고리즘이 처리를 종료하는데 걸리는 시간의 대부분을 차지한다. 따라서, 분산알고리즘의 효율은 주로 메시지복잡도(message complexity)와 이상시간복잡도(ideal time complexity)로 평가된다. 메시지 복잡도란 분산알고리즘 실행중에 네트워크에서 교환되는 메시지의 총수를 말한다. 대부분의 네트워크 모델에서는 메시지의 전송시간에 관해서 유한이라는 것이외에는 아무 것도 가정하지 않는다(비동기식 네트워크). 그러므로 일반적으로 순차알고리즘의 평가에 적용되고 있는 시간복잡도(time complexity)의 개념은 적용할 수 없다. 따라서 분산알고리즘의 실행시간을 평가하는데 이상시간복잡도라는 것을 채택하고 있는데, 이상시간복잡도란 프로세서 내에서의 처리시간을 무시하고, 메시지가 링크를 통과하는 전송시간을 1단위시간으로 했을 때의 분산알고리즘이 종료할 때까지의 단위시간수를 말한다.

강결합요소(strongly-connected component)란 네트워크상에 존재할 수 있는 교착상태(deadlock)에 해당하며, 네트워크상에 교착상태가 발생하는 것은 바람직하지 못하고 교착상태가 발생한 경우에는 즉시 교착상태를 해결해야 한다. 따라서, 네트워크상에서 작업을 할 때에는 수시로 그 네트워크가 강결합인지 아니면 강결합요소가 네트워크상에 부분적으로 존재하는지를 조사하여 강결합요소가 존재한다는 것이 판명되었을 때에는 재빨리 교착상태를 해결해서 작업을 수행하는데 차질이 없게 해 주어야 한다. 이러한 경우에 강결합요소를 구하는 분산알고리즘이 매우 유효하게 사용된다.

네트워크의 토폴로지는 동적으로 변화하기 때문에, 생성트리(spanning tree) 구성문제라든가, 강결합요소 문제를 포함하여 각종 문제를 해결했다고 하더라도 네트워크의 변화에 따라 이들 문제를 다시 해결해야 하는 경우가 발생한다. 이와 같이 네트워크의 변화에 따라 다시 해결하는 문제를 갱신 문제(updating problem)라고 한다. 즉, 임의의 네트워크에 대한 생성트리가 구성되어 있는 상황에서 네트워크가 변화했을 때, 생성트리를 다시 구성하는 문제를 생성트리 갱신 문제(spanning tree updating problem)라고 한다.

갱신문제는 네트워크의 변화에 따라 문제를 다시 해결한다는 점을 고려하지 않고, 변화후의 새로운 네트워크에 대해 처음부터 그 문제를 해결하는 기존의 알고리즘을 적용시키더라도 해결할 수 있다. 즉, 문헌 [5]에 제한된 알고리즘을 새로운 네트워크에 적용할 경우, 생성트리 갱신 문제를 해결할 수 있다. 그러나, 갱신문제의 경우, 각 프로세서는 토폴로지 변화전의 네트워크에 대한 답을 가지고 있으므로 각 프로세서가 가지고 있는 여러 가지 정보를 이용하여 갱신문제를 효율적으로 해결할 수 있으며, 이들 정보를 이용한 알고리즘이 제안되었다[6].

생성트리와 강결합요소갱신문제를 해결하기 위한 분산알고리즘은 제안되지 않았으며, 문헌 [7]에 제안된 강결합요소문제를 해결하는 분산 알고리즘을 토폴로지 변화후의 새로운 네트워크 $N'=(V', E')$에 적용시켰을 때, 강결합요소 갱신문제를 메시지복잡도 $O(n'\log n' + e')$와 이상시간복잡도 $O(n'\log n')$에 해결할 수 있다. 즉, 문헌 [7]의 알고리즘에서는 토폴로지 변화전의 네트워크에 대해 각 프로세서가 가지고 있는 생성트리와 강결합요소에 대한 답을 이용하지 않고 있다.

본 논문에서는 토폴로지가 변하기 전의 네트워크에 대한 생성트리와 강결합요소에 대한 답 즉, 알고리즘 시작시 각 프로세서는 토폴로지 변화전의 네트워크상에서 어느 링크가 생성트리에 속하는지와 어느 링크와 어느 링크가 같은 강결합요소에 속하는지를 알고 있으므로 이들 답을 보조정보로서 효율적으로 이용함으로써 생성트리와 강결합요소 갱신문제를 효율적으로 해결하는 분산알고리즘을 제안한다. 즉, 본 논문에서 제안하는 분산알고리즘에서는 각 프로세서가 알고 있는 극히 소수의 정보만을 이용한다.

본 논문에서 제안하는 분산알고리즘의 메시지복잡도는 $O(n'\log n' + (n' + s' + t))$, 이상시간 복잡도는 $O(n'\log n')$이다. 여기서 n'는 토폴로지 변화후의 네트워크의 프로세서수, s는 추가 링크수를 나타낸다. 또 t는 삭제 링크를 포함하는 강결합요소에 포함되어 있는 전체 링크수를 나타낸다.

2. 정 의

본 논문에서는 유방향 연결그래프만을 취급하므로 이후 유방향 연결그래프를 단순히 그래프라고 한다.

먼저 그래프에 관한 용어를 다음과 같이 정의한다.

[정의1] 강결합 그래프(strongly connected graph) : 그

래프 G=(V,E)상의 임의 두 노드 u, v∈V 사이에 경로가 존재할 때에 한해 G를 강결합 그래프라고 한다.

[정의2] 강결합요소(strongly connected component) : 그래프 G의 강결합 부분그래프에서 극대부분을 G의 강결합요소라고 한다.

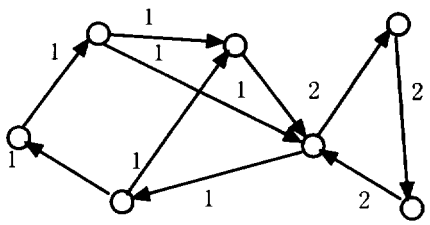
네트워크와 분산알고리즘에 관해서 다음과 같은 가정을 둔다.

[가정1] 네트워크상에 공유메모리는 없고, 프로세서간의 통신은 링크를 통한 메시지 교환만으로 행한다. 프로세서 u가 인접 프로세서 v에게 보낸 메시지는 보낸 순서대로, 유한시간내에, 도중에 없어지지 않고, v에게 반드시 보내진다.

[정의3] 강결합요소문제 : 각 프로세서는 각 인접 링크 e에 대해 다음의 조건을 만족하는 레이블을 할당한다. 단 프로세서 u가 링크 e에 할당하는 레이블을 L(u,e)라고 한다.

u, v를 임의의 프로세서라고 하고, e, e'를 각각 u, v에 연결된 임의의 링크라고 한다. e, e'와 같은 강결합요소에 속할 때에 한해 L(u,e) = L(u,e)이다. 단, L(u,e) = 0인 링크는 어느 강결합요소에도 속하지 않음을 의미한다.

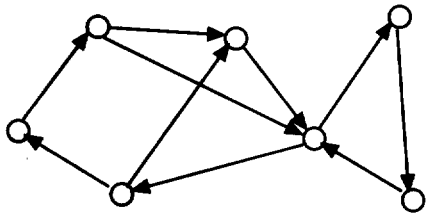
[예1] 다음 그래프에는 두 개의 강결합요소가 존재하며, 같은 강결합요소에 속하는 모든 링크에는 다음과 같이 같은 레이블이 할당되며, 각 프로세서는 이들 레이블을 결과로서 가지게 된다.



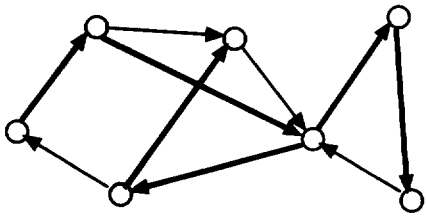
(그림 1) 강결합요소문제의 예
(Fig. 2) An example of Strongly connected component Problem

[정의4] 생성트리구성문제 : 각 프로세서는 인접링크 중에서 어느 링크가 생성트라에 속하는 링크인지를 알고 있다.

[예2] (그림 2) (a)의 그래프에 대한 생성트리는 (b)에 굵은 선으로 나타내었다. 각 프로세서는 인접 링크중에서 어느 링크가 생성트리에 속하는지를 결과로서 가지게 된다.



(a) 유방향 그래프 G



(b) G의 생성트리

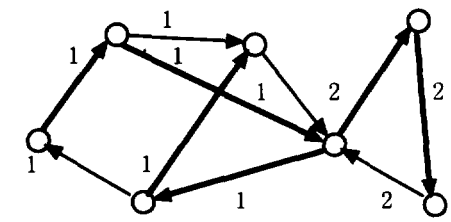
(그림 2) 생성트리구성문제의 예
(Fig. 2) An example of Spanning tree Constmction Problem

[정의5] 생성트리와 강결합요소 갱신문제(Spanning tree and strongly-connected components Updating Problem, 이후 SSUP라고 한다) : 생성트리와 강결합요소 갱신문제는 다음의 초기상태에서 시작하여 최종상태에 이르는 문제를 말한다.

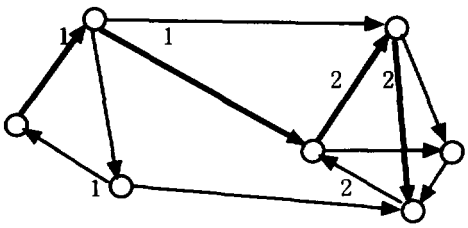
(초기상태) 토폴로지 변화전의 네트워크 N에 대한 생성트리와 강결합요소가 이미 구성되어 있으며, 각 프로세서는 토폴로지 변화후의 네트워크 N'에서 어느 링크가 새로 추가되고 삭제되었는지를 알고 있다.

(최종상태) 토폴로지 변화후의 네트워크 N'에 대한 생성트리와 강결합요소가 구성되어 있다.

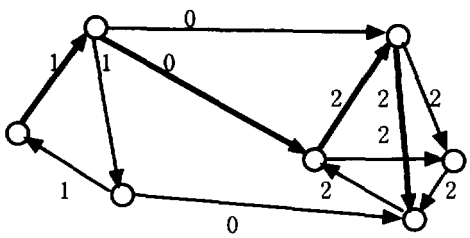
[예3] (그림 3) (a)와 같이 토폴로지 변화전의 네트워크에 대해 강결합요소와 생성트리가 구성되어 있는 상황에서, (b)와 같이 토폴로지 변화가 생겼을 때, (c)와 같이 생성트리와 강결합요소를 재구성한다.



(a) 토폴로지 변화전의 네트워크



(b) 토폴로지 변화후의 네트워크



(c) SSUP의 답

(그림 3) 생성트리와 강결합요소 갱신문제의 예
(Fig. 3) An example of SSUP

[정의6] 분산알고리즘 : 분산알고리즘이란 각 프로세서가 실행하는 프로그램으로 구성된다. 자발적으로 프로그램 실행을 개시하는 프로세서를 시작 프로세서(start processor)라고 한다. 시작 프로세서이외의 프로세서는 인접 프로세서로부터 메시지를 받으면 프로그램 실행을 개시하는데, 여기서는 시작 프로세서는 하나라고 가정한다. 메시지 전송 지연의 차이, 프로세서 동작 속도의 차이 등에 의해 여러 가지 실행과정이 발생할 수 있으나, 어느 실행과정에 있어서도 유한개의 명령 실행후에 모든 프로세서가 프로그램 실행을 끝냈을 때 분산알고리즘이 종료했다고 한다. 어떤 분산알고리즘 A가 SSUP를 해결한다는 것은 유한시간내에 A가 종료되고, 그때 각 프로세서가 어느 인접 링크가 생성트리에 속하는지 알고, 모든 인접 링크에 대해 강결합요소 문제의 해답이 되는 레이블을 가지는 것이다.

[가정2] 각 프로세서가 실행하는 프로그램은 동일하다.

분산알고리즘의 평가는 일반적으로 메시지량과 시간을 이용해서 행한다. 시간평가는 메시지의 전송지연과 각 프로세서의 동작 속도에 차이가 있기 때문에 용이하지 않아서 일반적으로 이상시간복잡도라는 개념을 이용한다.

[정의7] 메시지복잡도는 분산알고리즘 실행도중에 네트워크의 모든 프로세서 사이에서 교환되는 메시지수이다. 또, 이상적인 시간복잡도는 프로세서내에서 처리시간을 무시하고, 메시지가 링크에 전달되는 전송시간을 1단위시간으로 했을 때 알고리즘이 종료할 때까지의 단위시간수이다.

3. 알고리즘

여기서는 G=(V, E)에 대한 생성트리 갱신문제와 강결합요소 갱신문제를 동시에 해결하는 분산 알고리즘을 제안한다. 본 알고리즘은 알고리즘 시작시 네트워크상의 임의 개의 프로세서가 시작 프로세서가 되는 경우에도 확장하여 실행할 수 있지만, 이 장에서는 설명을 간단히 하기 위해 다음과 같은 가정을 두기로 한다.

[가정3] 알고리즘시작시 네트워크상의 하나의 프로세서가 시작 프로세서이다.

보다 일반적인 가정인 임의 개의 프로세서가 시작 프로세서인 경우에 대해서는 4장에서 언급하기로 한다.

3.1 강결합요소의 성질

본 알고리즘에서는 강결합요소의 성질 분석을 통해 알고리즘의 효율화를 꾀하였으므로 먼저 강결합요소의 성질 분석을 하기로 한다.

문헌 [6]에는 시작 프로세서가 하나인 경우의 강결합요소문제를 해결하는 분산 알고리즘이 제안되었는데, 이 알고리즘을 토폴로지 변화후의 새로운 네트워크

N′=(V′, E′)에 적용시켰을 때, 강결합요소 갱신문제를 메시지복잡도 O(e′)와 이상시간복잡도 O(n′)에 해결할 수 있다. 여기서, n′와 e′는 N′에서의 프로세서수(=|V′|)와 링크수(=|E′|)를 나타낸다. 문헌 [6]의 알고리즘에서는 강결합요소문제를 해결하기 위해 깊이우선탐색법(depth-first search)을 이용하는데, 깊이우선탐색법에서는 네트워크상에 있는 모든 링크를 체크해야 하므로 O(e′)의 메시지복잡도가 소요된다.

그러나, 강결합요소에 있어서 다음 보조정리가 성립하는데, 이 보조정리는 네트워크상에 있는 모든 링크를 체크할 필요가 없으며, 이로 인해 강결합요소 갱신문제의 메시지복잡도를 줄일 수 있음을 의미한다.

[보조정리] 토폴로지 변화전의 네트워크 상의 임의의 강결합요소(SC라고 한다)에 삭제 링크가 없다면, SC는 토폴로지 변화후의 네트워크에 있어서 하나의 강결합요소에 포함된다.

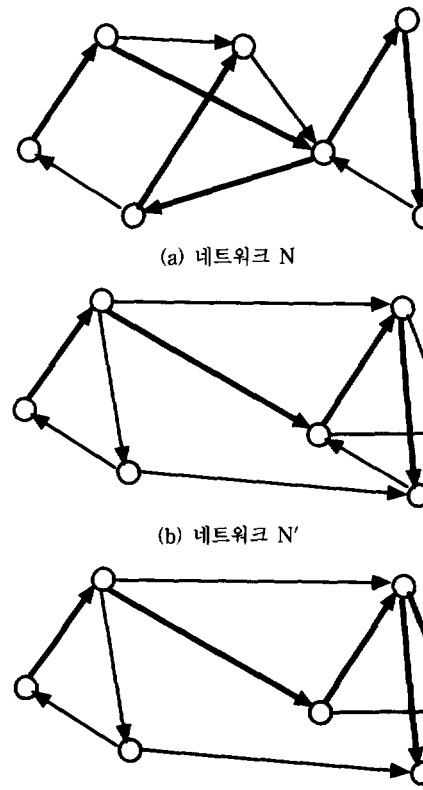
삭제링크를 포함한 강결합요소의 경우, 링크의 삭제로 인해 강결합요소가 여러 개의 강결합요소로 분할될 수 있으므로 이러한 강결합요소에 대해서는 모든 링크를 체크해야 한다. 그러나, 상기 보조정리로부터 토폴로지 변화전의 네트워크 상의 임의의 강결합요소 SC에 삭제 링크가 없다면, SC는 토폴로지 변화후의 네트워크에 있어서 하나의 강결합요소에 포함되므로 SC상의 링크에 대해서는 체크할 필요는 없지만, 이들 링크가 토폴로지 변화후의 새로운 네트워크상에서보다 큰 강결합요소에 포함될지 모른다. 따라서 이러한 강결합요소 SC에 대해서도 어느 정도까지는 체크를 해야 한다. 강결합요소 SC가 다른 강결합요소와 함께 보다 큰 강결합요소를 구성하는지 여부를 체크하기 위해, 본 논문에서는 강결합요소 SC의 생성트리에 속하는 링크만을 이용함으로써 알고리즘의 효율화를 꾀한다.

본 논문에서 생성트리와 강결합요소를 재구성하는 문제를 효율적으로 해결하기 위해 다음과 같은 네트워크 $N^*=(P', L^*)$을 정의하여 이용하기로 한다. 여기서 L_a와 L_d는 각각 추가된 링크집합과 삭제된 링크집합을 나타내고, SC_i와 T_i는 각각 N상의 하나의 강결합요소와 강결합요소에 대한 생성트리를 나타낸다.

$$N^* = \bigcup_{1\le i\le m} A_i \cup L_a - L_d$$

$$\begin{cases} A_i = T_i & SC_i \cap L_d = 0\text{인 경우} \\ A_i = SC_i & SC_i \cap L_d \neq 0\text{인 경우} \end{cases}$$

[예4] (그림 4) (b)에서는 (a)에 나타낸 네트워크에서 토폴로지 변화로 인해 변화된 네트워크를 나타냈으며, (c)에는 네트워크 N^*를 나타내었다.



(a) 네트워크 N

(b) 네트워크 N′

(c) 네트워크 N^*

(그림 4) 네트워크 N^*의 예

3.2 알고리즘의 개략

본 알고리즘의 개략은 다음과 같다.

[스텝1] 앞에서 정의한 네트워크 N^*를 구성한다.
토폴로지 변화후의 새로운 네트워크 N′상에 있는 모든 프로세서는 어느 인접 링크가 N^*에 속하는지를 결정한다.

[스텝2] N^*에 대한 강결합요소를 판정한다.
같은 레이블을 가진 링크가 N^*에 대한 강결합요소를 이루도록 각 프로세서는 N^*상의 각 인접 링크에 대해 레이블을 할당한다.

[스텝3] 토폴로지 변화후의 네트워크 N'에 대한 강결합요소를 판정한다.
같은 레이블을 가진 링크가 N'에 대한 강결합요소를 이루도록 각 프로세서는 N'상의 각 인접 링크에 대해 레이블을 할당한다.

3.3 알고리즘

여기서는 SSUP를 효율적으로 해결하고 알고리즘에 대해 보다 자세히 설명하기로 한다.

3.3.1 스텝1

스텝1에서 N^*를 구성한다. 즉, 토폴로지 변화후의 새로운 네트워크 N'상의 각 프로세서는 어느 인접링크가 N^*에 속하는지를 결정한다.

N^*를 효율적으로 구성하기 위해, 시작 프로세서를 루트로 하는 N^*에 대한 깊이우선탐색을 실시하면서 N^*를 구성한다. 깊이우선탐색에 의해 탐색된 각 프로세서 u는 어느 인접 링크가 N^*에 속하는지를 다음과 같이 결정하는데, 어느 인접 링크가 N^*상의 링크인지에 대한 판정이 끝나면, 프로세서 u는 깊이우선탐색을 계속한다. 즉, u는 인접프로세서에게 깊이우선탐색을 위한 토큰을 보낸다.

- 새로 추가된 링크 (u,v)에 대해, (u,v)가 N^*에 속하는 것으로 한다.
- 토폴로지 변화전의 네트워크 N상의 각 링크 (u,v)에 대해, (u,v)가 포함된 강결합요소에 삭제된 링크가 존재하거나 또는 (u,v)가 생성트리 T에도 속하지 않을 때 (u,v)가 N^*에 속하지 않는 것으로 한다.

토폴로지 변화전의 각 링크 (u,v)에 대해, 프로세서 u는 (u,v)를 포함하는 강결합요소 SC_i에 삭제링크가 존재하는지 여부를 정해야 한다. 토폴로지 변화전의 네트워크에 대한 생성트리 T의 일부가 토폴로지 변화후에도 존재하므로, 프로세서 u는 SC_i에 대한 생성트리 T_i를 이용하여 삭제링크의 존재여부를 정할 수 있다($T_i = T \cap SC_i$). 만일 SC_i에 삭제링크가 존재한다면, T_i는 여러 개의 프래그먼트로 분할될지 모른다. 그러나 u를 포함하는 프래그먼트에는 SC_i에서 삭제링크에 인접한 프로세서 w가 존재한다. 따라서 T_i의 프래그먼트를 이용함으로써 프로세서 u는 SC_i에 삭제링크가 있는지 여부를 효율적으로 판단할 수 있다. 그 프래그먼트에서의 이중 체크를 피하기 위해 프로세서 u는 그 프래그먼트상의 모든 프로세서에게 삭제링크의 존재여부를 알린다.

3.3.2 스텝2

스텝2에서는 문헌 [7]의 알고리즘 N^*에 적용시킴으로써 N^*의 강결합요소를 판정한다. 즉, 본 논문에서 N^*에 대해 깊이우선탐색법을 이용하는데, 스텝2를 실행한 결과 N^*에 대한 강결합요소와 함께 N^*에 대한 생성트리 T'를 구성한다(구체적으로 T'는 대한 N^*에 대한 깊이우선생성트리이다). 본 논문에서는 이 생성트리 T'를 스텝3에서 이용한다.

3.3.3 스텝3

스텝3에서는 토폴로지 변화후의 새로운 네트워크 N'에 대한 강결합요소를 판정한다[7]. 모든 프로세서는 어느 인접링크가 같은 강결합요소에 속하는지를 결정하는데, SSUP에서는 같은 강결합요소에 속하는 링크에는 같은 레이블을 할당해야 한다.

각 링크에 할당된 레이블을 정하기 위해, 본 논문에서는 스텝2에 구성된 N^*에 대한 생성트리 T'를 이용한다. N'상에서 같은 강결합요소에 속하는 프로세서는 T'에서 연속적으로 나타난다는 성질이 있는데, 본 논문에서는 이 성질을 이용함으로써 효율적으로 해결한다. 즉, $T' \cap SC_i'$는 새로운 네트워크 N'에 대한 각 강결합요소 $SC_i'(1 \le i \le m')$에 대한 SC_i의 생성트리가 된다. 각 SC_i'에 대해 SC_i'의 링크에 인접한 프로세서중에서 루트에 가장 가까운 프로세서(u_i라고 한다)가 존재하는데, 이러한 u_i를 SC_i'의 대표자라고 부르기로 한다.

스텝3에서는 T'에 대해 깊이우선탐색을 실시하는데, 깊이우선탐색을 수행하는 동안에 N'에 대한 강결합요소에 할당될 레이블인 t를 기억한다. 대표자 u_i가 깊이우선탐색을 위해 SC_i'의 링크를 탐색하기 시작할 때, u_i는 레이블 t를 t+1로 갱신해서 SC_i의 레이블을 t+1로 정한다. u_i에서부터 시작되는 SC_i에 대한 깊이우선탐색에 의해 SC_i의 링크에 인접한 프로세서는 SC_i의 레이블을 전달받게 된다.

3.4 알고리즘 평가

[정리1] 본 논문에서 제안한 분산알고리즘의 메시지복잡도는 $O(n' + s + t)$이다.

(증명) 스텝1에서는 네트워크 N^*를 구성하는데, 이를 위해 삭제된 링크를 포함하는 강결합요소에 대

해서는 모든 링크에 메시지를 보내고, 삭제된 링크를 포함하지 않는 강결합요소에 대해서는 강결합요소의 생성트리에 속하는 링크에만 메시지를 보낸다. 그리고, 새로 추가된 링크에도 메시지를 보내므로 스텝1에서는 O(n′ + s + t)의 메시지가 필요하다. 또한 스텝2와 3에서는 각각 N*와 N′ 대한 강결합요소를 판정하는데, 스텝1에서 메시지를 보낸 링크에 대해서만 메시지를 보내므로 O(n′ + s + t)의 메시지를 필요로 한다.

따라서, 본 논문에서 제안한 알고리즘의 메시지복잡도는 O(n′ + s + t)이다.

[정리2] 본 논문에서 제안한 분산알고리즘의 이상시간 복잡도는 O(n′)이다.

(증명) 스텝1에서는 네트워크 N*를 구성하는데, 이를 위해 삭제된 링크를 포함하는 강결합요소에 속하는 모든 링크에 메시지를 보내고, 삭제된 링크를 포함하지 않는 강결합요소에 대해서는 강결합요소의 생성트리에 속하는 링크에만 메시지를 보낸다. 그리고, 새로 추가된 링크에도 메시지를 보내므로 스텝1에서는 O(n′)의 시간이 필요하다, 또한 스텝2와 3에서는 각각 N*와 N′ 대한 강결합요소를 판정하는데, 이 때에도 스텝1에서 메시지를 보낸 링크에 대해서만 메시지를 보내므로 O(n′)의 시간을 필요로 한다.

따라서, 본 논문에서 제안한 알고리즘의 이상시간복잡도는 O(n′)이다.

4. 알고리즘 확장

앞에서는 설명을 위해 알고리즘 시작시 하나의 프로세서가 시작 프로세서라는 가정하에서 알고리즘을 설명했지만, 임의 개의 프로세서가 시작 프로세서로 되는 보다 일반적인 가정이 성립하는 경우에는 다음을 3장에서 제안한 알고리즘의 전단계로 둠으로써 알고리즘을 확장할 수 있다.

[전단계] 임의 개의 프로세서중에서 하나의 프로세서를 리더로 선택한다.

이 단계에서 선택된 프로세서를 시작 프로세서 즉, 루트로 해서 앞장에서 제안한 알고리즘을 적용함으로써 문제를 해결할 수 있다.

[정리3] SSUP를 위한 확장 알고리즘의 메시지복잡도는 O(n′log n′ + (n′ + s + t))이다.

(증명) 전단계에서는 문헌 [2]의 알고리즘을 이용하여 N′상의 프로세서중에서 하나의 프로세서를 시작 프로세서로 선택하되, 이 때도 삭제링크를 포함하지 않은 강결합요소에 대해서는 생성트리에 속하는 링크에만 메시지를 보냄으로써 O(n′log n′ + (n′ + s + t))의 메시지로 리더를 선택할 수 있다.

따라서, 본 논문에서 제안한 SSUP를 해결하기 위한 확장된 알고리즘의 메시지복잡도는 O(n′log n′ + (n′ + s + t))이다.

[정리4] SSUP를 위한 확장 알고리즘의 이상시간복잡도는 O(n′log n′)이다.

(증명) 전단계에서는 문헌 [2]의 알고리즘을 이용하여 N′상의 프로세서중에서 하나의 프로세서를 시작 프로세서로 선택하되, 이 때도 삭제링크를 포함하지 않은 강결합요소에 대해서는 생성트리에 속하는 링크에만 메시지를 전송함으로써 O(n′log n′)의 시간에 리더를 선택할 수 있다.

따라서, 본 논문에서 제안한 SSUP를 해결하기 위한 확장된 알고리즘의 이상시간복잡도는 O(n′log n′)이다.

참 고 문 헌

[1] P.Almonti, P.Flocchimi and N.Santoro : "Finding the Extrema of a Distributed Multiset," 8th International Workshop on Distributed Algorithms, Netherlands, 1994.

[2] B.Awerbuch : "Optimal Distributed Algorithm for Minimum Weight Spanning Tree, Counting, Leader Election and related problems," Proceeding 19th Annual ACM Symposium on Theory of Computing, pp.230-240, 1987.

[3] T.H.Cormen, C.E.Leiserson, and R.L.Rivest : "Introduction to Algorithms," The MIT Press,

1990.

[4] T.Kameda and M.Yamashita : "Distributed Algorithms(in Japanese)," Kindai-Kagaku-Sya. 1994.

[5] K.B.Lakshmanan, N.Meenakshi, and K.Thulasiraman : "A time optimal message efficient distributed algorithm for depth first search," Information Processing Letters, 25 : pp.103-109, 1987.

[6] 박정호, 민준영 : "MWST재구성 분산알고리즘", 한국정보처리학회논문지, Vol.1, No.2, July. 1993.

[7] 박정호, 조이남 : "네트워크상의 강결합요소를 구하는 최적의 분산알고리즘", 한국정보과학회논문지, Vol.19, No.4, 1992.

[8] G.Tel, "Introduction to Distributed Algorithms," Cambridge University Press, 1994.

박 정 호

e-mail : jhpark@omega.sunmoon.ac.kr

1980년 성균관대학교 사범대학 졸업(문학사)

1980년~1982년 성균관대학교 경영대학원 정보처리학과(경영학석사)

1985년~1987년 日本 오사까대학교 대학원 정보공학전공(공학석사)

1987년~1990년 日本 오사까대학교 대학원 정보공학전공(공학박사)

1996년~현재 한국정보처리학회 총무이사

1991년~현재 선문대학교 컴퓨터정보학부 부교수

관심분야 : 분산알고리즘, 소프트웨어공학, XML, 원격교육

박 윤 용

e-mail : yypark@omega.sunmoon.ac.kr

1983년 숭전대학교 계산통계학과 졸업(학사)

1985년 서울대학교 대학원 계산통계학과(이학 석사)

1994년 서울대학교 대학원 계산통계학과(이학 박사)

1985년~1992년 한국전자통신연구소 연구원

1992년~현재 선문대학교 컴퓨터정보학부 조교수

관심분야 : 분산처리운영체제, 멀티미디어시스템, 고장감래 시스템

최 성 희

e-mail : schoi@omega.sunmoon.ac.kr

1977년 2월 서강대학교 수학과 졸업(이학사)

1988년 8월 Pennsylvania State University, Computer Science(M.S.)

1994년 5월 University of Kentucky, Computer Science(Ph.D.)

1977년 3월~1982년 6월 국방과학연구소 연구원

1995년 3월~현재 선문대학교 컴퓨터정보학부 교수

관심분야 : Algorithm, Automate, 수치해석, Compiler Design