

트리에서 길이 제한이 있는 가장 무거운 경로를 찾는 알고리즘

김 성 권[†]

요 약

에지마다 길이와 무게(둘 다 양수, 음수, 0 가능)가 주어진 트리에서, 길이의 합이 주어진 값 이하이면서 무게의 합이 가장 큰 경로를 찾는 $O(n \log n \log \log n)$ 시간 알고리즘을 제시한다. 이전의 결과 $O(n \log^2 n)$ 보다 향상된 것이다. 여기서, n 은 트리가 가지는 노드의 수이다.

키워드 : 길이 제한이 있는 가장 무거운 경로, 트리, 알고리즘

Algorithm for finding a length-constrained heaviest path of a tree

Sung-Kwon Kim[†]

ABSTRACT

Abstract: In a tree with each edge associated with a length and a weight (positive, negative, or zero are possible) we develop an $O(n \log n \log \log n)$ time algorithm for finding a path such that its sum of weights is maximized and its sum of lengths does not exceed a given value. The previously best-known result is $O(n \log^2 n)$, where n is the number of nodes in the tree.

Key Words : Length-Constrained Path, Tree, Algorithm

1. 서 론

트리 $T=(V,E)$ 는 노드 집합 V 와 에지 집합 E 로 이뤄진다. $E \neq \emptyset$ 이라 가정한다. 각 에지 $e \in E$ 마다 길이 (length)와 무게 (weight)라 부르는 실수 (양수, 음수, 0 모두 가능) $l(e)$ 와 $w(e)$ 가 주어진다. T 에서 두 개의 다른 노드 u 와 v 를 잇는 경로를 $\pi(u,v)$ 로 표시하자. 경로 $\pi(u,v)$ 의 에지들의 길이를 모두 합한 것을 그 경로의 길이라 하고, 무게를 모두 합한 것을 그 경로의 무게라 한다. 즉, $\pi(u,v)$ 의 길이는 $l(\pi(u,v)) = \sum_{e \in \pi(u,v)} l(e)$ 이고, 무게는 $w(\pi(u,v)) = \sum_{e \in \pi(u,v)} w(e)$ 이다.

LHP(길이 제한이 있는 가장 무거운 경로, length-constrained heaviest path) 문제: 각 에지마다 길이와 무게가 있는 트리 $T=(V,E)$ 와 실수 L 이 주어질 때, 길이가 L 이하이면서 무게가 가장 무거운 경로를 T 에서 찾으시오. 즉, $\max\{w(\pi(u,v)) \mid u,v \in V \text{ and } l(\pi(u,v)) \leq L\}$ 가 되는 $\pi(u,v)$ 를 찾으시오.

트리 모양으로 구성된 네트워크에서 트리 네트워크의 일부를 고속설비로 교체하고자 할 경우, 어느 곳을 선택하는 것이 가장 좋은가라는 문제를 해결할 때 LHP 문제를 이용할 수 있다. 책정된 예산 내에서 가장 효율이 높은 곳을 찾아야 하는데, 각 에지의 길이는 그 에지를 교체하는데 필요한 금액이고, 각 에지의 무게는 그 에지를 교체하여 얻을 수 있는 이익이 된다. 여기서, 이익은 에지의 트래픽 양에 따라 정하는 척도이면 된다.

LHP 문제를 해결하는 알고리즘은 [3]에 있는 $O(n \log^2 n)$ 시간 알고리즘이 현재로는 가장 빠르다. 본 논문에서는 이를 $O(n \log n \log \log n)$ 으로 향상 시킨다. 여기서 n 은 T 가 가지는 노드의 수다. 2절에서는 [3]의 알고리즘과는 전혀 다른 $O(n \log^2 n)$ 시간 알고리즘을 제시한다. 그리고 3절에서는 이 알고리즘의 구현 방법을 개선하여 $O(n \log n \log \log n)$ 시간에 구현할 수 있음을 보인다.

2. $O(n \log^2 n)$ 시간 알고리즘

2.1 알고리즘 개요

트리 $T=(V,E)$ 가 인접 리스트 방법으로 저장된다고 가정한다. $|V|=n$ 이고 $|E|=n-1$ 이다. 노드 u 에 연결된 이웃 노드들의 집합을 $N(u)$ 로 표시한다. 즉, $N(u) = \{v \mid (u,v) \in E\}$ 이다. T 에서 에지 (u,v) 를 제거하면 두 개의 서브트리가 생기는데, 하나는 u 를 포함하고 다른 하나는 v 를 포함한다. 이때, u 를 포함하는 서브트리의 크기 (노드의 개수)를 $s(u,v)$ 로 표시한다. 따라서 u 가 리프노드이면 $s(u,v)=1$ 이 된다. 또 모든 에지 (u,v) 에 대해서 $s(u,v)+s(v,u)=n$ 이 성립한다.

만약 모든 $u \in N(x)$ 에 대해서 $s(u,x) \leq n/2$ 가 성립하면 x 를 T 의 센트로이드 (centroid)라 부른다. T 의 센트로이드는 $O(n)$ 시간에 구할 수 있다. 가장 널리 알려진 방법은 T 를 임의의 노드를 루트로 하여 루트가 있는 트리로 바꾼 다음, 모든 노드 u 에 대해서 $d(u)$ 를 계산한다. $d(u)$ 는 u 와 u 의 후손노드들로 이뤄진 서브트리의 크기이다. u 가 리프노드이면 $d(u)=1$ 이고, u 가 $k \geq 1$ 개의 자식노드 v_1, \dots, v_k 를 가지면

$d(u) = 1 + \sum_{i=1}^k d(v_i)$ 이다. 트리를 포스트오더로 방문하면서 모든

* 본 논문은 2005년도 중앙대학교 학술연구비 지원에 의한 것임.

† 종신회원 : 중앙대학교 컴퓨터공학부 교수

논문접수 : 2006년 7월 26일, 심사완료 : 2006년 9월 14일

노드에 대해서 $d(u)$ 를 계산한다. 이 과정 중에 처음으로 $d(u) \geq n/2$ 가 되는 노드 u 가 T 의 센트로이드가 된다.

소정리 1은 센트로이드를 중심으로 트리를 최대 세 개의 서브트리로 분할할 수 있음을 보이는 것으로 증명은 논문 [2]에 있다.

[소정리 1]: $n \geq 3$ 일 때, x 가 T 의 센트로이드이면 모든 $i=1,2,3$ 에 대해서 $\sum_{u \in N_i} s(u,x) \leq n/2$ 가 되도록 $N(x)$ 를 세 개의 부분집합 N_1, N_2, N_3 로 나눌 수 있다. N_3 는 공집합이 될 수도 있다.

소정리 1의 N_1, N_2, N_3 를 이용하여 아래와 같이 세 개의 트리 T_1, T_2, T_3 를 만든다. $N_3 = \emptyset$ 이면 $T_3 = \emptyset$ 이다. 각 T_i 는 x 를 포함하고, x 와 N_i 에 있는 노드들을 연결하는 에지들을 포함하고, x 에서 N_i 에 있는 노드들을 통하여 도달할 수 있는 T 의 모든 노드들과 에지들을 포함한다. 모든 T_i 가 x 를 포함하고, x 를 제외하면 T_i 들은 서로 중복되는 곳이 없다. 따라서 $n_i = |T_i|$ 라 하면, $n_i \leq n/2 + 1$ 이고 $n_1 + n_2 + n_3 = n + 2$ 이다. $N_3 = \emptyset$ 이면 $n_3 = 0$ 이고 $n_1 + n_2 = n + 1$ 이다.

제시하려는 알고리즘은 앞에서 언급했듯이 분할정복 방법으로 수행한다. 알고리즘의 기술 편의상 T_3 가 존재한다고 가정한다.

입력: 트리 T 와 실수 L .

출력: T 에서 길이가 L 이하인 경로가 있으면 그 중에서 가장 무거운 경로의 무게, 만약 그런 경로가 없으면 $-\infty$. (아래 알고리즘을 조금 고치면 경로 자체를 구할 수 있으므로 편의상 무게만 구한다.)

[분할] T 가 에지 e 로만 이뤄진 경우, $l(e) \leq L$ 이면 $w(e)$ 를 아니면 $-\infty$ 을 반환한다. 그 외 경우, T 의 센트로이드 x 를 구하고, 소정리 1의 방법을 이용하여 세 개의 트리 T_1, T_2, T_3 를 만든다.

[정복] $i=1,2,3$ 에 대해서 T_i 내에서 길이가 L 이하이면서 가장 무거운 경로의 무게 W_i 를 재귀적으로 구한다.

[통합] 이제, T 에서 x 를 지나면서 T_i 와 T_j ($i \neq j$)에 걸쳐 있는 경로들 중에서 길이가 L 이하이면서 가장 무거운 경로의 무게 W_x 를 구한다. 그러면 $\max\{W_1, W_2, W_3, W_x\}$ 가 우리가 원하는 답이 된다.

$T(n)$ 을 위 알고리즘이 n 개의 노드를 가진 트리에서 길이가 L 이하이면서 가장 무거운 경로의 무게를 구하는데 필요한 최악의 경우의 수행시간이라 하자. $n=1,2$ 에서 $T(n) = O(1)$ 이고, $n \geq 3$ 에 대해서

$$T(n) = \sum_{i=1,2,3} T(n_i) + \text{Divide}(n) + \text{Combine}(n)$$

이 성립한다. 여기서, $i=1,2,3$ 에 대해 $n_i \leq n/2 + 1$ 이고, $n_1 + n_2 + n_3 = n + 2$ 이며, $\text{Divide}(n)$ 은 [분할] 단계에 필요한 시간이고, $\text{Combine}(n)$ 은 [통합] 단계에 필요한 시간이다. [분할] 단계에서 x 를 구하는 것과 T_i 들을 구하는 것은 모두 선형 시간에 가능하므로 $\text{Divide}(n) = O(n)$ 이다. 앞으로 $\text{Combine}(n) = O(n \log n)$ 임을

보이면 $T(n) = O(n \log^2 n)$ 이 되어서 우리가 원하는 결과를 얻는다.

2.2 [통합] 단계

양 끝점이 모두 x 가 아니면서 하나의 끝점은 T_i 에 있고 다른 하나는 T_j ($i \neq j$)에 있는 모든 경로들 중에서 길이가 L 이하이면서 가장 무거운 경로의 무게를 $W_x^{i,j}$ 라 하자. $(i,j) \in \{(1,2), (1,3), (2,3)\}$ 에 대해서 $W_x^{i,j}$ 를 구한 후, 그 중 최대가 W_x 가 된다. 즉, $W_x = \max\{W_x^{1,2}, W_x^{1,3}, W_x^{2,3}\}$ 이다. 여기서는 $W_x^{1,2}$ 를 $O(n \log n)$ 시간에 구하는 방법만 설명한다. 다른 두 개도 비슷한 방법으로 구할 수 있다.

T_1 의 n_1 개의 노드가 정수 $1, \dots, n_1$ 으로 번호가 주어진다 가정하고, 배열 $A[1..n_1]$ 을 만든다. T_1 의 모든 노드 $i=1, \dots, n_1$ 에 대해서 $A[i].l$ 과 $A[i].w$ 을 구한다. $A[i].l$ 과 $A[i].w$ 는 x 와 i 를 잇는 경로의 길이와 무게이다. T_1 을 프리오더 순으로 방문하면서 $O(n_1)$ 시간에 $A[1..n_1]$ 들을 모두 계산할 수 있다. 비슷하게 T_2 에서 $B[j].l$ 과 $B[j].w$ 를 x 와 j 를 잇는 경로의 길이와 무게로 정의한다 ($j=1, \dots, n_2$). 모든 $B[1..n_2]$ 들도 $O(n_2)$ 시간에 구할 수 있다.

당연히, $W_x^{1,2} = \max\{A[i].w + B[j].w \mid A[i].l + B[j].l \leq L\} \cup \{0\}$ 가 된다. 즉, $A[i].l + B[j].l \leq L$ 인 i, j 쌍 중에서 $A[i].w + B[j].w$ 가 최대가 되는 것을 찾으면 된다. 모든 i, j 쌍을 모두 고려하면 $O(n_1 n_2)$ 시간이 걸리므로 더 빠른 방법이 필요하다.

배열 A 를 무게가 증가하는 순으로 정렬하고, 노드들의 번호도 그 순서대로 재배정한다. 즉, 노드들의 번호가 $A[1].w \leq A[2].w \leq \dots \leq A[n_1].w$ 가 되도록 재배정한다. 배열 B 도 비슷하게 무게가 증가하는 순으로 정렬하여 $B[1].w \leq B[2].w \leq \dots \leq B[n_2].w$ 가 되도록 노드 번호를 재배정한다.

j_0, j_1, \dots, j_m 을 아래처럼 정의한다. $B[0].l = -\infty$ 라 가정한다.

- * $j_0 = 0$.
- * $k=1, 2, \dots$ 에 대해서 j_k 는 $B[j_{k-1}+1].l, B[j_{k-1}+2].l, \dots, B[n_2].l$ 중에서 최소인 것을 가리키는 인덱스이다. 즉, $B[j_k].l = \min\{B[j_{k-1}+1].l, B[j_{k-1}+2].l, \dots, B[n_2].l\}$ 이다. 만약 최소가 여러이면 가장 큰 인덱스를 택한다.
- * 이렇게 계속해서 $j_k = n_2$ 가 될 때의 k 가 m 이다. $B[j_0].l < B[j_1].l < \dots < B[j_m].l$ 이 성립한다. j_0, j_1, \dots, j_m 을 아래와 같이 정의하더라도 위의 정의와 같은 결과가 나온다.
 - * $j_m = n_2$.
 - * $k=m-1, m-2, \dots, 0$ 에 대해서, $j_k = \max\{j \mid j < j_{k+1} \text{ and } B[j].l < B[j_{k+1}].l\}$ 이다. 즉, j_{k+1} 에서 j 가 작아지는 쪽으로 갈 때 처음으로 $B[j].l < B[j_{k+1}].l$ 가 되는 j 가 j_k 이다.

이 정의를 따르면 j_0, j_1, \dots, j_m 를 $O(n_2)$ 시간에 계산할 수 있다.

[소정리 2] B 에서 $B[j_1], B[j_2], \dots, B[j_m]$ 만 고려하면 된다.

[증명] $1 \leq k \leq m$ 에 대해서, $B[j_k]$ 가 $B[j_{k-1}+1], B[j_{k-1}+2], \dots, B[j_k-1]$ 들과 비교하면 길이는 작거나 같지만 무게는 크거나 같다. 따라서 $B[j_k]$ 만 고려하고 $B[j_{k-1}+1], B[j_{k-1}+2], \dots, B[j_k-1]$ 는 고려할 필요가 없다.

$1 \leq k \leq m$ 에 대해서 $i_k = \max\{\{i \mid A[i].l + B[j_k].l \leq L\} \cup \{0\}\}$ 라 한다. 즉, i_k 는 $A[i].l + B[j_k].l \leq L$ 가 성립하는 최대 인덱스 i 이고, 그런 i 가 존재하지 않으면 $i_k = 0$ 이다. A 가 무게 순으로 정렬되어 있으므로 $A[i_k].w + B[j_k].w = \max\{A[i].w + B[j_k].w \mid A[i].l + B[j_k].l \leq L\}$ 가 된다. 따라서 i_1, \dots, i_m 를 모두 구하여, $W = \max_k\{A[i_k].w + B[j_k].w\}$ 를 계산하면 이것이 우리가 원하는 $W_x^{1,2}$ 이다. $B[j_1].l < \dots < B[j_m].l$ 이므로 $i_1 \geq \dots \geq i_m$ 가 성립한다. 아래 알고리즘은 i_1, \dots, i_m 을 구하면서 동시에 W 를 계산한다.

```

(1)   $i = n_1; k = 1; W = -\infty;$ 
(2)  while( $k \leq m$ )
(3)    while( $i > 0$  and  $A[i].l + B[j_k].l > L$ )  $i--$ ;
(4)    if( $i \leq 0$ ) return  $W$ ;
(5)    if( $W < A[i].w + B[j_k].w$ )
       $W = A[i].w + B[j_k].w$ ;
(6)     $k++$ ;
(7)  return  $W$ ;

```

(1)에서 i, k, W 를 초기화한다. (3)에서 k 를 고정시킨 상태에서 i 를 감소하면서 $A[i].l + B[j_k].l \leq L$ 가 되는 i 를 찾는다. (3)의 while이 끝나면 $i_k = i$ 가 된다. 만약 $i_k \leq 0$ 이면 더 이상 알고리즘을 진행할 필요가 없으므로 (4)에서 정지한다. $i_k > 0$ 이면 (5)에서 W 와 $A[i_k].w + B[j_k].w$ 중에서 큰 것을 새로운 W 로 한다. (6)에서 k 를 증가한다. $i_k \geq i_{k+1}$ 이므로 i 는 그대로 유지하면서 (3)으로 간다. 만약 $k > m$ 이면 (2)에서 while이 끝나고, (7)에서 W 를 반환하면서 알고리즘의 수행이 끝난다. i 는 감소만 하고 k 는 증가만 하므로 수행 시간은 $O(n_1 + m)$ 이다.

[소정리 3] n 개의 노드를 가지고 있고 각 에지마다 길이와 무게가 있는 트리 T 와 실수 L 이 주어질 때, T 에서 길이가 L 이하이면서 가장 무거운 경로를 $O(n \log^2 n)$ 시간에 찾을 수 있다.

[증명] 알고리즘의 정확성은 앞에서 설명했으며, [통합] 단계의 수행 시간은 A 와 B 를 정렬하는 것을 제외하면 모두 선형 시간 걸리므로 $\text{Comline}(n) = O(n \log n)$ 이 돼서 전체 알고리즘은 $T(n) = O(n \log^2 n)$ 시간이 걸린다.

3. $O(n \log n \log \log n)$ 시간 알고리즘

2절의 알고리즘 수행시간이 $O(n \log^2 n)$ 이 되는 이유는 [통합] 단계에서 배열 A 를(물론 B 도) 정렬하는 과정에서 $O(n \log n)$ 시간을 사용하기 때문이다. [통합] 단계의 다른 과정은 모두 선형 시간에 가능하므로 A 를 $O(n \log \log n)$ 시간에 정렬하는 방법을 제시하면 $\text{Comline}(n) = O(n \log \log n)$ 이 되므로 알고리즘의 수행시간은 $T(n) = O(n \log n \log \log n)$ 이 된다. 앞으로 T 에서 연결된 서브그래프 (connected subgraph)를 간단히 컴포넌트(component) 부른다. 가장 작은 컴포넌트는 에지 하나로 이뤄진 것이다.

3.1 컴포넌트 트리

2절의 알고리즘은 재귀적 알고리즘으로서 동작하는 과정을 살펴보면, T 의 센트로이드를 구한 후 센트로이드를 기준으로

세 개의 컴포넌트 T_1, T_2, T_3 로 분할한다. 이 컴포넌트를 그것의 센트로이드를 중심으로 다시 서브 컴포넌트들로 분할한다. 이를 계속 반복하여 컴포넌트가 에지 하나로만 이뤄진 경우 까지 분할한다. 이 과정을 컴포넌트 트리라 부르는 트리 CT 로 나타낼 수 있다.

- CT 의 루트는 주어진 입력 트리 T 를 나타낸다.
- CT 의 각 노드는 알고리즘 동작 중에 만들어진 컴포넌트를 나타낸다.
- CT 의 어느 노드 α 가 세 개의 자식노드 β, γ, δ 를 가지면, α 가 나타내는 컴포넌트가 세 개의 서브 컴포넌트로 분할되었음을 의미하고, 노드 β, γ, δ 가 세 개의 서브 컴포넌트를 각각 나타낸다.
- CT 의 리프 노드가 나타내는 컴포넌트는 에지 하나로만 이뤄진 것이다.

CT 의 노드 α 가 T 의 컴포넌트 P 를 나타낸다는 것은 α 가 P 를 상징적으로 대신한다는 것이지, α 가 P 에 대한 모든 정보를 가지고 있다는 것은 아니다. 실제 α 가 가지고 있어야 할 정보는 다음에 설명한다. 소정리 1에 의하면 세 개 중 하나의 서브 컴포넌트가 공집합인 경우가 있을 수 있지만, 편의상 각 컴포넌트는 항상 세 개의 서브 컴포넌트로 분할된다고 가정한다. 당연히 CT 의 높이는 $O(\log n)$ 이다.

앞으로 CT 의 노드를 언급할 때는 노드 대신 컴포넌트를 사용하고, T 의 노드에 대해서는 노드를 계속 사용한다. 따라서 CT 의 컴포넌트에 대해서 그것의 자식 컴포넌트, 부모 컴포넌트, 형제 컴포넌트, 후손 컴포넌트라는 용어를 사용할 수 있다. 예를 들어, CT 의 루트 컴포넌트 T 는 세 개의 자식 컴포넌트 T_1, T_2, T_3 를 가진다. T_1, T_2, T_3 는 서로 형제 컴포넌트이다.

CT 의 컴포넌트 P 와 T 의 노드 v 에 대해서

- $par(P)$ 는 P 의 부모 컴포넌트를 나타낸다.
- $cent(P)$ 는 P 의 센트로이드를 나타낸다.
- $con(P) = cent(par(P))$ 로 정의한다.
- $list(v, P)$ 는 T 에서 v 와 P 의 각 노드 u 를 잇는 경로들의 무게들을 크기 순서로 정렬한 리스트이다. 즉, $list(v, P)$ 는 $\{w(\pi(v, u)) \mid u \in P\}$ 를 크기 순서로 정렬한 리스트이다. 당연히 $|list(v, P)| = |P|$ 이다.
- $list(P) = list(con(P), P)$ 로 정의한다.

$con(P)$ 에 대해 더 설명하면, P 의 부모 컴포넌트 $par(P)$ 가 센트로이드 $cent(par(P)) = con(P)$ 를 기준으로 세 개의 서브 컴포넌트로 분할되는데, 그 중 하나가 P 이다. 역으로 이 세 개의 서브 컴포넌트가 $con(P)$ 를 연결 노드(connection node)로 해서 $par(P)$ 로 통합된다. P 와 Q 가 형제 컴포넌트이면 $con(P) = con(Q)$ 이다. 2절에서 $par(T_1) = T$ 이고 $cent(T) = con(T_1) = x$ 이다. 또, A 를 무게 순으로 정렬한 것은 $list(x, T_1) = list(con(T_1), T_1) = list(T_1)$ 가 된다.

$list(v, P)$ 는 집합이 아니라 리스트이므로 같은 수가 여러 번 나올 수 있다. 편의상 앞으로 집합 연산에 사용되는 연산자를 리스트 연산에도 사용하기로 한다. 실수 a 에 대해서, $list(v, P) \oplus a = \{a + b \mid b \in list(v, P)\}$ 로 정의한다. 즉, $list(v, P)$ 에 속한 모든 수에 a 를 각각 더하여 만든 리스트이다. $list(v, P)$

가 정렬이 되어 있으므로 $list(v, P) \oplus a$ 역시 정렬되어 있다. $list(v, P) = \emptyset$ 이면 $list(v, P) \oplus a = \emptyset$ 이다. \oplus 는 \cup 보다 우선순위가 높은 연산자이다.

3.2 $cent(P)$, $con(P)$ 와 $list(P)$ 계산

CT 를 구성하는 것은 앞 절에서 설명한대로 센트로이드를 이용한 분할 방법으로 할 수 있다. 이런 과정 중에 CT 의 각 컴포넌트 P 에 대해서 $cent(P)$ 와 $con(P)$ 도 모두 구할 수 있다. $list(P)$ 는 CT 를 아래서 위로 탐색하면서 계산한다. P 가 CT 의 리프 컴포넌트인 경우에는, CT 는 한 개의 에지만 가진다. P 가 에지 e 만 가지면 $list(P) = \{w(e)\}$ 이다.

P 가 리프 컴포넌트가 아닌 경우는, P 의 모든 후손 컴포넌트 Q 에 대해서 $list(Q)$ 가 이미 계산되어 있는 상태이다. $O(\log n)$ 개의 후손 컴포넌트들을 선택하여 이들의 $list(\cdot)$ 를 병합(merge)하면 $list(P)$ 를 구할 수 있다. 어느 후손 컴포넌트를 선택하는지를 설명한다. 표기를 간단히 하기 위하여 $c = con(P)$ 라 한다. c 의 P 에서의 차수에 따라 두 가지 경우로 나눌 수 있다 (T 에서의 차수가 아니라 P 에서의 차수를 고려한다).

- 1) c 의 차수가 1인 경우: c 의 차수가 1이므로 c 는 P 의 어느 후손 컴포넌트에 대해서도 그것의 센트로이드가 될 수 없다. P 의 후손 리프 컴포넌트 중에서 c 를 포함하는 것을 Q 라 한다. Q 는 유일하다.
- 2) c 의 차수가 2 이상인 경우: c 의 차수가 2이상이므로 c 는 P 의 후손 컴포넌트들에 대해서 그것의 센트로이드로 사용된다. P 의 후손 컴포넌트 중에서 그것의 센트로이드가 c 이면 P 에서 가장 가까운 컴포넌트를 Q 이라 하자. 여기서 가장 가까운 것은 CT 에서 P 와 Q 의 사이의 에지의 수가 작다는 것이다. Q 의 자식 컴포넌트 중 하나를 Q' 라 한다.

CT 에서 $P = Q_0$ 와 $Q = Q_m$ 를 잇는 경로 Q_0, Q_1, \dots, Q_m 을 고려하자 (Q_i 는 Q_{i-1} 의 자식 컴포넌트). $i = 1, \dots, m$ 에 대해 Q_i 의 형제 컴포넌트를 R_i 와 S_i 라 하자. Q_i, R_i, S_i 가 Q_{i-1} 의 자식 컴포넌트이므로, $i = 1, \dots, m$ 에 대해서

$$list(c, Q_{i-1}) = list(c, Q_i) \cup list(c, R_i) \cup list(c, S_i) \text{가 된다.}$$

따라서

$$list(c, Q_0) = list(c, R_1) \cup list(c, S_1) \cup \dots$$

$$\cup list(c, R_m) \cup list(c, S_m) \cup list(c, Q_m) \text{가 된다.}$$

모든 $i = 1, \dots, m$ 에 대해서

$$list(c, R_i) = list(con(R_i), R_i) \oplus w(\pi(con(R_i), c)) =$$

$$list(R_i) \oplus w(\pi(con(R_i), c)) \text{가 되고, 위와 비슷하게 써보면}$$

$$list(c, S_i) = list(S_i) \oplus w(\pi(con(S_i), c)) \text{가 된다. 또,}$$

$$list(c, Q_m) = list(Q_m) \oplus w(\pi(con(Q_m), c)) \text{가 된다.}$$

모든 $1 \leq i \leq m$ 에 대해서 $list(R_i)$ 와 $list(S_i)$ 는 각각 정렬된 리스트이고 이미 계산되어 있는 상태이므로 $list(c, R_i)$ 와 $list(c, S_i)$ 는 모두 선형 시간에 계산할 수 있고 이들 역시 정렬된 리스트이다. 또, $list(Q_m)$ 도 이미 계산되어 있으므로 $list(c, Q_m)$ 도 선형 시간에 구할 수 있다. 그러므로 $list(P) = list(c, Q_0)$ 는 $2m+1$ 개의 정렬된 리스트를 병합하여 구할 수 있다. $m = O(\log n)$ 이므로 $list(P)$ 는 $O(n \log \log n)$ 시간에 구할 수 있다[1].

[유의] 1) c 의 차수가 1인 경우, Q_m, R_m, S_m 모두 리프 컴포넌트여서 그 것들의 $list(\cdot)$ 가 모두 계산 되어 있다. 2) c 의 차수가 2 이상인 경우, $Q = Q_{m-1}$ 이고 $c = cent(Q_{m-1})$ 이어서 $c = con(Q_m) = con(R_m) = con(S_m)$ 이다. 따라서 $list(c, Q_m) = list(Q_m)$, $list(c, R_m) = list(R_m)$, $list(c, S_m) = list(S_m)$ 여서 이 세 리스트도 이미 계산되어 있다.

따라서 $list(T_1)$, $list(T_2)$, $list(T_3)$ 모두를 $O(n \log \log n)$ 시간에 구할 수 있다. 즉, 2절에서 배열 A 와 B 를 무계 순으로 정렬하는 일을 $O(n \log \log n)$ 시간에 할 수 있다.

[정리 1] n 개의 노드를 가지고 있고 각 에지마다 길이와 무게가 있는 트리 T 와 실수 L 이 주어질 때, T 에서 길이가 L 이하이면서 가장 무거운 경로를 $O(n \log n \log \log n)$ 시간에 찾을 수 있다.

[증명] [통합] 단계에서 A 와 B 를 $O(n \log \log n)$ 시간에 정렬할 수 있으므로 $Comline(n) = O(n \log \log n)$ 이 돼서 전체 알고리즘은 $T(n) = O(n \log n \log \log n)$ 시간이 걸린다.

4. 결 론

본 논문에서는 LHP 문제를 해결하는 알고리즘을 제시하였다. 먼저 제시한 알고리즘은 $O(n \log^2 n)$ 시간이 걸린다. 이는 [통합] 단계에서 배열을 정렬하는 시간 $O(n \log n)$ 이 필요하기 때문인데, 이의 수행 시간을 줄이기 위해 컴포넌트 트리라는 개념을 새로 만들었다. 컴포넌트 트리를 이용하여 $O(\log n)$ 개의 정렬된 리스트를 병합함으로써 배열 정렬이 $O(n \log \log n)$ 시간에 가능하게 되었다. 이를 이용하여 알고리즘의 수행 시간을 $O(n \log n \log \log n)$ 을 줄일 수 있었다. 앞으로의 과제는 알고리즘의 수행 시간을 더 줄일 수 있는 방법을 찾는 것이다.

참 고 문 헌

- [1] E. Horowitz, S. Sahni, and S. Anderson-Freed, Fundamentals of Data Structures in C, Computer Science Press, 1993.
- [2] H. Shen, Fast parallel algorithm for finding k-th longest path in a tree, Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference (APDC '97), IEEE, pp. 164-169, 1997.
- [3] B.Y. Wu, K.-M. Chao, and C.Y. Tang, An efficient algorithm for the length-constrained heaviest path problem on a tree, Information Processing Letters, vol.69, pp.63-67, 1999.

김 성 권



e-mail : skkim@cau.ac.kr

1981년 서울대학교 계산통계학과(학사)

1983년 한국과학기술원 전산학과(석사)

1990년 미국 University of Washington 전산학과(박사)

1983년 3월~1985년 9월 목포대학교 교수

1991년 3월~1996년 2월 경성대학교 교수

1996년 3월부터 중앙대학교 컴퓨터공학부 교수

관심분야: 알고리즘 및 정보보호