제한된 오프라인 경쟁삭제 문제를 해결하기 위한 AT^2 최적의 재구성 가능 메쉬 알고리즘

이 광 의†

요 약

제한된 경쟁삭제 문제는 집합연산 문제의 간단한 형태의 하나로서, 경쟁삭제 문제에서 삽입연산을 제외한 것이다. 현재 경쟁삭제 문제에 대해서는 최적의 순차 알고리즘이 개발되어 있으며, 병렬 알고리즘의 경우 O(n/loglogn)개의 처리기를 사용하여 $O(\log^2 n \log\log n)$에 수행되는 알고리즘이 개발되어 있다. 본 논문에서는 제한된 경우의 경쟁삭제 문제를 해결하기 위한 재구성 가능 메쉬 알고리즘을 제시한다. 제안된 알고리즘은 n×n 재구성 가능 메쉬에서 상수시간에 수행되며, 이 결과는 AT^2 최적이다.

An AT^2 Optimal Reconfigurable Mesh Algorithm for The Constrained Off-line Competitive Deletion Problem

Kwang Eui Lee†

ABSTRACT

The constrained off-line competitive deletion problem is a simple form of the set manipulation operations problem. It excludes the insertion operation from the off-line competitive deletion problem. An optimal sequential algorithm and a CREW PRAM algorithm which runs $O(\log^2 n \log\log n)$ time using O(n/loglogn) processors were already presented in the literature. In this paper, we present a reconfigurable mesh algorithm for the constrained off-line competitive deletion problem. The proposed algorithm is executed in a constant time on an n× n reconfigurable mesh, and the result is AT^2 optimal.

키워드 : 집합연산문제(Set Manipulation Operations Problem), 재구성 가능 메쉬(Reconfigurable Mesh)

1. 서 론

제한된 경쟁삭제 문제는 초기에 주어진 집합 S와 그 집합에서 원소들을 삭제하는 연산들로 구성된다. 즉, 초기에 원소들이 포함되어 있는 집합과, 연산의 순서열이 주어지게 되는데 이 연산의 순서열은 ExtractMin 연산과 Delete(x) 연산으로 구성된다. ExtractMin 연산(E로 표기하고, "최소삭제연산"이라 한다)은 집합 S에서 최소의 원소를 삭제하는 것으로 집합 S가 공집합이 아니라면 최소의 원소가 E에 대한 응답이 되고 집합 S에서 최소의 원소는 삭제되며, 만약 S가 공집합이라면 응답은 null이 된다. Delete(x) 연산(D(x)로 표기)은 집합 S에서 x를 삭제하는 연산으로 집합 S에 x가 존재하면 그 원소를 삭제하고 x를 응답으로 내며, 집합 S에 x가 존재하지 않으면 응답은 null이 된다.

제한된 오프라인 경쟁삭제 문제는 초기 집합 S와 삭제연산들의 순서열을 미리 주어지고, 연산들을 순서열의 순서대로 적용하였을 때 각 연산에 대한 응답과 모든 연산이 적용된 후에 남아있는 집합의 내용을 결정하는 문제이다.

제한된 오프라인 경쟁삭제 문제의 순차 알고리즘은 균형 잡힌 이진 탐색 트리(balanced binary search tree)와 힙(heap)을 구성하고, 이들 사이에 상호참조 링크를 유지함으로서 O(nlogn) 시간에 해결할 수 있다. 초기집합 $S = \{S_0, S_1, \ldots, S_{n-1}\}$에 대하여, n개의 최소삭제연산을 갖는 제한된 오프라인 경쟁삭제 문제의 결과는 집합 S를 정렬하는 것과 동일하므로 제한된 오프라인 경쟁삭제 문제의 하한은 $\Omega(n\log n)$이 되고, 따라서 앞의 순차 알고리즘의 결과는 최적이다. 병렬 알고리즘의 경우 1988년에 O(n/loglogn)개의 CREW 처리기를 이용하여 $O(\log^2 n \log\log n)$ 시간에 해결

† 정 회 원 : 동의대학교 멀티미디어공학과 교수
논문접수 : 2002년 1월 25일, 심사완료 : 2002년 4월 17일

하는 PRAM 알고리즘이 M. J. Atallah, M. T. Goodrich, S. R. Kosaraju등에 의해서 제시되었다[1].

집합연산 문제에 대한 재구성 메쉬에서의 기존의 연구결과는 1997년에 집합에 S에 원소 x를 삽입하는 Insert(x) 연산과 Delete(x) 연산을 포함하는 이진탐색트리 문제와 Insert(x)연산과 ExtractMin 연산을 포함하는 우선순위큐 문제가 K. E. Lee, J. Chang 등에 의해서 $n \times n$의 재구성 메쉬상에서 상수시간에 해결된 결과가 있다[2, 3]. 이 결과는 주어진 문제를 해결하기 위한 하드웨어 구성에서, 필요한 정보이동의 양에 의한 평가척도 AT^2를 기준으로 볼 때 최적이다. 이때 A는 하드웨어의 면적을, T는 수행시간을 나타낸다[4].

본 논문에서는 제한된 오프라인 경쟁삭제 문제 즉, Delete(x)와 ExtractMin 연산을 포함하는 집합연산 문제를 $n \times n$의 재구성 가능 메쉬(앞으로 재구성 메쉬라 하겠다)에서 상수시간에 해결하는 방법을 제시한다. 이 결과는, 정렬을 상수시간에 실행하기 위해서는 최소한 $n \times n$의 재구성 메쉬가 필요하며[5], 정렬이 제한된 오프라인 경쟁삭제 문제로 간단히 축약됨을 볼 때 AT^2최적이다.

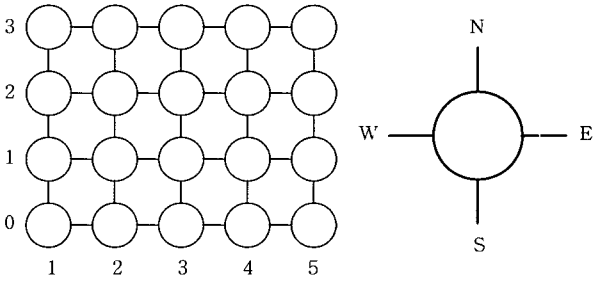
본 논문의 2절에서는 재구성 메쉬의 구조 및 재구성 메쉬상에서의 기존 연구결과에 대해 기술하고, 3절에서는 제한된 오프라인 경쟁삭제 문제에 대해서, 마지막으로 4절에서 결론과 함께 향후 연구방향에 대해서 알아보도록 하겠다.

2. 재구성 메쉬의 구조 및 연구결과

크기 $n \times n$의 재구성 메쉬는 $n \times n$의 메쉬 연결구조에 재구성 가능한 버스를 추가한 구조로서, (그림 1)에서 보여지는 것은 4×5의 이차원 재구성 메쉬이다. 이차원 재구성 메쉬의 경우 각각의 처리기는 네 개의 포트(port)를 갖는데 각각의 이름은 E(East), W(West), N(North), S(South)이고, 각각의 포트는 순서대로 처리기의 오른쪽, 왼쪽, 위쪽, 그리고, 아래쪽 처리기와 연결된다. 우리는 이러한 포트에 대해서, 각 처리기가 포트들에 대한 내부연결을 변경 함으로서 동적인 버스를 구성할 수 있다[6].

앞으로의 표기에서 $R[i,j]$는 i번째 행과 j번째 열이 교차하는 위치에 있는 처리기를 의미한다. $R[i, p \cdots q]$는 처리기 $R[i,p]$, $R[i,p+1], \cdots, R[i,q]$를 의미하며, $R[x \cdots y, p \cdots q]$는 $R[x, p \cdots q]$, $R[x+1, p \cdots q], \cdots, R[y, p \cdots, q]$의 처리기들을 의미한다. $R[x \cdots, y, p \cdots q]$.W는 $R[x \cdots y, p \cdots q]$의 처리기들의 W 포트를 의미하며, $s[i,j]$는 처리기 $R[i,j]$에 있는 지역변수 s를 의미한다. 본 논문에서 사용하는 재구성 메쉬의 버스 폭은 $\log n$ 비트로 한다.

재구성 메쉬에서는 많은 문제들이 상수시간에 해결되고 있는데 본 논문의 알고리즘에서 사용되는 몇몇 결과들을 보이겠다.



(그림 1) 이차원 재구성 메쉬의 구조

보조정리 2.1. 길이가 n인 0/1순서열 $a = \langle a_i \rangle$, $a_i = 0$ or 1, $0 \le i \le n-1$에 대한 전위합은 $n \times n$의 재구성 가능한 메쉬에서 상수시간에 계산 가능하다. 이때 입력 a_i는 $R[0,i]$에 주어지고 출력 $c_i = a_0 + a_1 + \cdots + a_i$는 $R[0,i]$에 저장된다[7].

보조정리 2.2. $n \times n$의 재구성 메쉬의 하나의 행에 주어진 n개의 수는 상수시간에 정렬 가능하다. 이때 출력도 하나의 행에 저장될 수 있다[5].

보조정리 2.3. $n \times n$의 재구성 메쉬의 하나의 행에 분포되어 있는 $m(<n)$개의 수는 주어진 행에서의 원래의 순서를 유지하면서 동일한 행의 0번째 열부터 m-1번째 열로 상수시간에 이동 가능하다. 이를 데이터 압축(data compaction)이라 한다[5].

이 이외에 방송(broadcast), 순위정하기(ranking), 이동(shift), 연속합(consecutive sum), 정렬(sorting), 임의접근읽기(random access read), 임의접근쓰기(random access write), 순열(permutation) 등의 자료이동에 관한 문제와 계산기하학에서의 많은 문제에 대한 상수시간 알고리즘이 개발되어 있다[5, 6].

3. 제한된 오프라인 경쟁삭제

제한된 오프라인 경쟁삭제 문제의 입력은 D(x)와 E로 구성된 집합연산의 순서열 $O = \langle O_i \rangle$, $O_i = (\text{opr}, \text{val})$, $0 \le i \le n-1$와 이러한 연산들이 적용될 대상인 $S = \{s_0, s_1, s_2, \cdots, s_{m-1}\}$로 주어진다. 이때, 각 s_i는 정수이고, 이들은 정렬된 상태로 주어짐을 가정한다. 즉, $i < j$일 때 $s_i \le s_j$이다. 연산 O_i는 연산의 종류를 나타내는 opr과 값을 나타내는 val로 구성되며, O_i.opr에 따라 다음의 두 종류로 분류된다.

$O_i.opr = D$: 집합 S에서 x를 삭제하는 연산, 이때 x는 $O_i.val$에 주어진다.

$O_i.opr = E$: 집합 S에서 최소값을 삭제하는 연산, 이때 $O_i.val$은 null을 갖는다

앞으로의 표기에서 E_i는 연산의 순서열 O에서 최소삭제 연산인 E 연산 중 i+1번째 E 연산을 의미하고, D_i는 연산의 순서열 O에서 D 연산 중 i+1번째 D 연산을 의미한다. 이때 초기입력으로 s_i는 지역변수 s[i,0]에 저장되어 있고, O_j는 지역변수 o[0,j]에 저장되어 있음을 가정하자.

다음의 전처리 과정을 통하여 우리는 일반성을 잃지 않고, 집합 S의 모든 원소는 유일하며, 집합 S의 원소 x′에 대하여 연산의 순서열에 D(x′)가 존재하고, 역으로 연산 D(x″)에 대하여 집합 S에 원소 x″가 존재함을 가정 할 수 있다.

먼저 집합 S에 s_i부터 s_{i+p-1}까지의 p개의 x′이 존재하고, $D_{f(0)}(x')$부터 $D_{f(q-1)}(x')$까지 q개의 D(x′)이 존재하는 경우를 고려하여 보자. 이때 f(i)는 순서열 O에서 i번째 D(x′)의 인덱스를 계산하는 함수이다. 여기서 $q <= p$임을 가정할 수 있는데 만약 $q > p$라면, $D_{f(p)}(x')$부터 $D_{f(q-1)}(x')$은 삭제에 실패하고 null 값을 가지므로 전처리 과정에서 데이터 압축을 이용하여 제거할 수 있다. 이제 $s_{i-1} < s_i < s_{i+1} < \cdots < s_{i+p-1} < s_{i+p}$가 되도록 적당히 s_i부터 s_{i+p-1}까지의 값을 변경한다. 다음 $D_{f(j)}(x')$가 s_{i+p-j}를 삭제하도록 $D_{f(0)}(x')$부터 $D_{f(q-1)}(x')$까지의 x′ 값들을 변경한다. 이러한 변화가 연산의 결과에 영향을 주지 않으며, 전위합 연산을 통하여 $n \times n$의 재구성 메쉬에서 상수시간에 수행가능 함을 알 수 있다.

앞의 과정이 수행되면, 집합 S의 원소 x′에 대하여 최대 하나의 D(x′)을 가질 수 있다. 방송을 통하여 간단히 D(x′) 쌍을 갖지 않는 x′를 찾을 수 있으며, 순서열의 맨 뒤에 대응되는 D(x′)들을 추가하여 모든 x′에 대하여 D(x′)가 꼭 하나씩 존재하도록 할 수 있다. 이렇게 추가되는 원소는 문제의 입력크기를 증가시키지만 이러한 D(x′)들은 실제로 삽입될 필요가 없음을 뒤에서 보이도록 하겠다. 따라서 문제의 크기는 증가하지 않는다.

제한된 오프라인 경쟁삭제 문제에서 우리가 해결해야 하는 것은 각 E 연산과 각 D(x) 연산에 대한 응답과 최종적으로 남는 집합 S의 내용이 된다. 이제 집합 S의 원소 s_i에 대하여 연산의 순서열 O에서 $D(s_i)$앞에 위치하는 E 연산의 개수를 $ne(s_i)$라 하자. 초기집합에 주어진 원소 s_i에 대해서, 이 원소는 E_0부터 $E_{ne(si)}$의 연산에 의해서 삭제되지 않는다면 $D(s_i)$ 연산에 의해서 삭제됨을 알수 있다. 이러한 관찰로부터 우리는 오프라인 경쟁삭제 문제를 다음과 같이 세 개의 단계로 해결하고자 한다. 첫번째 단계에서 모든 s_i에 대해서 $ne(s_i)$를 구한다. 다음 단계로 E 연산에 대한 응답을 구하고, 마지막 단계로 $D(s_i)$ 연산에 대한 응답과 최종적으로 얻어지는 집합을 구한다. 다음의 소절들에서 각 단계의 계산과정을 기술한다.

3.1 S의 원소 x에 대한 ne(x)의 계산

정의에 따라 ne(x)는 집합연산의 순서열에서 D(x) 연산에 선행하는 E 연산의 개수이다. 이는 재구성 메쉬에서 전위합을 이용해서 다음과 같이 상수시간에 계산 가능하다.

```
Algorithm ComputeNE

/* o[0,i]에 Oi가 저장되어 있음을 가정한다. */
단계 1 : o[0,i]연산이 D(x) 연산인 경우 0을, E 연산인 경우 1을
         n[0,i]에 저장한다.
단계 2 : n[0,i]를 이용하여 전위합을 수행한다.
단계 3 : o[0,i]가 D(sj)의 연산인 경우 n[0,i]에 저장된 값을
         R[j,0]로 전달한다.
```

보조정리 3.1 : 집합 S의 모든 s_i에 대해서 $ne(s_i)$는 $n \times n$의 재구성 메쉬에서 상수시간에 구할 수 있다.

증명. 위의 알고리즘이 집합 S의 모든 s_i에 대해서 $ne(s_i)$를 구함을 알 수 있다. 수행시간을 보면, 단계 1은 각각의 처리기에서 상수시간에 수행 가능하며, 단계 2는 보조정리 2.1에 따라서 상수시간에 수행 가능하다. 마지막으로, 단계 3은 각각의 R[0,i]에 저장된 값을 R[j,i]로 이동한 후 다시 R[j,0]로 이동하여 충돌 없이 상수시간에 수행 가능하므로 정리는 성립한다. ■

3.2 E 연산에 대한 응답 계산

다음의 보조정리 3.2에 따라서 각각의 s_i에 대해서 s_i가 $D(s_i)$ 연산에 의해서 삭제되는지, 또는 E 연산에 의해서 삭제되는지를 판단 함으로서 E 연산에 대한 응답을 결정할 수 있다.

보조정리 3.2 : s_0부터 s_i까지의 원소 중 E 연산에 의해서 삭제되는 원소의 수를 G(i)라 하면 다음이 성립한다. 이때, G(-1)은 G(i)의 정의에 따라서 0이 된다.

만약 $ne(s_i) > G(i-1)$라면 s_i는 E 연산에 의해서 삭제되고 $G(i) = G(i-1)+1$이 된다.

만약 $ne(s_i)$ $G(i-1)$라면 s_i는 D(x) 연산에 의해서 삭제되고 $G(i) = G(i-1)$이 된다.

증명. 수학적 귀납법을 이용하여 증명한다.

귀납법 기초 :

만약 $ne(s_0) > 0$ 라면 s_0는 E 연산에 의해서 삭제되고 $G(0) = 1$이 된다.

만약 $ne(s_0) \leq 0$ 라면 s_0는 D(x) 연산에 의해서 삭제되고 $G(0) = 0$이 된다.

s_0가 E 연산에 의해서 삭제되는지 또는 D(x) 연산에 의해서 삭제되는지의 결정은 다음과 같다. 먼저 $ne(s_0)$가 0이라면, 즉 연산의 순서열이 $[\cdots D(s_0) \cdots E_0 \cdots]$이라면, E_0는 연산의 순서열에서 나타나는 최초의 E 연산이므로, s_0는 $D(s_0)$에 의해서 삭제될 것이다. 그러나 만약 $ne(s_0)$가 1 이상이라면 연산의 순서열은 $[\cdots E_0 \cdots D(s_0) \cdots]$가 되고, s_0는 S에서 최소의 원소이므로 E 연산에 의해서 삭제될 것이다. 그러므로 성립한다.

귀납법 가정 :

s_0부터 s_i까지의 원소에 대하여 위의 정리가 성립함을 가정

귀납법 단계 :

G(i)의 의미로부터 우리는 $E_{G(i-1)+1}$이 수행될 때 집합 S에 원소 s_i가 남아 있다면 $E_{G(i-1)+1}$는 s_i를 삭제함을 의미한다. 왜냐하면 s_i보다 작은 원소들은 E_0부터 $E_{G(i-1)}$에 의해서 삭제되었거나 D(x)에 의해서 삭제되었기 때문이다. 따라서 우리는 연산의 순서열에서 $D(s_i)$와 $E_{G(i-1)+1}$중 어느 것이 먼저 나타나는가를 결정 함으로서 s_i가 E에 의해서 삭제되는지 또는 $D(s_i)$에 의해서 삭제되는지를 결정할 수 있다. 즉, $ne(s_i)$가 G(i-1)보다 크다면, 다시 말해서 $D(s_i)$보다 $E_{G(i-1)+1}$가 먼저 나타난다면 s_i는 $E_{G(i-1)+1}$에 의해서 삭제되게 되고 $G_{(i)}$는 $G_{(i-1)+1}$이 되며, $ne(s_i)$가 G(i-1)보다 작거나 같다면, 다시 말해서 $D(s_i)$ 뒤에 $E_{G(i-1)-1}$가 나타난다면 s_i는 $D(s_i)$에 의해서 삭제되게 되고, $G_{(i)}$는 $G_{(i-1)}$이 된다.

따라서 우리가 $ne(s_i)$와 G(i-1)을 알고 있다면, s_i가 D(x)에 의해서 삭제될지 또는 E에 의해서 삭제될 것인가를 결정할 수 있으며, 또한 G(i)를 결정할 수 있다. 따라서 수학적 귀납법에 의하여 위의 정리는 성립한다. ■

보조정리 3.2에 따라 재구성 메쉬의 i번째 열에 s_i가 저장되어 있는 경우 재구성 메쉬의 i번째 열에서 G(i)를 구할 수 있으며, 이를 계산하기 위한 재구성 메쉬의 구성은 다음과 같다.

Algorithm ComputeE

/* ne[0,j]에는 ne(j)가, s[0,j]에는 sj가 저장되어 있음을 가정하자, $0 \leq j \leq m-1$ */

단계 1 : R[0, j]는 ne[0, j]를 R[0…n-1, j]에 방송한다.

단계 2 : R[i, j]는 ne[0, j] = k인 경우에 다음과 같이 스위치를 설정 한다.

R[0…k-1,j] = {WN}{SE}

R[k, j] = {SE}

R[k+1…n-1,j] = {WE}

단계 3 : sj가 E0에 의하여 삭제되는 경우, R[0,j…n-1].S에 1을 방송한다.

단계 4 : 각 열에는 R[0…q-1, j]. W = 1, R[q…n-1, j].W = 0인 R[q,j]가 존재하게 되는데 이때

R[q,j].N이 1이면 sj는 E에 의해서 삭제되며,

R[q,j].N이 0이면 sj는 E에 의해서 삭제되지 않는다.

단계 5 : 이제 sj가 어떤 연산에 의해서 삭제되는지를 알 수 있으므로 E에 의해서 삭제되는 경우 몇 번째 E에 의해서 삭제되는가를 전위합을 이용하여 결정한다.

보조정리 3.3 : 알고리즘 ComputeE는 $n \times n$의 재구성 메쉬에서 상수시간에 E 연산에 대한 응답을 구한다.

증명. 알고리즘 ComputeE의 단계 2, 3, 4 에서의 판단 과정이 앞의 보조정리 3.2에서 s_i에 대해서 G(i)를 구하는 관계식과 동일하므로 보조정리 3.2에 의하여 알고리즘 ComputeE가 바른 결과를 계산함을 알 수 있다. 시간 및 처리기 복잡도를 보면, 단계 1, 2, 4는 방송과 스위치 설정 등 재구성 메쉬의 기능으로 상수시간에 가능하다. 단계 3에서 E_0에 의하여 삭제되는 s_j는 E_0 이전의 위치하는 D(x) 연산들에 의해서 삭제되지 않는 집합 S의 원소 중 가장 작은 원소이므로 상수시간에 구할 수 있으며, 단계 5의 전위합 역시 상수시간에 가능하다. 따라서 성립한다. ■

3.3 D(x)에 대한 응답과 집합 S의 계산

앞의 단계에서 모든 E 연산에 대해서 응답을 구했다. 모든 D(x) 연산에 대한 응답은 각 D(x) 연산에 대해서 유일하게 x가 존재하므로, S의 원소 중 E 연산에 의해서 삭제되지 않은 x들에 대해서 D(x) 연산이 그 D(x) 연산이 삭제에 성공함을 알리면 된다. 따라서 다음과 같이 해결 가능하다.

Algorithm ComputeDS

단계 1 : si가 E에 의해서 삭제되지 않은 경우 si를 R[0…n-1,i]의 처리기에 알린다.

단계 2 : Oj = D(x)인 경우 R[j,0…n-1]처리기에 단계 1에서 방송된 x가 존재하는가를 확인하여 존재한다면 D(x)는 성공하는 것이고 , 그렇지 않다면 실패하는 것이 된다.

/* 이제 각 D(x)는 성공/실패 여부를 알 수 있다. */
단계 3 : 새롭게 추가한 D(x)들에 대해서 이 연산이 성공인 경우 x들을 최종적으로 남는 집합 S의 원소로 한다.

보조정리 3.4 : Algorithm ComputeDT는 $n \times n$의 재구성 메쉬에서 상수시간에 D 연산의 응답과 최종적으로 얻어지는 집합 S를 구한다.

증명. 알고리즘 ComputeE에서 E 연산에 의해 삭제되는 원소들을 모두 결정하였으므로, D(x) 연산의 응답을 알기 위해서는 단순히 S에 x가 삭제되지 않고 존재하는지의 여부를 확인하면 된다. 따라서 위의 알고리즘은 바르게 동작한다. 시간 및 처리기 복잡도를 보면 모두 $n \times n$의 재구성 메쉬에서 상수시간에 가능하다. 따라서 보조정리 3.4는 성립한다. ■

3.4 제한된 오프라인 경쟁삭제 알고리즘

우리는 초기집합 S 그리고, E와 D(x)의 집합연산의 순서열이 주어진 경우 이 집합연산 문제를 다음의 알고리즘으로 해결 가능하다.

Aalgorithm CompetitiveDeletion
단계 1 : Algorithm ComputeNE를 수행한다.
단계 2 : Algorithm ComputeE를 수행한다.
단계 3 : Algorithm ComputeDT를 수행한다.

우리는 앞의 전처리 과정에서 S에 존재하는 모든 s_i에 대해서 $D(s_i)$가 존재하도록 $D(s_i)$들을 연산의 순서열에 삽입하였는데, 이렇게 삽입된 $D(s_i)$들은 위의 알고리즘에서 단지 $ne(s_i)$를 구하는데 사용되었다, 그런데 $ne(s_i)$는 이러한 $D(s_i)$를 삽입하지 않고도 간단히 구할 수 있으므로(실제로 D(x)가 존재하지 않는 x에 대해서 ne(x)는 전체 연산의 순서열에 존재하는 E 연산의 개수이다) 우리는 $D(s_i)$들을 삽입하지 않고도 문제를 해결할 수 있다.

정리 1. Algorithm CompetitiveDeletion은 $n \times n$의 재구성 메쉬에서 상수시간에 제한된 오프라인 경쟁삭제 문제를 해결한다.

증명. 보조정리 3.1, 3.3, 3.4에 의해서 각각의 알고리즘이 바른 결과를 낸다는 것을 알 수 있다. 또한 알고리즘을 수행하기 위해 사용했던 가정들에 대해서, 입력의 크기를 변경하지 않도록 하면서 $n \times n$의 재구성 메쉬에서 상수시간에 가정들을 만족하도록 입력을 변경할 수 있음을 앞의 관찰에서 보였다. 따라서 정리 1은 성립한다. ■

우리는 위와 같은 방법으로 초기집합 S 그리고, E와 D(x)의 집합연산의 순서열이 주어진 경우의 문제를 $n \times n$의 재구성 메쉬에서 상수시간에 해결 할 수 있다.

4. 결 론

본 논문에서는 초기집합과 D(x) 연산들과 E 연산들로 구성된 연산의 순서열이 주어지는 제한된 형태의 경쟁삭제 문제를 $n \times n$의 재구성 메쉬에서 상수시간에 해결하는 방법을 제시하였다. 이는 AT^2최적의 결과이다. 앞으로 이 논문에서의 결과를 확장하여 Insert(x) 연산을 추가한 일반적인 경우의 집합연산 문제에 대한 연구가 필요하다.

참 고 문 헌

[1] M. J. Atallah, M. T. Goddrich, S. R. Kosaraju, "Parallel Algoritms for Evaluating Sequences of Set-Manupulation Operations," AWOC 1988, Springer-Verlag, New York, 1988, pp.1-10.

[2] K. E. Lee, J. Chang, "A Constant Time Reconfigurable Mesh Algorithm for Solving the Off-line Priority Queue Problem," Journal of the Korea Information Science Society, Vol.24, No.7, pp.686-696 Jul., 1997.

[3] K. E. Lee, J. Chang, "An AT^2 Optimal Reconfigurable Mesh Algorithm for Solving the Off-line Binary Search Tree Problem," Journal of the Korea Information Science Society, Vol.24, No.3, pp.240-246, Mar. 1997.

[4] Jeffrey D. Ullman, 'Computational Aspects of VLSI', Computer Science Press, 1984.

[5] J. Jang, V. K. Prassana, "An optimal Sorting Algorithm on Reconfigurable Mesh," Journal of Parallel and Distributed Computing, pp.III.127-III.130, Aug. 1992.

[6] R. Miller, V. K. Prassana, D. I. Reisis, Q. F. Stout, "Meshes with Reconfigurable Buses," Proc. of MIT Conference on Advanced Research in *VLSI*, pp.163-178, April 1988.

[7] G. H. Chen, S. Olariu, J. L. Schwing, B. F. Wang, J. Zhang, "Constant-Time Tree Algorithms on Reconfigurable Meshes on Size $n \times n$," Journal of Parallel and Distributed Computing, pp.137-150, 1995.

[8] Jingfu Jenq and Sartaj Sahni, "Reconfigurable Mesh Algorithms for Fundamental Data Manipulation Operations," Computing on Distributed Memory Multiprocessors, NATO Series F, ed. F. Ozguner, Spring Verlag, 1993.

[9] J. Jang, V. K. Prassana, S. Sahni, "Constant Time Algorithms for Computational Geometry on Reconfigurable Mesh," IEEE Trans. on Parallel and Distributed Systems, Vol.8, No.1, pp.1-12, 1997.

이 광 의

e-mail : kelee@dongeui.ac.kr

1990년 서강대학교 컴퓨터학과(공학사)

1992년 서강대학교 컴퓨터학과(공학석사)

1997년 서강대학교 컴퓨터학과(공학박사)

1997년~2001년 한국전자통신연구원 선임연구원

2001년~현재 동의대학교 멀티미디어공학과 전임강사

관심분야 : 멀티미디어 시스템 및 응용, 저작도구, 계산이론