# 캐쉬 효과를 고려한 확장된 Pairing Heap 알고리즘

김경훈, 정균락
홍익대학교 전자계산학과
{khkim, chong}@cs.hongik.ac.kr

# Extended Pairing Heap Algorithms Considering Cache Effect

Kyoung-Hoon Kim Kyun-Rak Chong
Dept. of Computer Science, Hongik University

요 약

VLSI 기술의 발전에 따라 프로세서의 속도는 빠르게 증가하고 있는 반면 메모리의 속도는 이를 뒷받침하지 못하여 속도의 차이를 줄이기 위해 캐쉬(cache) 메모리를 사용하고 있다. 캐쉬가 알고리즘의 실행시간에 미치는 영향이 점점 더 커지고 있으나 이제까지 개발된 대부분의 알고리즘들은 이러한 캐쉬의 중요성을 고려하지 않고 개발되었다. 본 논문에서는 캐쉬 효과를 고려한 확장된 Pairing Heap 알고리즘에 관해 연구하였고, 실험을 통하여 기존의 Pairing Heap 알고리즘과 제안된 알고리즘의 성능을 비교하였다.

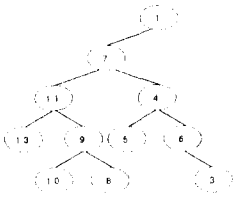
## 1. 서론

캐쉬 메모리를 얼마나 효과적으로 사용하여 메모리 접근에 드는 비용을 최소화하는가의 문제는 알고리즘의 성능에 결정적인 영향을 미치지만 이전의 알고리즘들은 캐쉬 효과에 대한 고려 없이 설계되어져 왔다. 최근에 이 문제에 대한 연구가 진행되고 있는데, Lam과 Rothberg 그리고 Wolf는 블록(block)을 이용한 행렬의 곱셈 알고리즘에 대하여 연구하였다 [1]. Lamarca와 Ladner [2]는 heap에서의 캐쉬 효과에 연구하였고 이를 통하여 d-heap을 제안하였다. 또 그들은 정렬 알고리즘 [3]과 탐색과 무작위 접근 알고리즘 [4]에서의 캐쉬 효과에 대해서도 연구하였다.

본 논문에서는 캐쉬를 효율적으로 이용하기 위하여 K-블록노드 개념을 이용한 확장된 Pairing Heap 알고리즘을 개발하였고, 실험을 통해 기존의 Pairing Heap 알고리즘보다 성능이 향상됨을 보였다.

## 2. Pairing Heap 알고리즘

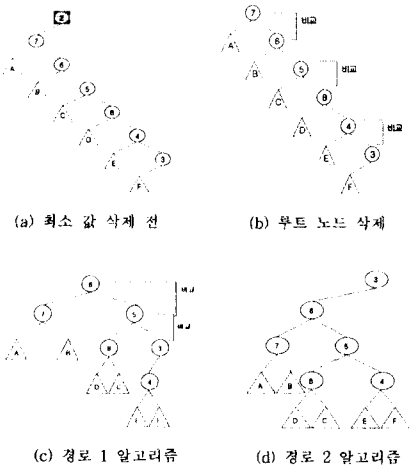
Pairing heap 알고리즘은 Freedman, Sdegewic-k 그리고 Sleator에 의해 1986년에 제안되었다 [5]. 그들은 heap 구조에 기반한 우선 순위 큐(priority queue)로 pairing heap을 사용하였다. 따라서, pairing heap은 삽입 연산과 최소 값 삭제 연산을 지원한다. 그림 1은 이진 트리로 표현된 pairing heap을 나타낸다. 루트 노드의 값은 트리에 있는 값들 중 가장 작은 값이다. 한 노드의 값은 그 노드를 루트로 하는 왼쪽 부 트리에 있는 모든 노드들의 값보다 작거나 같지만, 오른쪽 부 트리에 있는 노드들의 값보다 클 수가 있다.



[그림 1]. pairing heap의 예

Pairing heap의 삽입 연산은 매우 간단하다. Pairing heap에 새로운 값 $v$를 삽입하는 경우에는 새로운 노드를 할당받아 $v$를 그 노드의 값으로 한다. $v$와 pairing heap의 루트 노드의 값을 비교한 후, $v$가 작으면 새 노드가 새 루트 노드가 되고 이전의 루트 노드는 새 노드의 왼쪽 자식 노드가 된다. 반대로 $v$가 크면 루트 노드의 왼쪽 자식 노드가 새로운 노드의 오른쪽 자식 노드가 되고 새 노드는 루트 노드의 왼쪽 자식 노드가 된다. Pairing heap에서의 최소 값 삭제 연산은 이중 경로 알고리즘이나 다중 경로 알고리즘을 이용하여 수행되어진다[6]. 본 논문에서는 이중

경로 알고리즘을 사용한다. 최소 값 삭제를 위해서는 루트 노드를 삭제한 다음, 최소 값을 포함하고 있는 노드가 새로운 루트 노드가 될 수 있도록 트리를 재구성하여야 한다. 트리를 재구성하는 과정에서 이중 경로 알고리즘을 사용한다. 경로 1 알고리즘은 루트 노드에서 잎 노드까지 오른쪽 자식 노드를 따라서 수행된다. 두 개의 연속된 노드의 값을 비교한 후 큰 값을 포함하고 있는 노드는 작은 값을 포함하고 있는 노드의 왼쪽 자식 노드가 된다. 이러한 과정을 잎 노드에 도달할 때까지 반복한다. 경로 2 알고리즘은 경로 1 알고리즘에 의해 재구성된 트리에서 잎 노드에서부터 시작하여 루프 노드에 도달할 때까지 오른쪽 자식 노드를 따라서 이루어진다. 경로 1 알고리즘에서와 마찬가지로 큰 값을 포함하고 있는 노드가 작은 값을 포함하고 있는 노드의 왼쪽 자식 노드가 된다[7]. 그림 2는 pairing heap에서의 최소 값 삭제 연산의 예를 보여준다. (a)는 삭제 연산이 수행되기 전의 트리이고, (b)는 최소 값을 포함하는 루트 노드를 삭제한 트리이다. (c)는 (b)의 트리를 가지고 경로 1 알고리즘을 수행한 결과로 생성된 트리이고, (d)는 (c)의 트리를 가지고 경로 2 알고리즘을 수행한 최종적으로 최소 값 삭제 연산이 완료된 상태의 트리이다.



(a) 최소 값 삭제 전 (b) 루트 노드 삭제

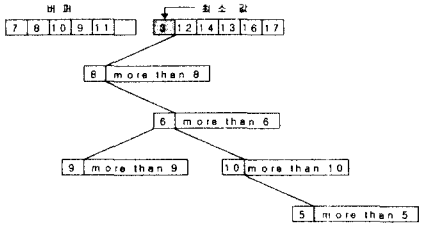
(c) 경로 1 알고리즘 (d) 경로 2 알고리즘

[그림 2]. Pairing heap 최소 값 삭제 연산의 예

### 3. K-블록노드를 이용한 확장된 Pairing Heap

Pairing heap 알고리즘은 여러 가지 문제점들을 가지고 있다. Pairing heap 최소 값 삭제 연산의 경우, 한 번에 오직 두 개의 노드의 값만을 비교하기 때문에 비교 회수가 너무 많아지고, 경로 1 알고리즘과 경로 2 알고리즘을 수행하는 동안 계속해서 트리 노드에 접근을 해야 하는데 노드들이 동적으로 할당되어 있기 때문에 일반적인 메모리 계층 구조에서의 캐쉬 메모리 블록의 비효율적인 활용을 초래하게 된다.

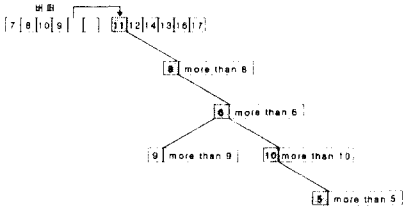
본 논문에서는 $K$-블록노드를 사용하여 이러한 문제들을 해결하고자 한다. 하나의 블록 노드는 $K$개의 데이터 배열과 두 개의 자식 블록노드를 가리키는 포인터로 이루어진다. 블록노드 안의 $K$개의 데이터는 첫 번째 배열의 데이터 값이 항상 배열 내에서의 최소 값을 유지하며 두 개의 자식 노드를 가리키는 포인터는 pairing heap에서와 같은 역할을 한다. 그림 3은 $K$=6인 $K$-블록노드를 이용한 확장된 pairing heap을 나타낸다. K-블록노드의 데이터 배열은 메모리 공간을 절약하기 위해서 항상 차 있어야 한다. 따라서, 데이터를 삽입하거나 최소 값을 삭제할 때 데이터의 이동이 발생한다. 데이터의 이동에 드는 비용이 너무 크기 때문에 이 비용을 최소화하기 위하여 추가적으로 K개의 데이터를 저장할 수 있는 버퍼를 사용한다. 데이터 삽입 시, 데이터를 버퍼에다 삽입한 다음 버퍼가 차면 새 블록노드를 할당하고 버퍼의 데이터를 블록노드의 데이터 배열로 옮긴 후 새 블록노드를 확장된 pairing heap에 삽입하는 방식으로 데이터의 이동을 최소화 할 수 있다.



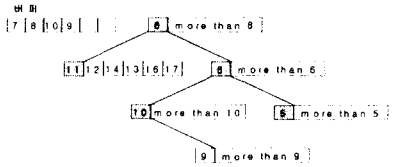

[그림 3]. 확장된 pairing heap 트리

$K$-블록노드를 이용한 확장된 pairing heap 삽입 연산은 우선 삽입하고자하는 데이터를 버퍼에 넣는다. 그리고, 만약 버퍼가 가득 차 있다면 새로운 블록노드를 할당하여 버퍼에 있는 데이터들을 버퍼로 옮기고 할당된 블록노드를 트리에 pairing heap의 삽입 연산과 동일한 방법으로 삽입한다. K-블록노드를 이용한 확장된 pairing heap 최소 값 삭제 연산은 기존의 pairing heap 최소 값 삭제 연산보다 다소 복잡하다. 확장된 pairing heap 트리가 비어있다면 버퍼의 첫 번째 데이터를 삭제하고 나머지 데이터들 중의 최소 값을 첫 번째 데이터로 옮겨주면 된다. 버퍼가 비어있고 확장된 pairing heap 트리가 비어있지 않다면 확장된 pairing heap 트리의 루트 블록노드에서 첫 번째 데이터를 삭제한 후, 나머지 데이터들을 버퍼로 모두 옮긴 후 루트 블록노드를 삭제한다. 그리고 난 뒤 블록노드의 첫 번째 데이터만을 가지고 pairing heap에서의 경로 1 알고리즘과 경로 2 알고리즘을 수행하면 된다. 만약 확장된 pairing heap 트리와 버퍼 모두 비어있지 않다면 버퍼의 첫 번째 데이터와 확장된 pairing heap 트리의 루트 블록노드의 첫 번째 데이터 값을 비교한 후, 버퍼의 첫 번째 데이터 값이 작다면 버퍼의 첫 번째 데이터를 삭제하면 된다. 반대로, 루트 블록노드의 첫 번째 데이터 값이 작다면 루트 블록노드의 첫 번째 데이터 값을 삭제하고 버퍼의 맨 끝에 있는 데이터를 가져와서 이 데이터와 루트 블록노드의 나머지 데이터를 중에 최소 값을 배열의 첫 번째 원소에 유지한 다음, 첫 번째 데이터가 변했기 때문에 왼쪽 자식 블록노드의 첫 번째 데이터보다 작다는 것을 보장할 수 없으

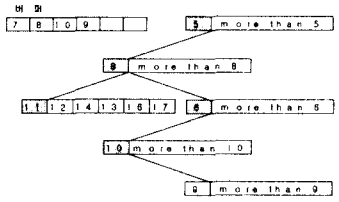
므로 루트 블록노드의 왼쪽 자식 노드를 오른쪽 자식노드로 옮긴다. 그리고 나서 루트 블록노드에서부터 경로 1 알고리즘과 경로 2 알고리즘을 수행하면 최소 값 삭제 연산이 완료된다. 그림 4는 K-블록노드를 이용한 확장된 pairing heap에서의 최소 값 삭제 연산의 예를 보여준다. 그림 3의 확장된 pairing heap 트리에서 최소 값인 '3'을 삭제하고 버퍼의 마지막 데이터 값 '11'을 루트 블록노드로 옮겨 배열의 첫 번째 원소에 최소 값을 유지한 후, 포인터를 이동한다(a). 그리고 난 뒤 경로 1 알고리즘을 수행한다(b). 마지막으로, 경로 2 알고리즘을 수행하면 최종적으로 '5'를 포함하는 블록노드를 새로운 루트 블록노드로 하는 확장된 pairing heap 트리가 얻어진다(c).




(a) 최소 값을 삭제하고 포인터를 이동




(b) 경로 1 알고리즘 수행 결과




(c) 경로 2 알고리즘 수행 결과

[그림 4] 확장된 pairing heap 최소 값 삭제 연산의 예

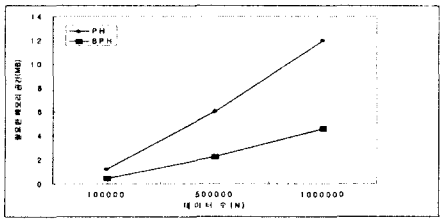
## 3. 실험 결과 및 분석

제안된 알고리즘들은 C로 구현되어 SUN Enterprise 3000에서 실행되었다. 표 1은 K=1, 6, 14, 30, 62 즉, K-블록노드의 크기가 32, 64, 128, 256바이트일 때 정수 데이터를 100만개 저장한 상태에서 삽입 연산과 최소 값 삭제 연산을 각각 확률 0.5로 100만 번 실행했을 때의 결과이다. 표에서 실행 시간의 단위는 ms이다. K=1인 경우는 기존의 pairing heap 알고리즘의 경우이다. 표에서 K=14 즉 블록노드의 크기가 64바이트의 경우에 속도가 가장 빨라짐을 알 수 있다.

[표 1] 실행 시간 비교

| K | 노드의 크기 | 실행 시간 |
|---|---|---|
| 1 | 12bytes | 2718000 |
| 6 | 32bytes | 1790000 |
| 14 | 64bytes | 1762000 |
| 30 | 128bytes | 2012000 |
| 62 | 256bytes | 2206000 |

그림 5는 10만개, 50만개, 100만개의 데이터를 유지하는 경우에 필요한 메모리 공간의 크기를 나타낸다. PH는 pairing heap을 의미하고 BPH는 K=14인 K-블록노드를 사용한 확장된 pairing heap을 의미한다.




[그림 5]. 필요한 메모리 공간의 크기

## 4. 결론

메모리 접근에 드는 비용이 커짐에 따라 캐쉬를 얼마나 효율적으로 사용하느냐의 문제는 프로그램의 실행시간에 결정적인 영향을 준다. 본 논문에서는 K-블록노드를 이용한 확장된 pairing heap 알고리즘을 개발하였고, 실험을 통하여 기존의 pairing heap 알고리즘보다 실행 시간과 메모리 공간 사용 면에서 성능이 향상됨을 보였다.

## 5. 참고문헌

[1] M. S. Lam and E. E. Rothberg. The Cache Performance and Optimizations of Blocked Algorithms. Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV), pages 63-74, 1991

[2] A. LaMarca and R. E. Ladner. The Influence of Caches on the Performance of Heaps. Journal of Experimental Algorithmics, Vol 1, Article 4, 1996

[3] A. Lamarca and R. E. Ladner. The Influence of Caches on the Performance of Sorting . In Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 370-379, 1997

[4] R. E. Ladner , J. D. Fix and A. LaMarca. Cache Performance Analysis of Traversals and Random Accesses. In Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 613-622, 1999

[5] Fredman, M.L., and Sdegewick, R., Sleator, D.D., and Tarjan, R.E. The pairing heap: A new form of self-adjusting heap. Algorithmica 1 (Mar, 1986.) 111-129.

[6] John T. Stasko and Jeffrey Scott Vitter. Pairing heaps: Experiments and Analysis. ACM (Mar 1987.) Volume 30 Number 3.