트리에서 가장 긴 비음수 경로를 찾는 직렬 및 병렬 알고리즘

(Sequential and Parallel Algorithms for Finding a Longest Non-negative Path in a Tree)

김 성 권 †
(Sung Kwon Kim)

요 약 각 에지에 무게(양수, 음수, 0 가능)가 주어진 트리에서, 경로의 에지들의 무게의 합이 비음수이면서 길이가 가장 긴 경로를 구하는 문제를 해결하고자 한다. 트리에서 가장 긴 비음수 경로를 찾는 $O(n \log n)$ 시간 직렬 알고리즘과 $O(\log^2 n)$ 시간과 $O(n)$개의 프로세서를 사용하는 CREW PRAM 병렬 알고리즘을 제시한다. 여기서, n은 트리가 가지는 노드의 수이다.

키워드 : 비음수 경로, 트리, 직렬 및 병렬 알고리즘

Abstract In an edge-weighted (positive, negative, or zero weights are possible) tree, we want to solve the problem of finding a longest path such that the sum of the weights of the edges in the path is non-negative. To find a longest non-negative path of a tree we present a sequential algorithm with $O(n \log n)$ time and a CREW PRAM parallel algorithm with $O(\log^2 n)$ time and $O(n)$ processors, where n is the number of nodes in the tree.

Key words : trees, non-negative paths, sequential and parallel algorithms

1. 서 론

배열 $X[1..n]$은 실수를 원소로 가지는 배열이다. 부분 배열 $X[i..j]$에 대해서$(1 \le i \le j \le n)$, 이것의 길이는 $j-i+1$이고 합은 $X[i]+\cdots+X[j]$이다. X의 부분 배열 중에서 합이 0 이상, 즉 비음수(non-negative)이면서 길이가 가장 긴 부분 배열을 찾는 일은 생물정보학 분야에서 중요한 응용을 가지고 있다[1,2]. DNA 시퀀스나 단백질 시퀀스를 문제 성격에 따라 적당한 실수 배열로 바꿔서 원하는 부분 배열을 찾는 것이다. 이 일은 [1,2]에 있는 알고리즘에 의해서 $O(n)$ 시간에 가능하다.

이를 일반화하여, 트리에 적용할 수 있다. 트리 $T=(V,E)$는 노드 집합 V와 에지 집합 E로 이뤄진다. 각 에지 $e \in E$에 무게(weight)라 부르는 실수 (양수, 음수, 0 모두 가능) $w(e)$가 주어진다. 두 개의 다른 노드를 잇는 경로 P를 고려하자. P에 있는 에지의 개수를 이 경로의 길이라 부른다. P의 에지들에 있는 무게를 모두 합한 것을 이 경로의 무게라 하고 $w(P)$로 표시한다. 즉, $w(P)=\sum_{e \in P} w(e)$가 된다. $w(P) \ge 0$이면 P는 비음수 경로라 부른다. 트리에서 가장 긴 비음수 경로를 찾는 것은 $O(n \log^2 n)$ 시간에 가능하고[3], 트리의 차수가 상수인 경우는 $O(n \log n)$ 시간에 가능하다[4]. 여기서 n은 T가 가지는 노드의 수다.

본 논문에서는 트리에서 가장 긴 비음수 경로(longest non-negative path)를 찾는 것이 목적이며, 이를 위해 $O(n \log n)$ 시간 직렬 알고리즘을 2절에서 제시하고, 3절에서는 $O(\log^2 n)$ 시간과 $O(n)$개의 프로세서를 사용하는 CREW PRAM 병렬 알고리즘을 제시한다. PRAM 모델에 대한 설명은 [5]에 있다.

2. 직렬 알고리즘

본 논문에서 제시하는 직렬과 병렬 알고리즘 모두 분할정복(divide-and-conquer)에 의한 재귀적(recursive) 구조를 가진다.

트리 $T=(V,E)$가 인접 리스트 방법으로 저장된다고 가정한다. $|V|=n$이고 $|E|=n-1$이다. 노드 u에 연결된 이웃 노드들의 집합을 $N(u)$로 표시한다. 즉,

· 본 논문은 2004년도 중앙대학교 학술연구비 지원을 받아 이뤄짐
† 종신회원 : 중앙대학교 컴퓨터공학과 교수
skkim@cau.ac.kr
논문접수 : 2006년 1월 19일
심사완료 : 2006년 9월 14일

$N(u)=\{v \mid (u,v)\in E\}$이다. T에서 에지(u,v)를 제거하면 두 개의 서브트리가 생기는데, 하나는 u를 포함하고 다른 하나는 v를 포함한다. 이때, u를 포함하는 서브트리의 크기(노드의 개수)를 $s(u,v)$로 표시한다. 따라서 u가 리프노드이면 $s(u,v)=1$이 되고, 에지(u,v)에 대해서 $s(u,v)+s(v,u)=n$이 성립한다.

만약 모든 $u\in N(x)$에 대해서 $s(u,x)\le n/2$가 성립하면 x를 T의 센트로이드 (centroid)라 부른다. (n이 짝수가 아니면 $\lceil n/2 \rceil$가 된다. 편의상 n이 짝수라 가정하자.) 모든 트리는 한 개 또는 두 개의 센트로이드를 가진다. 두 개를 가지는 경우는 이 둘은 반드시 이웃 한다[6]. T의 센트로이드는 $O(n)$ 시간에 구할 수 있다. 가장 널리 알려진 방법은 T를 임의의 노드를 루트로 하여 루트가 있는 트리로 바꾼 다음, 모든 노드 u에 대해서 $d(u)$를 계산한다. $d(u)$는 u와 u의 후손노드들로 이뤄진 서브트리의 크기이다. u가 리프노드이면 $d(u)=1$이고, u가 $k\ge 1$개의 자식노드 $v_1,\cdots v_k$를 가지면 $d(u)=1+\sum_{i=1}^{k} d(v_i)$이다. 트리를 포스트오더로 방문하면서 모든 노드에 대해서 $d(u)$를 계산한다. 이 과정 중에 처음으로 $d(u)\ge n/2$가 되는 노드 u가 T의 센트로이드가 된다.

트리에 분할정복 알고리즘을 적용하려면 트리를 재귀적으로 분할하는 방법이 필요하다. 센트로이드를 이용하여 T를 분할하는 방법을 설명하기 위하여 논문[7]에 나와 있는 소정리를 소개한다.

소정리 1: $n\ge 3$일 때, x가 T의 센트로이드이면 모든 $i=1,2,3$에 대해서 $\sum_{u\in N_i} s(u,x)\le n/2$가 되도록 $N(x)$를 세 개의 부분집합 N_1, N_2, N_3로 나눌 수 있다. N_3는 공집합이 될 수도 있다.

증명: 논문 [7]에 나와 있는 것을 옮긴다. $N(x)=\{v_1,\cdots,v_k\}$라 가정하고, $i=1,\cdots,k$에 대해서 $s_i=s(v_i,x)$라 한다. x가 센트로이드이므로 모든 i 대해 $s_i\le n/2$이다. $s_1+\cdots+s_p>n/2$가 되는 최소 p를 구한다. 또, $s_p+s_{p+1}+\cdots+s_q>n/2$가 되는 최소 $q>p$를 구한다. 그리고 $N_1=\{v_1,\cdots,v_{p-1}\}$, $N_2=\{v_p,\cdots,v_{q-1}\}$, $N_3=\{v_q,\cdots,v_k\}$로 놓으면 조건을 모두 만족한다. 만약 $s_1+\cdots+s_{q-1}=n-1$이면 $N_3=\varnothing$이다. □

소정리 1의 N_1, N_2, N_3를 이용하여 T를 세 개의 트리 T_1, T_2, T_3로 나눈다. T를 x를 루트로 하는 트리로 생각할 때, T_i는 x를 포함하고, x와 N_i의 노드를 연결하는 에지들을 포함하고, 또 N_i에 있는 노드를 루트로 하는 서브트리들을 (총 $|N_i|$ 개) 포함한다. 모든 T_i가 x를 포함하고, x를 제외하면 T_i들은 서로 중복되는 곳이 없다. 따라서 $n_i=|T_i|$라 하면, $n_i\le n/2+1$이고 $n_1+n_2+n_3=n+2$이다.

제시하려는 직렬 알고리즘은 앞에서 언급했듯이 분할 정복 방법으로 수행한다.

입력: 트리 T.

출력: T의 가장 긴 비음수 경로의 길이 (아래 알고리즘을 조금 고치면 경로 자체를 구할 수 있으므로, 편의상 길이만 구하는 것으로 한다.)

[분할] T가 하나의 노드로만 이뤄진 경우는, 0을 반환한다. T가 두 노드와 이들을 잇는 에지 e로만 이뤄진 경우는, $w(e)\ge 0$이면 1을 아니면 0을 반환한다. 그 외 경우는, T의 센트로이드 x를 구하고, 소정리 1의 방법을 이용하여 세 개의 트리 T_1,T_2,T_3를 만든다.

[정복] $i=1,2,3$에 대해서 T_i 내에서 가장 긴 비음수 경로의 길이 L_i를 재귀적으로 구한다.

[통합] 이제, T에서 x를 지나면서 T_i와 T_j $(i\ne j)$에 걸쳐 있는 경로들 중에서 가장 긴 비음수 경로의 길이 L_x를 구한다. $\max\{L_1,L_2,L_3,L_x\}$를 구하면 T의 가장 긴 비음수 경로의 길이로서 우리가 원하는 답이 된다.

$W(n)$을 위 알고리즘이 n개의 노드를 가진 트리에서 가장 긴 비음수 경로의 길이를 구하는데 필요한 최악의 경우의 수행시간이라 하자. $n=1,2$에서 $W(n)=O(1)$이고, $n\ge 3$에 대해서

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

이 성립한다. 여기서, $i=1,2,3$에 대해 $n_i\le n/2+1$이고, $n_1+n_2+n_3=n+2$이며, Divide(n)은 [분할] 단계에 필요한 시간이고, Combine(n)은 [통합] 단계에 필요한 시간이다. [분할] 단계에서 x를 구하는 것과 T_i들을 구하는 것은 모두 선형 시간에 가능하므로 $\mathrm{Divide}(n)=O(n)$이다. 앞으로 $\mathrm{Combine}(n)=O(n)$임을 보이면 $W(n)=O(n\log n)$이 되어서 우리가 원하는 결과를 얻는다.

[통합] 단계에서 하는 일을 설명하기 위하여 $L_x^{i,j}$를 양 끝점이 모두 x가 아니면서 하나의 끝점은 T_i에 있고 다른 하나는 T_j $(i\ne j)$에 있는 경로들 중에서 가장 긴 비음수 경로의 길이라 정의한다. 모든 $(i,j)\in\{(1,2),(1,3),(2,3)\}$에 대해서 $L_x^{i,j}$를 구하면 그 중 최대가 L_x가 된다. 즉, $L_x=\max\left\{L_x^{1,2},L_x^{1,3},L_x^{2,3}\right\}$이다. 여기서는 $L_x^{1,2}$를 선형시간에 구하는 방법만 설명한다. 다른 두 개도 비슷한 방법으로 구할 수 있다.

$L_x^{1,2}$를 구하는 방법은 논문 [4]에 나와 있는 방법을 사용한다. [4]의 방법이 다음 절의 병렬 알고리즘에도

사용되므로 여기에 설명한다. 먼저, T_1에서 $A[i]$ $(i=1,\cdots,n_1)$를 구한다. $A[i]$는 x가 시작점인 길이 i인 (즉, i개의 에지를 이용하여 갈 수 있는) 모든 경로 중에서 무게가 가장 큰 경로의 무게이다. 만약 길이 i인 경로가 T_1에 없으면 $A[i]=-\infty$로 한다. T_1을 프리오더 순으로 방문하면서 $O(n_1)$ 시간에 모든 $A[i]$들을 계산할 수 있다. 비슷하게 T_2에서 $B[j]$를 x를 시작점으로 가지는 길이 j인모든 경로 중에서 무게의 합이 가장 큰 경로의 무게로 정의 한다 $(j=1,\cdots,n_2)$. 모든 $B[j]$들도 $O(n_2)$ 시간에 구할 수 있다. 두 배열 A와 B로부터 [4]에 있는 다음 소정리는 쉽게 유추할 수 있다.

소정리 2: $L_x^{1,2}=\max\{\{i+j \mid A[i]+B[j] \geq 0\} \cup \{0\}\}$

소정리 2를 이용하여 [4]에서는 $L_x^{1,2}$를 $O(n_1+n_2)$시간에 구하는 방법을 보여 주고 있다. $L_x^{1,3}$과 $L_x^{2,3}$도 비슷한 방법을 이용하여 선형시간에 구할 수 있으므로, L_x를 역시 선형시간에 구할 수 있다. 그러므로 $\text{Comtine}(n)=O(n)$이다.

정리 1: n개의 노드를 가지는 트리에서 가장 긴 비음수 경로는 $O(n\log n)$ 시간에 구할 수 있다.

참고: [4]의 방법과 본 논문의 방법 모두 센트로이드를 이용한 분할 방법을 사용하지만, 차이점은 [4]에서는 분할 후 생기는 서브트리의 수가 x의 차수와 같은 반면, 본 논문의 방법 (소정리 1)은 x의 차수와 무관하게 서브트리가 최대 세 개 생긴다는 것이다. 따라서 계산해야 할 $L_x^{i,j}$의 수가 세 개로 제한되어 [통합] 단계의 시간이 $O(n)$이 되므로, 우리가 원하는 수행시간을 얻을 수 있다.

3. 병렬 알고리즘

본 절에서는 앞 절에서 제시한 직렬 알고리즘을 CREW PRAM 상에서 $O(n)$개의 프로세서를 이용하여 $O(\log^2 n)$ 시간에 구현할 수 있음을 보인다. 2절의 직렬 알고리즘은 분할정복 방법을 사용하는 재귀적 알고리즘으로 최대 $O(\log n)$ 번의 리커젼을 반복한다. 따라서 [분할]과 [통합] 단계의 일을 $O(\log n)$ 시간에 $O(n)$개의 프로세서를 사용하여 할 수 있음을 보이면, 전체적으로 $O(\log^2 n)$ 시간과 $O(n)$개의 프로세서를 사용하게 된다.

알고리즘 설명에 앞서 병렬 알고리즘에 사용되는 기본 기법을 소개한다. 이 기법들 모두 CREW PRAM에서 $O(n)$개의 프로세서로 $O(\log n)$시간에 수행할 수 있다. 자세한 사항은 [5]를 참조하면 된다.

- 전위합(prefix sum) 계산: 배열 $X[1..n]$가 주어질 때, 모든 $1 \leq i \leq n$에 대해서 전위합 $S[i]=X[1]+\cdots+X[i]$를 계산한다.
- **리스트 랭킹(list ranking):** 길이가 n인 연결리스트 (linked list)가 주어질 때, 각 노드의 랭크 (rank)를 계산한다. 연결리스트는 n개의 노드로 구성되며 각 노드는 다음 노드를 가리키는 포인터를 가지고 있고 마지막 노드의 포인터는 null이다. 또 첫 노드를 가리키는 포인터를 별도로 가진다. 노드의 랭크는 그 노드가 연결리스트에서 몇 번째 노드인가를 나타낸다. 예를 들어, 첫 노드의 랭크는 1이고 마지막 노드의 랭크는 n이다.
- **ANSV(all nearest smaller values) 계산:** 배열 $X[1..n]$가 주어질 때, 모든 $1 \leq i \leq n$에 대해서 $\text{NEXT}(i)$를 계산한다. $\text{NEXT}(i)=\min\{i' \mid i<i'$ and $X[i]>X[i']\}$이다. 즉, $X[i]$에서 i가 커지는 방향으로 갈 때 처음으로 $X[i]$ 보다 작은 것의 인덱스가 $\text{NEXT}(i)$이다. $X[n+1]=-\infty$라 가정한다.

[분할] 단계에서 하는 일은 센트로이드 x를 구하는 것과 소정리 1을 이용하여 T_1, T_2, T_3를 구하는 것이다. x를 구하는 것은 $O(\log n)$ 시간에 $O(n)$ 개의 프로세서를 이용하여 구할 수 있다[5]. 또 소정리 1의 수열 $s_1,\cdots,s_k$를 배열로 바꾼 후 전위합 계산을 이용하면 p와 q를 구할 수 있다. 이를 이용하여 N_i를 구하고 T_i를 만든다 $(i=1,2,3)$. [분할] 단계의 모든 일은 $O(n)$개의 프로세서를 이용하여 $O(\log n)$ 시간에 가능하다.

[통합] 단계에서 하는 중요한 일은 $L_x^{i,j}$를 구하는 것이다 $((i,j)\in\{(1,2),(1,3),(2,3)\})$. 여기서는 $L_x^{1,2}$에 대해서만 설명한다. 그러기 위해서는 먼저 $A[i]$ $(i=1,\cdots,n_1)$와 $B[j]$ $(j=1,\cdots,n_2)$를 구해야 한다. T_1에서 $A[i]$들은 다음처럼 구할 수 있다. T_1의 모든 노드 u에 대해서 $pre(u)$, $level(u)$, $weight(u)$를 계산한다. $pre(u)$는 u의 프리오더 번호이다. 즉, T_1을 프리오더로 방문할 때 u가 몇 번째로 방문되는가를 나타낸다. $level(u)$과 $weight(u)$는 x와 u를 잇는 경로의 길이 (u의 레벨)와 무게를 각각 나타낸다. T_1의 모든 노드들에 대해서 이 세 값을 계산하는데 $O(\log n_1)$시간과 $O(n_1)$개의 프로세서로 충분하다[5].

길이 n_1인 배열 $P[1..n_1]$을 마련한다. $P[i]$는 $P[i].level$과 $P[i].weight$로 구성된다. 각 노드 $u\in T_1$에 대해서, $P[pre(u)].level=level(u)$과 $P[pre(u)].weight=weight(u)$로 놓는다. P를 $P[i].level$가 증가하는 순으로 정렬한다. 그러면 레벨이 같은 노드들이 연속적으로 위치한다. P_j를 레벨 j인 노드들로 구성된 P의 부분 배열이라 하자 ($j=1,\cdots,h$, h는 T_1의 높이). 각 P_j에서

*weight*가 최대인 항을 골라서 $A[j]$에 기록한다 $(j=1,\cdots,h)$. $j=h+1,\cdots,n_1$에 대해서는 $A[j]=-\infty$로 한다. 정렬은 $O(\log n_1)$시간과 $O(n_1)$개의 프로세서로 할 수 있으며 [5], P_j에서 최대를 구하는 것은 $O(\log|P_j|)$ 시간에 $O(|P_j|)$개의 프로세서로 할 수 있다[5]. 따라서 모든 $A[i]$들을 $O(\log n_1)$시간에 $O(n_1)$개의 프로세서를 사용하여 계산할 수 있다.

T_2에서 $B[j]$ $(j=1,\cdots,n_2)$를 구하는 것도 비슷한 방법으로 $O(\log n_2)$시간에 $O(n_2)$개의 프로세서를 사용하여 할 수 있다.

이제 소정리 2의 $L_x^{1,2}=\max\{\{i+j\mid A[i]+B[j]\geq 0\}\cup\{0\}\}$를 이용하여 $L_x^{1,2}$를 구한다. 이를 위해 [4]에 있는 것처럼 $j=1,\cdots,n_2$에 대해서 $r(j)$를 아래처럼 정의한다.

$A[r(j)]+B[j]\geq 0$이고 모든 $r(j)<i\leq n_1$에 대해서 $A[i]+B[j]<0$이다

만약 모든 $1\leq i\leq n_1$에 대해서 $A[i]+B[j]<0$이면, $r(j)=0$이다.

$r(j)$는 j가 고정된 상태에서 $A[i]+B[j]\geq 0$이 되는 가장 큰 i를 나타낸다. 그러므로 소정리 1을 다시 쓰면 $L_x^{1,2}=\max\{\{r(j)+j\mid 1\leq j\leq n_2\}\cup\{0\}\}$가 된다. 따라서 모든 j에 대해서 $r(j)$를 구하면 $L_x^{1,2}$를 구할 수 있다.

$r(j)$들을 구하기 위하여, $a_0,\cdots,a_m$을 다음처럼 정의한다. $A[0]=\infty$라 가정한다

- $a_0=0$.
- $k\geq 1$에 대해서, a_k는 $A[a_k]=\max\{A[i]\mid a_{k-1}<i\leq n_1\}$가 되는 인덱스이다. 즉, $A[a_{k-1}+1],\cdots,A[n_1]$ 중에서 최대가 되는 것의 인덱스가 a_k이다. 만약 최대가 여러 개 있으면 인덱스가 가장 큰 것을 택한다.
- 이렇게 진행하여 $a_k=n_1$이 되면 멈추고, 그 때 $m=k$가 된다.

당연히 $A[a_0]>\cdots>A[a_m]$이다.

소정리 3: $r(j)$는 $A[a_k]\geq -B[j]>A[a_{k+1}]$를 만족하는 a_k이다.

증명: 위의 부등식을 다시 쓰면 $A[a_k]+B[j]\geq 0$이고 $A[a_{k+1}]+B[j]<0$이다. $A[a_{k+1}]$가 $A[a_k+1],\cdots,A[n_1]$ 중에서 최대이므로, 모든 $a_k<i\leq n_1$에 대해서 $A[i]+B[j]<0$이 된다. $r(j)$의 정의에 따라서 $r(j)=a_k$이다. □

소정리 3을 이용하여 $-B[j]$를 가지고 $A[a_0],\cdots,A[a_m]$에서 이진탐색을 하면 $r(j)$를 구할 수 있다. $a_0,\cdots,a_m$의 정의를 직접적으로 사용하여 그들을 구하는 빠른 병렬 방법을 찾는 것은 어려워 보이므로, 이를 해결하기 위하여 $b_0,\cdots,b_t$를 다음처럼 정의한다.

- $b_0=n_1$.
- $k=1,2,\cdots$에 대해서, $b_k=\max\{i\mid i<b_{k-1}\text{ and }A[i]>A[b_{k-1}]\}$이다. 즉, b_{k-1}에서 왼쪽으로 (i가 작아지는 쪽으로) 갈 때 처음으로 $A[i]>A[b_{k-1}]$가 되는 i가 b_k이다.
- 이렇게 진행하여 $b_k=0$이 되면 멈추고, 그 때의 k를 t라 한다. 앞에서 $A[0]=\infty$라 가정하였다.

$A[b_0]<\cdots<A[b_t]$는 당연히 성립하며, 아래 소정리는 $a_0,a_1,\cdots,a_m$과 $b_t,b_{t-1},\cdots,b_0$가 동일함을 보여준다. 즉, $m=t$이고 모든 $k=0,\cdots,m$에 대해서 $a_k=b_{m-k}$이다.

소정리 4: 모든 $0\leq k<t$에 대해서 $A[b_k]=\max\{A[i]\mid b_{k+1}<i\leq n_1\}$이다.

증명: $k=0$일 때는 b_1의 정의에 따라 $A[b_1]>A[b_0]$이고 $A[b_1+1],\cdots,A[b_0-1]$들은 모두 $A[b_0]$보다 작기 때문에 $A[b_0]=\max\{A[b_1+1],\cdots,A[b_0]\}$인 것은 확실하다. 모든 $0<k<t$에 대해서 $A[b_k]=\max\{A[i]\mid b_{k+1}<i\leq n_1\}$가 성립하면 소정리는 이미 증명된 것이다. 따라서 이것이 성립하지 않는 k가 있다고 가정하고, 그런 k들 중에서 최소를 k'이라 하자 또, c를 $A[c]=\max\{A[i]\mid b_{k'+1}<i\leq n_1\}$가 되는 가장 큰 인덱스라 하자. 그러면 $c<b_{k'}$이 된다. $A[c]>A[b_{k'}]$이므로 $b_{k'+1}\geq c$가 되어야 한다. 이는 $b_{k'+1}<c$에 모순된다. 따라서 그런 k'은 존재하지 않는다. □

소정리 4를 이용하여 $b_0,\cdots,b_m$들을 병렬로 구한다. 먼저, 모든 $1\leq i\leq n_1$에 대해서 $l(i)=\max\{i'\mid i'<i\text{ and }A[i']>A[i]\}$를 구한다. i에서 왼쪽으로 갈 때 처음으로 $A[i']>A[i]$가 되는 i'이 $l(i)$가 된다. 그러면, n_1, $l(n_1)$, $l(l(n_1))$, …으로 $l(\cdot)$를 따라가면, $b_0,\cdots,b_m$을 얻는다. 즉, $b_0=n_1$이고, $0<k\leq m$에서는 $b_k=l(b_{k-1})$가 된다. 모든 i $(i=1,\cdots,n_1)$에 대해서 $l(i)$를 구하는 것은 ANSV 계산을 변형하여 이용하면 되고, $l(i)$들로부터 $b_0,\cdots,b_m$를 구하는 것은 리스트 랭킹을 이용하면 된다[5]. 모두 $O(n_1)$개의 프로세서를 사용하여 $O(\log n_1)$ 시간에 가능하다.

배열 $A'[0..m]$을 마련한 후, 모든 $0\leq k\leq m$에 대해서 $A'[k]=A[b_{m-k}]$를 한다. 각 j $(1\leq j\leq n_2)$마다 하나의 프로세서를 배당하고 그 프로세서가 $-B[j]$를 가지고 A'에서 이진탐색을 하여 $r(j)$를 구한다. 그런 후 $L_x^{1,2}=\max\{\{r(j)+j\mid 1\leq j\leq n_2\}\cup\{0\}\}$를 계산한다. 이

를 위해 필요한 시간은 $O(\log n)$이고 필요한 프로세서의 수는 $O(n_2)$이다.

지금까지의 설명을 종합하면 $L_x^{1,2}$를 $O(\log n)$ 시간에 $O(n_1+n_2)$개의 프로세서를 사용하여 구할 수 있다. $L_x^{1,3}$과 $L_x^{2,3}$도 각각 $O(n_1+n_3)$개와 $O(n_2+n_3)$개의 프로세서를 사용하여 $O(\log n)$ 시간에 구할 수 있다. 따라서 [통합] 단계의 일은 $O(\log n)$ 시간에 $O(n)$개의 프로세서로 가능하다.

정리 2: n개의 노드를 가지는 트리에서 가장 긴 비음수 경로는 $O(n)$개의 CREW PRAM 프로세서를 사용하여 $O(\log^2 n)$ 시간에 구할 수 있다.

증명: [분할]과 [통합] 단계는 모두 $O(\log n)$ 시간에 가능하므로 전체 알고리즘의 수행시간은 $O(\log^2 n)$이 된다. 사용하는 프로세서는 모두 $O(n)$개가 되는 것은 자명하다. 그리고 사용하는 모든 병렬 기법은 [5]에 나와 있듯이 모두 CREW PRAM에서 가능하다. □

4. 결 론

본 논문에서는 n개의 노드를 가지는 트리에서 가장 긴 비음수 경로를 찾는 직렬과 병렬 알고리즘을 제시하였다. 앞으로의 과제는 직렬 알고리즘과 병렬 알고리즘 모두 수행 시간을 줄이는 것이고, 병렬 알고리즘의 경우는 사용하는 프로세서의 수를 줄이는 것도 관심을 둘만 한 문제이다.

참 고 문 헌

[1] L. Allison, Longest biased intervals and longest non-negative sum intervals, *Bioinformatics*, vol. 19(10), pp. 1294-1295, 2003.

[2] L. Wang and Y. Xu, SEGID: Identifying interesting segments in (multiple) sequence alignments, *Bioinformatics*, vol. 19(2), pp. 297-298, 2003.

[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.

[4] S.K. Kim, Finding a longest nonnegative path in a constant degree tree, *Information Processing Letters*, vol. 93, no. 6, pp. 275-279, March 2005.

[5] J. JaJa, An Introduction to Parallel Algorithms, Addison-Wesley, 1992.

[6] D.E. Knuth, The Art of Programming, Vol. 1. Fundamental Algorithms, 2nd Edition, Addison-Wesley, 1973.

[7] H. Shen, Fast parallel algorithm for finding kth longest path in a tree, Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference (APDC '97), IEEE, pp. 164-169, 1997.

김 성 권
1981년 서울대학교 계산통계학과 졸업
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. 관심분야는 알고리즘 및 정보보호