DOI: 10.3745/KIPSTA.2010.17A.4.167

우선순위 큐 성능 시험에 관한 연구

정 해 재†

요 약

본 논문에서는 우선순위 큐에 대한 성능 시험 모델을 제안하고, 제안된 모델에 따라 대표적인 우선순위 큐인 전통 힙, 후순위 힙, 및 페어링 힙의 성능 시험 결과를 보여준다. 이들 중 전통 힙이 분석된 시간복잡도에 있어서 최악인 것으로 알려져 있다.

그러나 제안된 성능 시험 모델에 근거한 성능 시험 결과에 따르면, 포인터를 사용하는 페어링 힙이 가장 느리고 전통 힙이 가장 빠른 것으로 나타났다. 두 묵시 힙에 대해서도, 분석된 시간복잡도로는 후순위 힙이 전통 힙보다 우수하지만, 성능 시험 결과는 반대인 것으로 나타났다.

키워드 : 자료 구조, 우선순위 큐, 힙

A Study on the Runtime Test of Priority Queues

Haejae Jung†

ABSTRACT

This paper proposes a set of runtime test models for priority queues and shows the runtime test results based on the proposed test models for the representative priority queues: the traditional heap, post-order heap, and pairing heap. Among these heaps, the traditional heap is the worst in time complexity analyzed.

But, according to our experimental results based on the test models proposed, it is shown that the slowest one is the pairing heap that utilizes pointers and the fastest one is the traditional heap. For the two implicit heaps, these results are in contrary to the fact that the post-order heap is better than the traditional heap in time complexity analyzed.

Keywords : Data Structure, Priority Queue, Heap

1. 서 론

우선순위 큐는 스케줄링, 사건 시뮬레이션, 우선순위에 따른 검색, 정렬 등과 같은 다양한 응용에 사용될 수 있기 때문에 많은 자료구조가 연구되어 발표되었으며, 소프트웨어 구현을 위한 기본 자료 구조로서 분류되고 있다.

최소 우선순위 큐(min priority queue)는 임의의 데이터 집합 S에서 특정 우선순위에 따른 최소값을 빨리 찾고자 고안된 자료구조로서, 다음과 같이 정의되는 삽입과 삭제 두 가지 연산을 기본으로 제공한다.

- insert(e, S): 집합 S에 데이터 e를 추가.
- removeMin(S): 집합 S로부터 우선순위가 가장 낮은 데이터를 삭제.

우선순위 큐 성능 비교를 위해 [1]에서는 성능 평가 모델을 제안하고 이에 따라 다양한 우선순위 큐를 구현하여 성능 비교를 하였다. 제안된 성능 평가 모델 중 홀드(hold) 모델은 그 이후에도 자료 구조 성능 평가 비교를 위해 이용되었다 [2, 3].

우선순위 큐 자료구조 중 포인터를 이용하는 자료구조로서는 피보나치 힙, 페어링 힙(pairing heap), M-힙 등이 있으며, 이들 중 페어링 힙이 우수한 성능을 나타내는 것으로 알려져 있다 [4-7]. 페어링 힙에서 삽입하는데 걸리는 시간은 최악의 경우 O(1)이고, 삭제하는데 걸리는 최악의 시간복잡도와 전이 시간복잡도는 각각 O(n)과 $O(\log n)$인 것으로 알려져 있다.

포인터를 사용하지 않는 묵시 우선순위 큐로는 전통 힙(traditional heap), 후순위 힙(post-order heap)이 대표적이다[8-10]. 전통 힙은 삽입과 삭제 연산에 각각 O(log*n*) 시간이 걸리고, 후순위 힙은 상수 삽입 전이 시간복잡도(amortized time complexity)와 O(log*n*) 삭제 시간복잡도를 가진다.

본 논문에서는 다양한 우선순위 큐 자료구조에 대한 성능

※ 이 논문은 2007학년도 안동대학교 국제학술교류보조금에 의하여 연구되었음
† 종신회원 : 안동대학교 공과대학 정보통신공학과 교수
논문접수 : 2010년 3월 18일
심사완료 : 2010년 5월 20일

을 비교하기 위해 성능 시험 모델을 제안하고, 제안된 모델에 근거하여 우선순위 큐 성능을 실험적으로 비교한다. 제안된 성능 시험 모델은 기존의 [1-3]에서 사용하였던 것에 무작위 모델(random model)을 추가하여 성능 시험 방법을 개선하였다. 성능 비교는 비교적 최근에 발표되고 시간복잡도 분석을 통하여 우수한 것으로 알려진 후순위 힙을 다른 힙과 실험을 통하여 비교함으로써 이루어진다. 이를 위해 후순위 힙과 동일한 종류 즉, 묵시 힙인 전통 힙을 성능 비교를 위해 우선 선정하였다. 이에 더하여, 묵시 힙이 아닌 포인터를 사용하는 우선순위 큐 중 우수한 성능을 보이는 것으로 알려진 페어링 힙과의 성능 비교도 하여, 묵시 힙과 묵시 힙이 아닌 우선순위 큐에 대한 성능 비교 결과도 보인다.

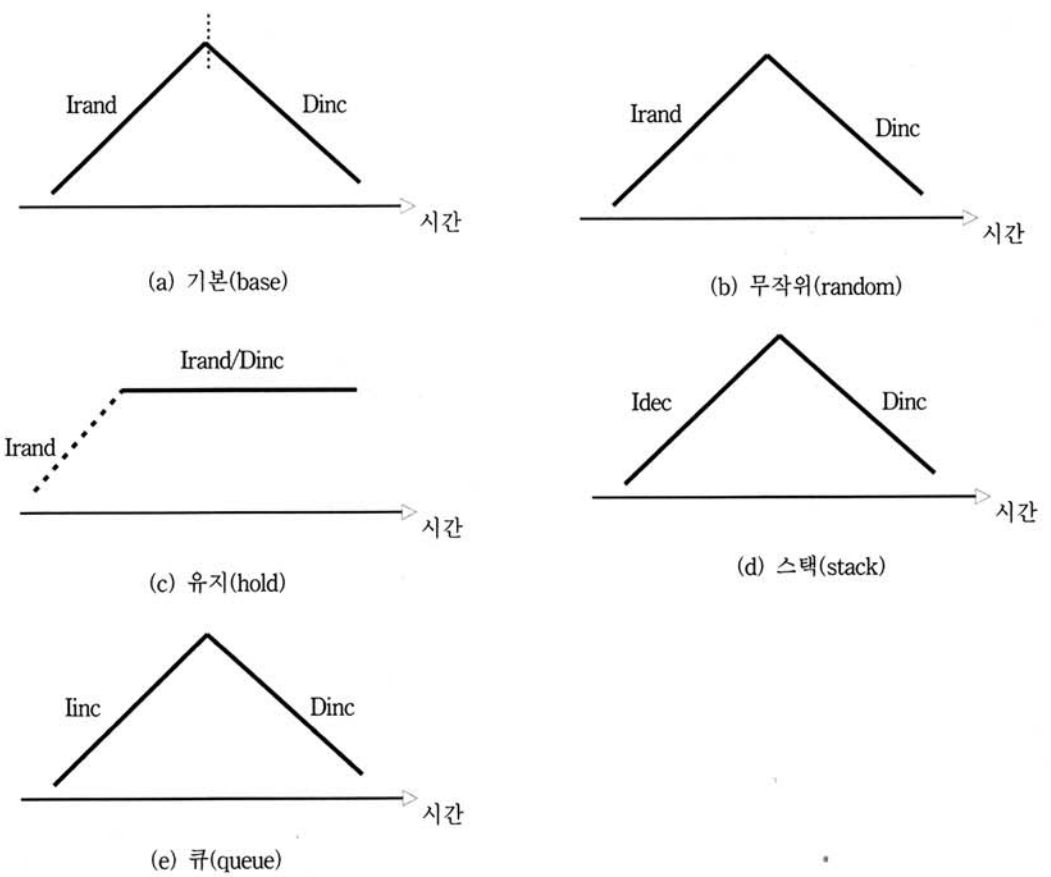
2. 성능 시험 모델

우선순위 큐 성능 시험을 위해서는 다양한 우선순위 큐 응용 분야를 포괄할 수 있는 실험 방법이 필요하다. 이를 위해 본 논문에서는 (그림 1)에 보인 우선순위 큐의 응용 분야에 근거한 5가지 성능 시험 모델을 제안하고, 제안된 모델을 이용하여 세 우선순위 큐 자료구조의 실행시간을 측정하여 성능을 서로 비교한다. 제안된 성능 시험 모델은 기본(base), 무작위(random), 유지(hold), 스택(stack), 및 큐(queue) 모델이다.

(그림 1)의 각 모델의 굵은 선 위의 첫 문자는 연산의 종류를 나타내는데, I는 삽입(Insert)을 D는 삭제(Delete) 연산을 나타낸다. 각 연산 종류와 연결된 소문자로 구성된 문자열은 연산에 적용되는 데이터의 성질을 나타내는데, rand는 무작위(random), inc는 오름차순(increasing order), dec는 내림차순(decreasing order)을 의미한다. 예를 들어, Dinc는 데이터를 오름차순으로 삭제를 한다는 의미이다.

(그림 1)의 기본 모델을 통하여 각 자료 구조에서 제공하는 연산 각각의 성능을 비교할 수 있고, 유지 모델을 이용하여 스케줄링이나 시뮬레이션과 같은 응용에서 자료 구조에 일정한 수의 데이터를 유지하면서 삽입과 삭제 연산이 반복 적용될 때의 성능을 측정 비교할 수 있다. 무작위 모델은 임의의 순서로 된 데이터를 정렬하는데 걸리는 시간을 측정 비교하기 위함이고, 스택과 큐 모델은 무작위 모델과는 달리 극단적인 패턴을 가진 데이터를 정렬하는데 걸리는 시간을 측정 비교하기 위함이다. 스택과 큐 모델을 이용한 시험은 특정한 순서의 데이터를 빨리 처리할 수 있도록 설계된 자료구조인지도 확인할 수 있다.

제안된 성능 시험 모델을 이용하여 실행 시간을 측정하는 방법에 대해 설명하면 아래와 같다.



(그림 1) 성능 시험 모델

- 기본(base) 모델 : n 개의 무작위 데이터를 빈(empty) 우선순위 큐에 삽입한 후, 삽입된 n 개의 데이터 모두를 삭제한다. n 개의 데이터를 삽입하고 삭제하는데 걸리는 시간을 각각 측정한다.
- 무작위(random) 모델 : 무작위 모델에서는 n 개의 무작위 데이터를 빈 우선순위 큐에 삽입한 후, 삽입된 n 개의 데이터를 모두 삭제한다. 이렇게 $2n$ 개의 삽입과 삭제 연산에 소요된 총 시간을 측정하여 각 힙의 성능을 비교한다.
- 유지(hold) 모델 : c 개의 무작위 데이터로 우선순위 큐 자료 구조를 초기화 한 후, 삽입과 삭제가 혼합된 n 회의 연산을 실행하는데 걸리는 총 시간을 측정한다. 이때, 삽입과 삭제가 혼합된 n 회의 연산이 실행되는 동안 자료 구조에 대략 c 개의 데이터가 유지되도록 하여야 한다.
- 스택(stack) 모델 : 스택 모델에서는 내림차순으로 n 개의 데이터를 삽입한 후, 삽입된 모든 n 데이터를 오름차순으로 삭제한다. 삽입과 삭제에 걸린 총 시간을 측정한다.
- 큐(queue) 모델 : 큐 모델에서는 n 개의 데이터를 오름차순으로 삽입한 후, 오름차순으로 n 데이터를 삭제한다. 삽입과 삭제에 소요된 총 시간을 측정한다.

3. 실행 시간 측정 및 결과 비교

본 논문에서는 전통 힙, 페어링 힙, 및 후순위 힙을 최소 힙으로 구현하여, 제안된 시험 모델에 근거하여 성능 비교를 하였다. 이들 중 페어링 힙의 최소값 삭제 함수 removeMin( ) 은 효율적인 2 단계 합병 방식으로 구현하였다. 즉, 왼쪽에서 오른쪽으로 가는 1단계에서는 이웃하는 두 컴포넌트 힙(component heap)을 합병(merge)하고, 오른쪽에서 왼쪽으로 가는 2단계에서는 1단계 합병 결과로 나타나는 컴포넌트 힙 모두를 하나로 합병한다. 또한, 포인터를 이용하는 페어링 힙 구현에서는 묵시 힙인 전통 힙과 후순위 힙과의 공정한 성능 비교를 위해 필요한 모든 메모리를 사전에 할당받아 연결 리스트로 유지하여 프로그램 실행 중 메모리 할당에 소요되는 시간을 최소화 하였다.

프로그램은 C++ 언어로 작성하였으며, 컴파일은 GNU C++ 컴파일러인 g++로 이루어졌다. 이때, 컴파일 옵션은 최대 최적화 옵션인 -O3를 이용하였다. 프로그램 실행 시간 측정은 8GB(기가바이트)의 주기억장치를 가지고 있는 SUN Sparc Station에서 이루어졌다.

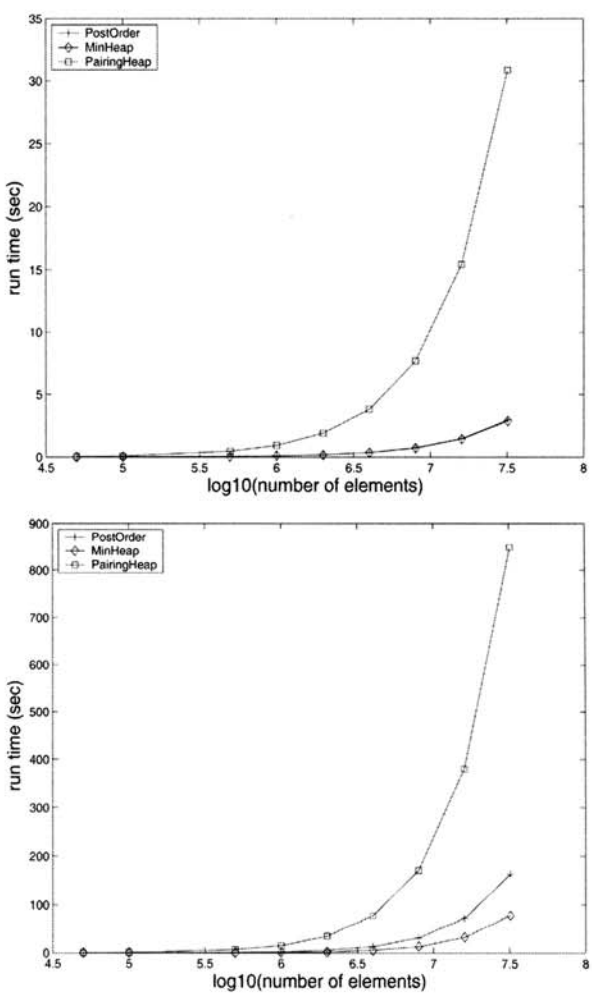
성능 평가 모델 각각에 대해 9개의 정수형 데이터 집합을 사용하여 실행 시간을 측정하였으며, 측정된 시간으로부터 평균 실행시간과 상대성능(speedup)을 계산하였다. 측정에 사용된 9개의 데이터 집합은 각각 5만, 10만, 50만, 백만, 2백만, 4백만, 8백만, 1600만, 3200만개 이다. 평균 실행시간은 각 데이터 집합에 대해 9회 실행하여 측정된 실행 시간의 평균을 계산한 값이다. 이 때, 무작위 데이터를 사용하는 성능 시험 모델 즉 기본, 무작위, 및 유지 모델의 경우, 실행 때 마다 서로 다른 데이터를 사용하였다. 스택과 큐 모델은 동일한 데이터를 이용하였다. 상대성능은 후순위 힙의 평균 실행시간을 다른 힙의 평균 실행시간으로 나눈 값으로 정의하여, 후순위 힙을 기준으로 다른 힙의 성능을 비교할 수 있게 하였다.

본 논문에서는 위와 같이 계산된 평균 실행시간과 상대성능을 3.1~3.5절에서 보는 바와 같이 그래프로 표현하였다. 그래프의 가로축은 성능 시험에 사용된 데이터 개수 n 에 로그 함수를 적용한 $\log_{10}(n)$ 값으로 나타내고, 세로축은 측정된 실행 시간을 통해 계산된 평균 실행시간/상대성능을 나타낸다.

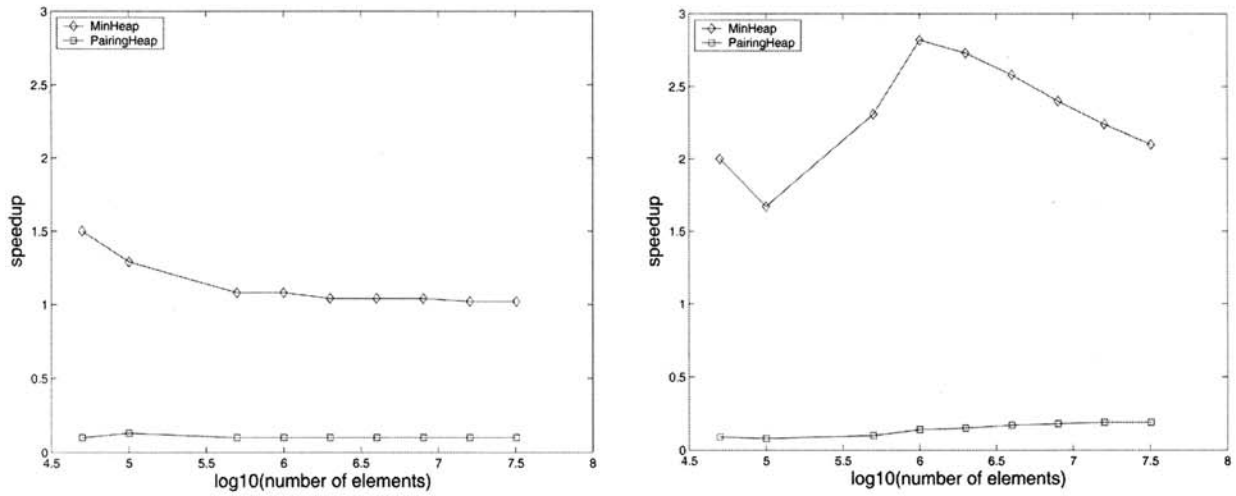
3.1 기본 모델

(그림 2)와 (그림 3)은 기본 모델에서의 평균 실행시간 및 상대성능을 각각 보여주고 있다.

기본 모델에서의 삽입 연산의 성능은 전통 힙(MinHeap으로 표기)이 가장 우수하고, 페어링 힙이 가장 느린 것으로 나타났다. 그러나 데이터의 개수가 증가함에 따라 전통 힙



(그림 2) 기본 모델에서의 실행 시간 : 삽입(위) 및 삭제(아래)



(그림 3) 기본 모델에서의 상대성능 : 삽입(왼쪽) 및 삭제(오른쪽)

과 후순위 힙이 유사한 성능을 보여주고 있다.

삭제의 경우에는 전통 힙이 가장 빠른 것으로 나타났으며, 페어링 힙이 가장 느린 것으로 나타났다. 전통 힙은 데이터가 백만 개 일 때 (그래프의 가로축 눈금이 6으로 표시), 후순위 힙보다 약 2.8배 빠름을 보여주고 있다.

3.2 무작위 모델

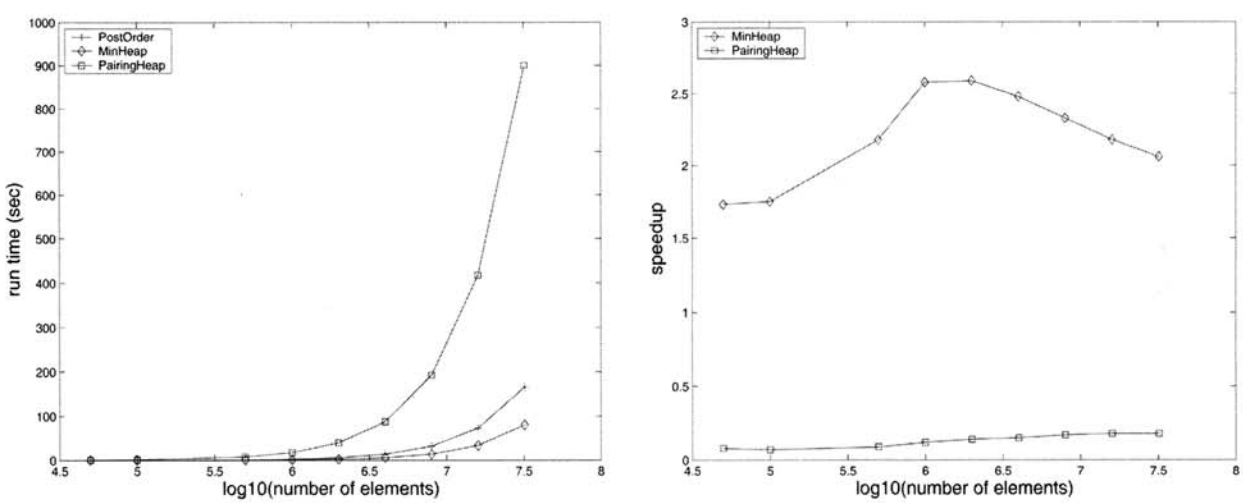
(그림 4)는 무작위 모델에서의 성능 실험 결과를 보여주고 있다. 무작위 모델에서도 전통 힙이 가장 빠른 것으로 나타났으며, 페어링 힙이 가장 느린 것으로 나타났다. 전통 힙은 후순위 힙보다 최대 약 2.6배까지 우수함을 보여주고 있다.

3.3 유지 모델

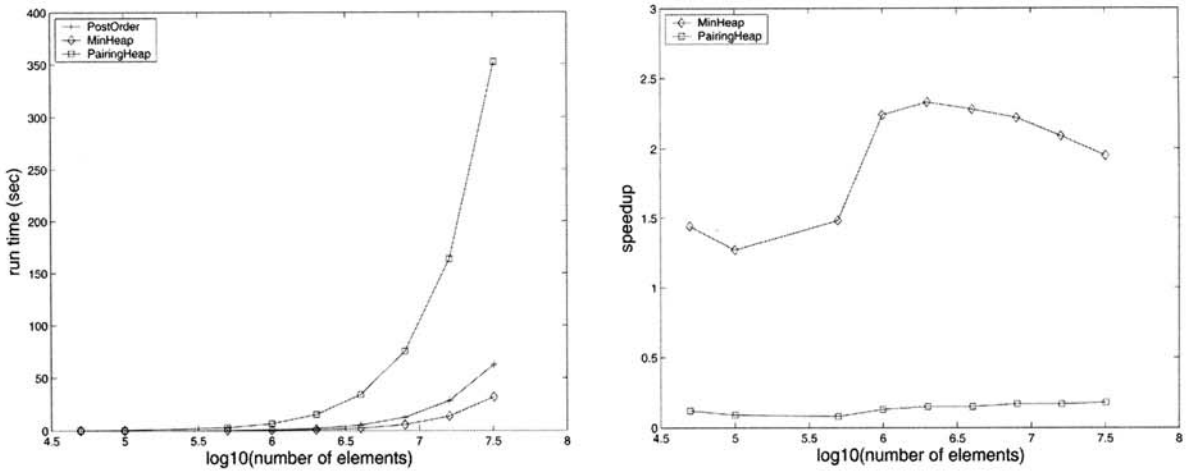
(그림 5)는 유지 모델에서도 전통 힙이 가장 빠르고 페어링 힙이 가장 느린 것을 보여주고 있다. 데이터 개수가 2백만 개 일 때, 전통 힙이 후순위 힙보다 약 2.3배 빠른 것으로 나타났다.

3.4 스택 모델

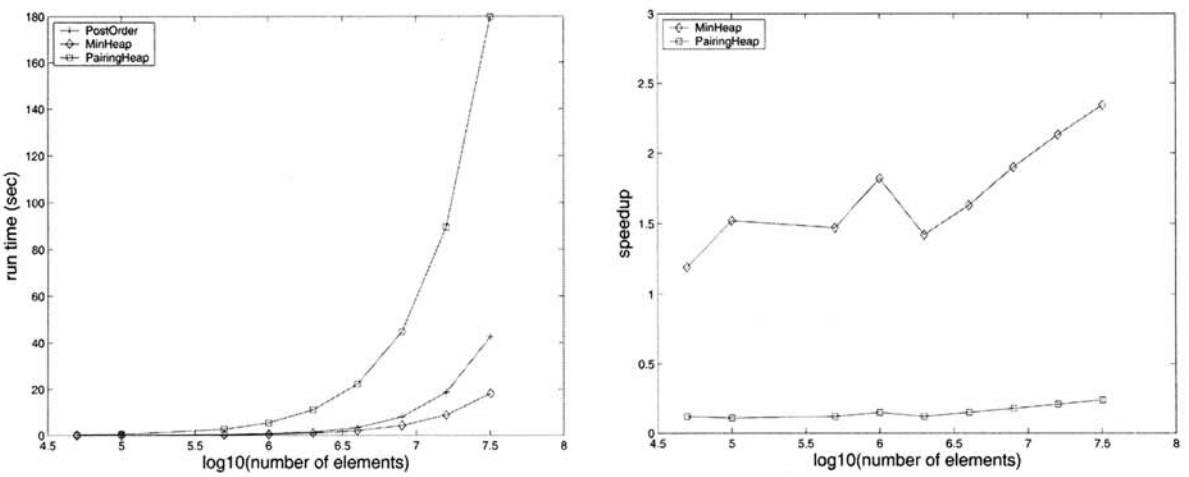
(그림 6)은 스택 모델에서의 성능 측정 결과를 보여주고 있다. 이 모델에서도 전통 힙이 가장 빠르고 페어링 힙이 가장 느린 것을 알 수 있다. 3200만 데이터에 대해 전통 힙이 후순위 힙보다 약 2.4배 빠른 것을 알 수 있다. 이 모델에서는 데이터 수가 증가함에 따라 전통 힙이 후순위 힙보다 지속적으로 빨라지는 것을 관찰 할 수 있다.



(그림 4) 무작위 모델에서의 성능 : 실행 시간(왼쪽) 및 상대성능(오른쪽)



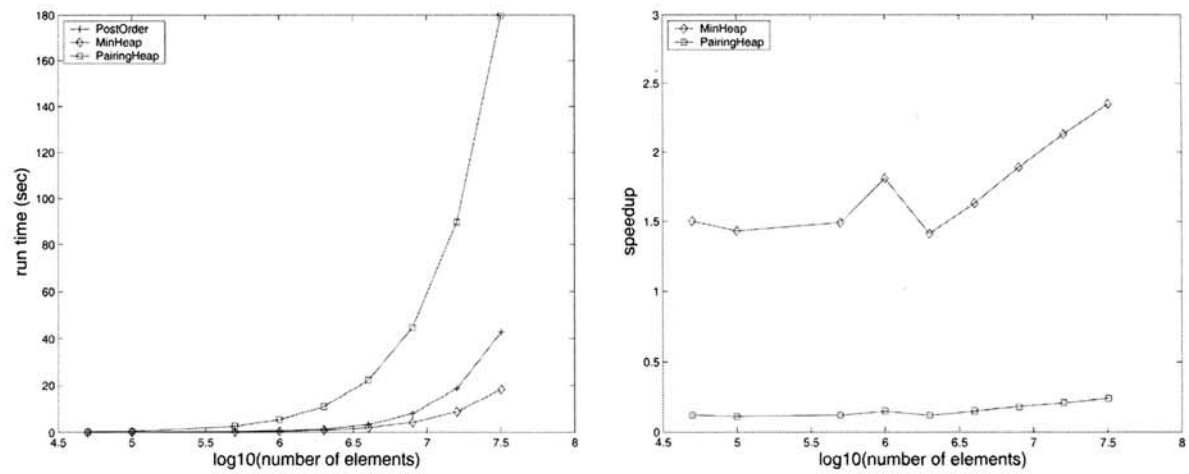
(그림 5) 유지 모델에서의 성능 : 실행 시간(왼쪽) 및 상대성능(오른쪽)



(그림 6) 스택 모델에서의 성능 비교 : 실행시간(왼쪽) 및 상대성능(오른쪽)

3.5 큐 모델

다른 시험 모델의 결과와 마찬가지로 큐 모델에서도 전통 힙이 가장 빠르고 페어링 힙이 가장 느린 것을 (그림 7)로부터 알 수 있다. 3200만 데이터에 대해 전통 힙은 후순위 힙보다 약 2.3배 빠름을 알 수 있다. 스택 모델에서와 같이 데이터 개수가 증가함에 따라 전통 힙이 후순위 힙보다 지속적으로 빨라짐을 보이고 있다.



(그림 7) 큐 모델에서의 성능 비교 : 실행 시간(왼쪽) 및 상대성능(오른쪽)

4. 실험 분석 및 결론

본 논문에서는 우선순위 큐의 다양한 응용 분야를 고려하여 제안한 5개의 성능 시험 모델을 이용하여 후순위 힙, 전통 힙, 및 페어링 힙의 성능을 측정하여 비교하였다.

실험 결과, 분석 상으로는 가장 느린 O(log*n*)의 삽입 및 삭제 시간복잡도를 가지는 전통 힙이 실제적으로는 가장 빠른 것으로 나타났고, 삽입과 삭제에 각각 O(1) 시간복잡도와 O(log*n*) 전이 시간복잡도를 가진 것으로 알려진 페어링 힙이 가장 느린 것으로 나타났다. 또한, 후순위 힙의 경우 삽입 연산의 전이 시간 복잡도가 $O(1)$이고 삭제 연산 시간 복잡도가 O(log*n*)이지만, 제안된 모든 성능 시험 모델에 대해 전통 힙보다 느린 것으로 나타났다. 단지 기본 모델의 삽입 연산에 대해서만 후순위 힙이 전통 힙과 유사한 성능을 보였을 뿐이다.

위와 같은 실험 결과는 다음과 같은 요소에 기인되는 것으로 분석된다.

- 전통 힙은 캐시 메모리를 다른 우선순위 큐보다 잘 이용하고 있다. 전통 힙은 배열의 인덱스 순서대로 데이터 삽입과 삭제가 이루어져, 삽입과 삭제 동안 루트 노드에서 임의의 리프 노드까지의 패스에 있는 노드들이 반복적으로 접근된다.
- 후순위 힙의 삭제 연산은 최악의 경우 $3\log n$ 시간이 걸리는 반면, 전통 힙은 log*n* 시간이 걸린다.
- 전통 힙은 완전이진트리 (complete binary tree)로 구성되며, 리프 노드의 개수는 전체 노드 개수의 약 반을 차지한다. 따라서 데이터 삽입은 대부분 리프노드 근처에서 끝날 확률이 크다.

참 고 문 헌

[1] D. Jones, "An empirical comparison of priority-queue and event-set implementations," Communications of the ACM, 29(4), pp.300-311, Apr., 1986.

[2] H. Jung, S. Sahni, "Supernode binary search trees," International Journal of Foundations of Computer Science, 14(3), 465-490, 2003.

[3] H. Jung, "The d-deap*: A fast and simple cache-aligned d-ary deap", Information Processing Letters, 93(2), pp.63-67, Jan., 2005.

[4] M. Fredman, R. Sedgewick, R. Sleator, and R. Tarjan, "The pairing heap: A new form of self-adjusting heap," Algorithmica, 1, pp.111-129, 1, Mar., 1986.

[5] T.J. Stasko and J.S. Vitter, "Pairing heaps: experiments and analysis", Communications of the ACM, 30(3), pp.234-249, 1987.

[6] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," JACM, 34(3), pp.596-615, 1987.

[7] S. Bansal, S. Sreekanth, and P. Gupta, "M-heap: A Modified heap data structures," International Journal of Foundations of Computer Science, 14(3), pp.491-502, 2003.

[8] J. Williams, "Algorithm 232 Heapsort," Communications of the ACM, 7(1), pp.347-348, 1964.

[9] N. Harvey and K. Zatloukal, "The Post-order heap," In Proceedings of the Third International Conference on Fun with Algorithms(FUN), May, 2004.

[10] E. Horowitz, S. Sahni, and D. Mehta, Fundamentals of Data Structures in C++, W. H. Freeman, San Francisco, 1995.

정 해 재

e-mail : hjjung@andong.ac.kr

1984년 경북대학교 전자계산학과(학사)

1987년 서울대학교 컴퓨터공학과(공학석사)

2000년 플로리다대학교(UF) 컴퓨터정보학과 (공학박사)

1988년~1995년 한국전자통신연구원 선임연구원

2001년~2002년 Numerical Technologies Inc. Staff Engineer

2003년~2005년 성신여자대학교 컴퓨터정보학부 교수

2005년~현 재 안동대학교 공과대학 정보통신공학과 교수

관심분야 : 알고리즘, 계산기하, 웹 데이터베이스, 웹 보안