8-힙* : 빠른 8-원 묵시 우선순위 큐

정 해 재†

요 약

스케줄링이나 정렬과 같은 응용에 이용될 수 있는 우선순위 큐는 포인터를 사용하는 것과 포인터를 이용하지 않고 묵시적으로 표현하는 두 가지가 있다. 묵시 우선순위 큐는 메모리 이용에 있어서 포인터를 사용하는 것보다 효율적이다. 묵시 우선순위 큐에는 이진 트리에 근거한 전통적인 2-힙이 있는데, 이는 캐쉬 메모리를 효율적으로 이용하는 8-원 트리에 근거한 8-힙보다 느린 것으로 나타났다. 본 논문에서는 구현하기 쉽고 빠른 새로운 묵시 우선순위 큐인 8-힙*를 제안한다. 실험을 통하여 8-힙*가 2-힙 뿐만 아니라 8-힙보다 빠름을 보인다.

8-heap* : A fast 8-ary implicit priority queue

Haejae Jung†

ABSTRACT

Proirity queues(PQ) can be used in applications such as scheduling or sorting. The data structures for PQ can be constructed with or without pointers. The implicit representation without pointers uses less memory space than pointer-based representation. It is shown that a 2-heap, a traditional implicit PQ based on a binary tree, is slower than an 8-heap based on a 8-ary tree. This is because 8-heap utilizes cache memory more efficiently. This paper presents a novel fast implicit heap called 8-heap* which is easier to implement. Experimental results show that the 8-heap* is faster than 8-heap as well as 2-heap.

키워드 : 자료 구조(Data Structures), 우선순위 큐(Priority Queue), 힙(Heap), 묵시(Implicit)

1. 서 론

우선 순위 큐(priority queue : PQ)는 운영 체제 스케줄링, 사건 시뮬레이션, 또는 정렬과 같은 응용에 사용될 수 있는데, 어떤 데이터 집합에서 우선순위가 가장 높거나 낮은 것을 빨리 찾는 자료 구조이다. 전자를 최대 우선순위 큐라 하고 후자를 최소 우선순위 큐라 하는데, 최대 우선순위 큐는 어떤 집합 S에 대해 다음과 같은 연산을 기본적으로 지원한다.

- Insert(e, S) : 집합 S에 새로운 임의의 데이터 e를 추가.
- DelMax(S) : 집합 S로부터 우선순위가 가장 높은 데이터를 삭제.

우선순위 큐를 구현하기 위한 자료구조 표현은 포인터를 이용한 방법과 포인터를 전혀 이용하지 않는 묵시 표현 방법이 있다. 포인터를 이용한 우선순위 큐에는 피보나치 힙이나 페어링 힙 등이 있는데, 이들은 시간 복잡도에 있어서 묵시 표현 방법보다 우수하다[1-3]. 그러나 포인터를 위한 공간이 추가로 필요하기 때문에 묵시 표현 방법보다 메모리 사용에 있어 효율적이지 못하다.

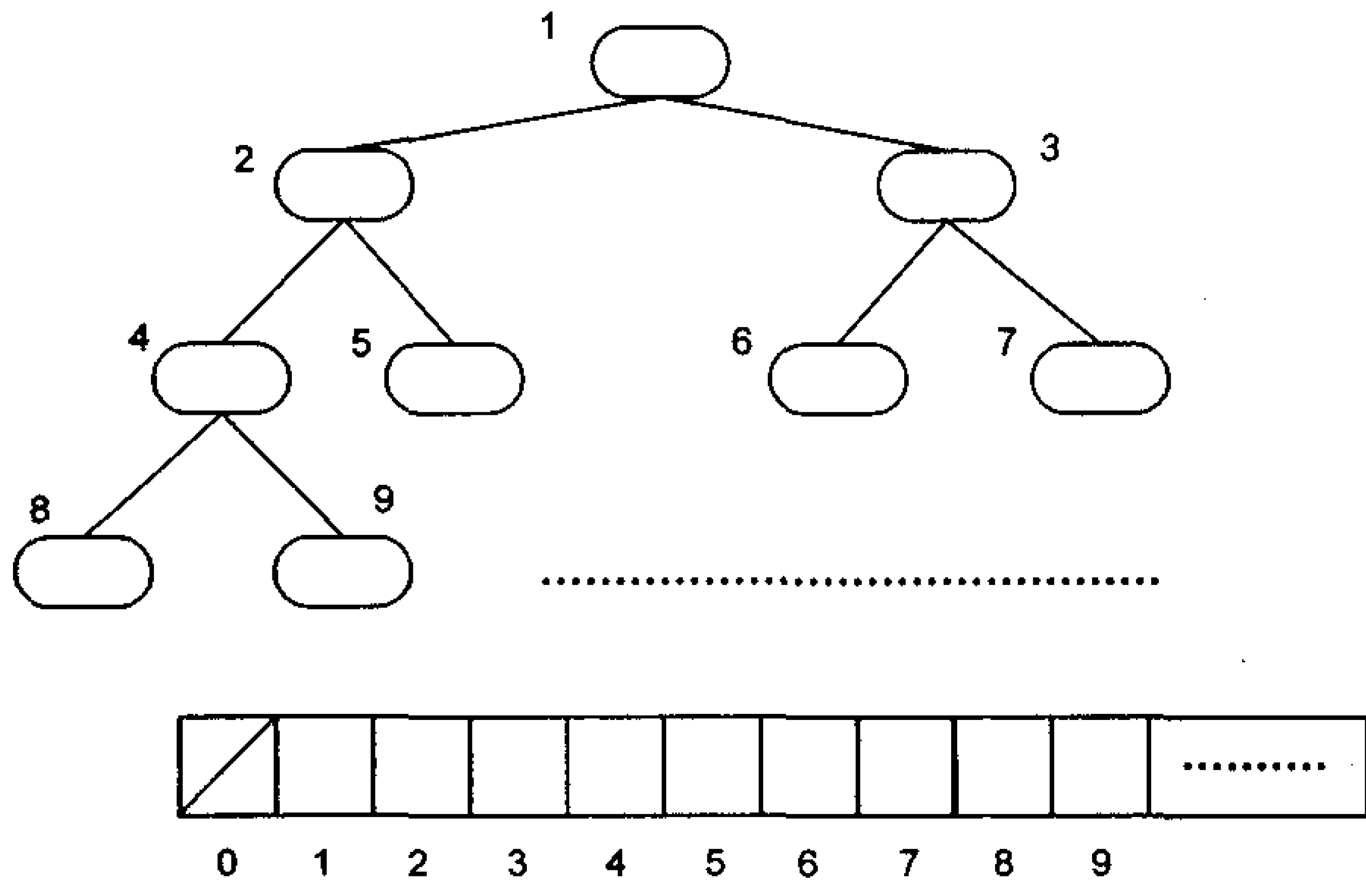
묵시 우선순위 큐는 완전 이진 트리를 메모리 배열에 사상시켜, 각 노드를 배열 인덱스를 통하여 접근하는데, 부모-자식 노드 관계는 단순한 배열 인덱스의 수식으로 표현된다[4]. 그러나, 완전 이진 트리에 근거한 묵시 힙인 2-힙이 이용하는 데이터 지역성 사용 형태는 현대 컴퓨터 시스템 메모리 계층 구조의 지역성을 충분히 활용하지 못하고 있다[5, 6]. 즉, DelMax() 함수 실행시, 트리의 각 레벨에서 두 형제 노드를 접근하게 되는데, 이 형제 노드들는 일반적으로 케쉬 메모리 블록의 일부만을 차지한다. 형제 노드가 서로 다른 캐쉬 블록에 있게 되는 경우, 우선순위 큐 연산 동안 트리의 각 레벨에서 캐쉬 미쓰가 두 번까지 증가되기 때문에 더욱 성능 저하를 일으킨다.

현대의 컴퓨터 시스템은 블록 크기가 $2^i(i > 0)$인 캐쉬 메모리를 가지고 있는데, 보통 32 또는 64 바이트도 구성되어 있다. 캐쉬 블록 크기가 32바이트이고 데이터 크기가 4 바이트 일 경우, 8개의 데이터가 하나의 캐쉬 블록에 적재된다. 따라서, 8개의 형제 노드로 구성된 묵시 힙을 구성하여 모든 형제 노드들을 하나의 캐쉬 블록에 모두 저장할

† 종신회원 : 성신여자대학교 컴퓨터정보학부 교수
논문접수 : 2004년 2월 21일, 심사완료 : 2004년 6월 10일

경우, 우선순위 큐 연산 동안 캐쉬 메모리를 효과적으로 이용하여 성능을 향상시킬 수 있게 된다.

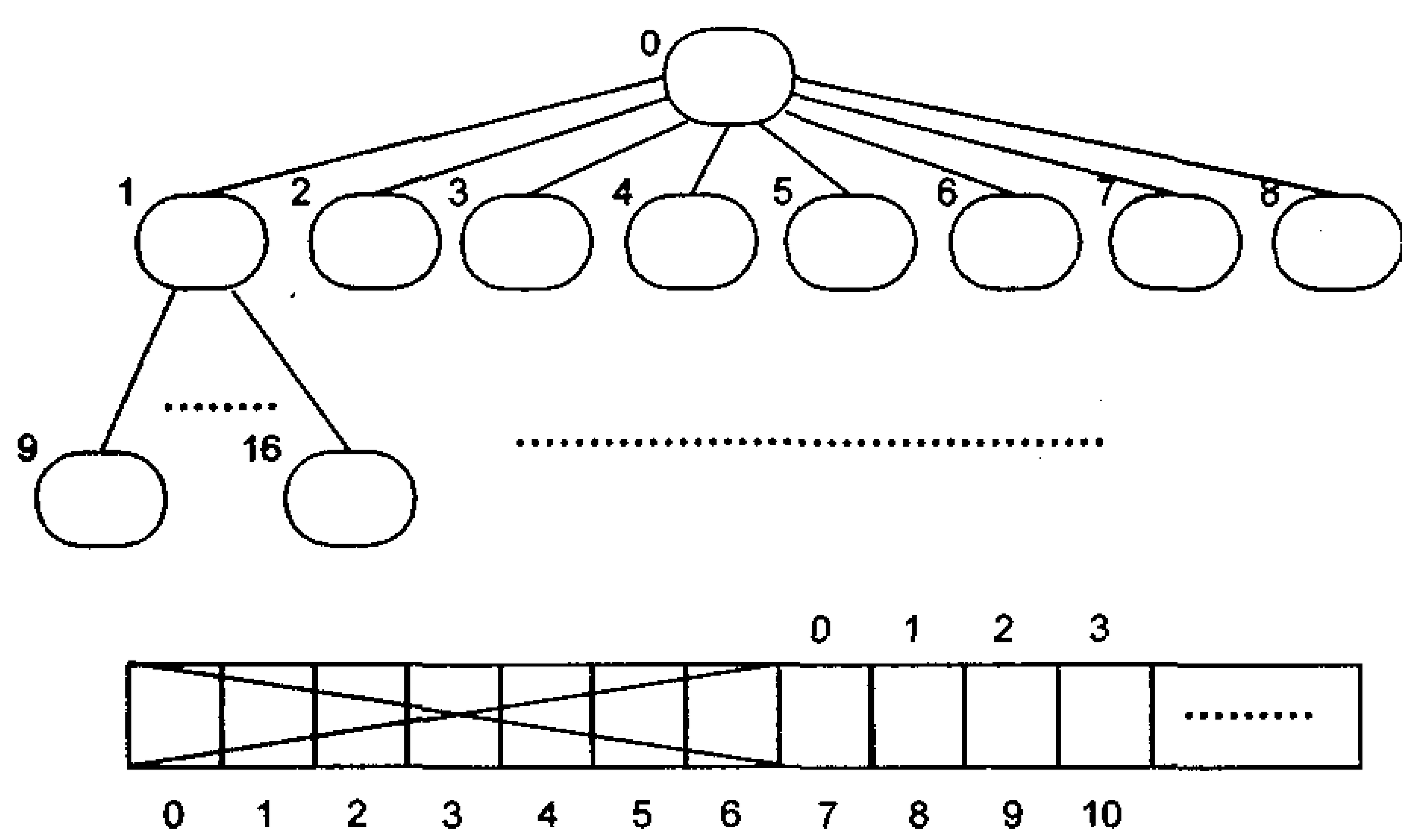
어떤 집합 S를 2-힙으로 구성할 경우 다음 (그림 1)과 같이 나타난다. 각 노드 밖의 수는 2-힙을 배열로 표현할 경우의 배열 인덱스를 나타낸다. 이 경우, 부모-자식 관계는 인덱스 i를 가진 어떤 노드에 대해 $parent(i) = \lfloor i/2 \rfloor$ 및 $children(i) = 2i, 2i+1$로 표현된다.



(그림 1) 2-힙 구조 및 배열 표현

(그림 1)의 경우 배열의 인덱스 0는 사용되지 않는다. 각 노드가 4바이트 크기이고 캐쉬 블록의 크기가 32바이트일 경우, 하나의 캐쉬 블록에 8개 노드가 저장된다. 어떤 캐쉬 블록에 노드 0으로부터 적재를 시작하면, 노드 0에서 7까지가 하나의 캐쉬 블록에 적재되고, 그 다음 8개의 노드가 다른 캐쉬 블록에 저장된다.

이렇게 저장할 경우 모든 형제 노드는 동일한 캐쉬 블록에 저장되어, 우선순위 큐 연산 동안 각 레벨에서의 캐쉬 미스가 최대 1번 이상 발생되지 않는다. 그러나, 연산 동안 각 레벨에서 캐쉬 블록의 일부만 접근하므로 캐쉬 메모리를 효과적으로 사용하지 못하고 있다. 이를 보완하기 위한 8원 트리에 근거한 8-힙은 (그림 2)와 같이 표현된다[5]. 8-힙의 부모-자식 관계는 인덱스 i를 가진 어떤 노드에 대해 $parent(i) = \lfloor (i-1)/8 \rfloor$ 및 $children(i) = 8i+1, 8i+2, \cdots, 8i+8$로 표현된다.



(그림 2) 8-힙 구조 및 배열 표현

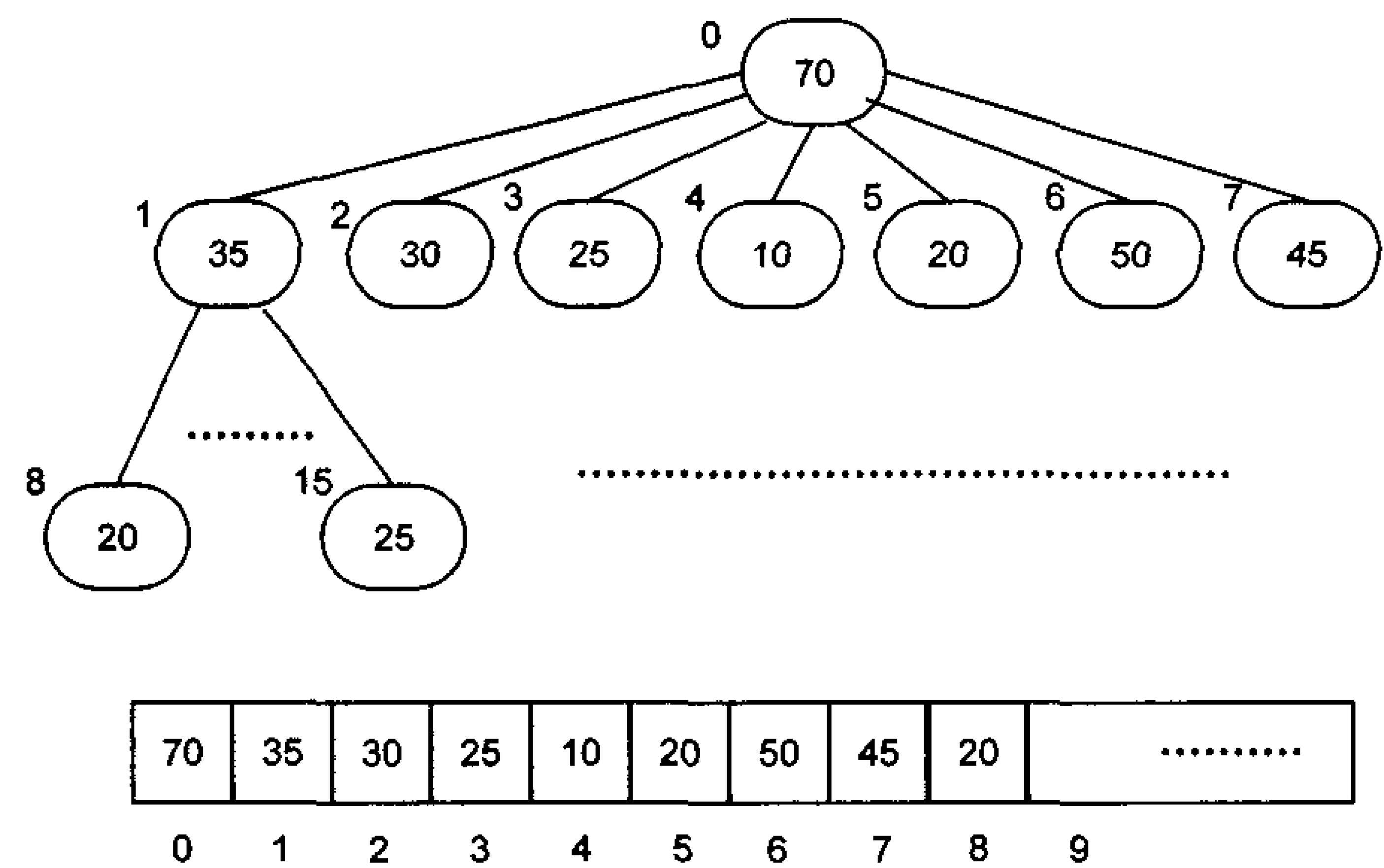
8-힙에서는 모든 형제 노드를 동일한 캐쉬 블록에 적재되도록 하기 위해, 배열 인덱스 0에서 6까지는 사용하지 않고 루트 노드는 배열 인덱스 7에 저장한다. 이렇게 함으로써, 형제 노드 1~8, 9~16, 등이 동일한 캐쉬 블록에 저장된다. 이 경우, 노드 인덱스와 배열의 인덱스가 일치하지 않기 때문에, 구현 시 인덱스 값의 조정이 필요하다.

본 논문에서는 캐쉬 메모리를 효율적으로 이용하고 노드 인덱스와 배열 인덱스가 일치하여 구현하기 쉬운 새로운 묵시 우선순위 큐인 8-힙*를 제안한다. 또한, 성능 실험을 통하여 제안된 8-힙*가 2-힙이나 8-힙보다 빠름을 보인다. 다음 절에서 8-힙*에 대해 설명하고, 3장에서 8-힙*에서의 연산에 대해 기술한다. 제 4장에서는 성능 실험 및 그 결과를 보이고, 5장에서 결론을 맺도록 한다.

본 논문에서는 특별한 언급이 없는 한 최대 힙을 단순히 힙이라 하고, 인덱스 i를 가진 노드를 노드 i라 한다.

2. 8-힙* 자료 구조

(그림 3)에 나타난 바와 같이, 8-힙*는 인덱스 0를 가진 노드를 루트 노드로 하고, 루트 노드는 7개의 자식 노드를 가지며, 루트 노드를 제외한 모든 노드는 8개의 자식 노드를 가진다.



(그림 3) 8-힙*

8-힙*에서의 부모-자식 관계는 노드 i에 대하여 다음과 같이 수식으로 표현된다.

- $parent(i) = \lfloor i/8 \rfloor$
- $children(i) = 8i, 8i+1, \cdots, 8i+7$

이렇게 표현된 8-힙*는 노드 인덱스와 배열 인덱스가 정확히 일치하게 되고, 형제 노드들은 항상 동일한 캐쉬 블록에 적재된다. 즉, 루트 노드와 루트 노드의 모든 자식 노드들이 하나의 캐쉬 블록에 적재되고(노드 0~7), 노드 1의 모든 자식 노드들이 그 다음 캐쉬 블록에 들어가며(노드 8~15), 계속 순서대로 대응되는 캐쉬 블록에 8개의 노드

씩 적재된다.

본 논문에서 제안한 8-힙*는 다음과 같은 성질을 가진다.

- 인덱스 0인 루트 노드는 7 개의 자식 노드를 가진다.
- 루트 노드를 제외한 모든 노드는 8 개의 자식 노드를 가진다.

3. 8-힙* 연산

8-힙* 연산은 자식 노드의 개수에서 나타나는 차이점 외에는 전통적인 2-힙의 연산과 동일하다. Insert()와 DelMax() 함수는 (알고리즘 1)과 (알고리즘 2)에 각각 나타나 있다. 여기서, 데이터는 배열 A[]에 저장되어 있고, 배열의 마지막 데이터는 인덱스 n에 의해 지정된다고 가정한다.

데이터의 삽입은 (알고리즘 1)에 나타난 바와 같이, 제일 마지막 노드 다음 노드로부터 시작하여 상향식으로 이루어진다. while 문을 실행 중 부모 노드가 더 큰 키 값을 가지고 있으면 입력 데이터 e의 삽입이 이루어진다.

```
void Insert( Element e )
{ n++ ; q = n ; p = q/8 ;
  while( q != 0 ) {
     if( c.key <= A[p].key ) break ;
     A[q] = A[p] ;
     q = p ; p = q/8 ;
  }
  A[q] = e ; return ;
}
```

(알고리즘 1) 최대 8-힙* 삽입 알고리즘

(알고리즘 2)에 나타난바와 같이 최대 데이터의 삭제는 삽입과는 반대로 하향식으로 이루어진다. 배열 A[0]에 있는 삭제될 최대 키 값을 가지는 데이터를 변수 e에 임시 저장하고, 재 삽입될 데이터 A[n]이 추출되어 while 문을 통하여 삽입될 위치가 결정된다. while 문에서는 루트 노드로부터 시작하여 자식 노드 중 최대키를 가진 노드 c를 찾고, 그 노드 키가 재삽입될 데이터의 키보다 작으면 재삽입 데이터를 삽입한 후 종료한다. 그렇지 않으면, 그 자식 노드를 노드 q로 하여 반복하게 된다.

```
Element DelMax( void )
{ Element e = A[0] ;
  tmp = A[n] ; n-- ;
  q = 0 ;
  while( node q의 자식이 있는 동안 ) {
    c = 노드 q의 자식들 중 최대 키 값을 가진 노드 ;
    if( tmp.key >= A[c].key ) break ;
    A[q] = A[c] ; q = c ;
  }
  A[q] = tmp; return e ;
}
```

(알고리즘 2) 최대 8-힙* 삭제 알고리즘

[정리 1] 우선순위 큐 연산 동안, 최악의 경우 2-힙에 대한 8-힙*의 캐쉬 블록 접근수 비율은 1/3이다. 여기서, 모든 형제 노드는 동일한 캐쉬 블록에 들어 있다고 가정한다.

[증명] 8-힙*의 경우 연산동안 각 레벨에서 많아야 하나의 캐쉬 블록을 접근 한다. 따라서, 접근된 캐쉬 블록 수는 8-힙*의 높이인 $S_8 = \log_8 n$이 된다. 2-힙의 경우에 있어서도 모든 형제 노드는 동일한 캐쉬 블록에 있으므로, 캐쉬 블록 접근 수는 $S_2 = \log_2 n$이 된다. 따라서, 캐쉬 블록 접근 비율은 $S_8/S_2 = 1/3$이 된다. ■

따라서, 8-힙* 연산 동안의 캐쉬 미쓰가 최악의 경우 2-힙보다 1/3로 줄어든다.

4. 실험 결과

성능 비교를 위해 각 노드 크기가 4 바이트인 2-힙, 8-힙 및 8-힙*를 C++로 구현하여, 1기가 바이트의 메인 메모리와 32 바이트 L1 캐쉬 블록 크기를 가지고 있는 Ultra-sparc 워크스테이션에서 실험이 이루어졌다. 구현된 모든 힙에 대해 형제 노드들은 항상 동일한 캐쉬 블록에 적재되도록 했으며, 실행 파일은 컴파일러 g++의 최대 최적화 옵션인 O3를 이용하여 생성되었다. 실험은 다음의 두 가지 시험 모델에 대해 이루어 졌다.

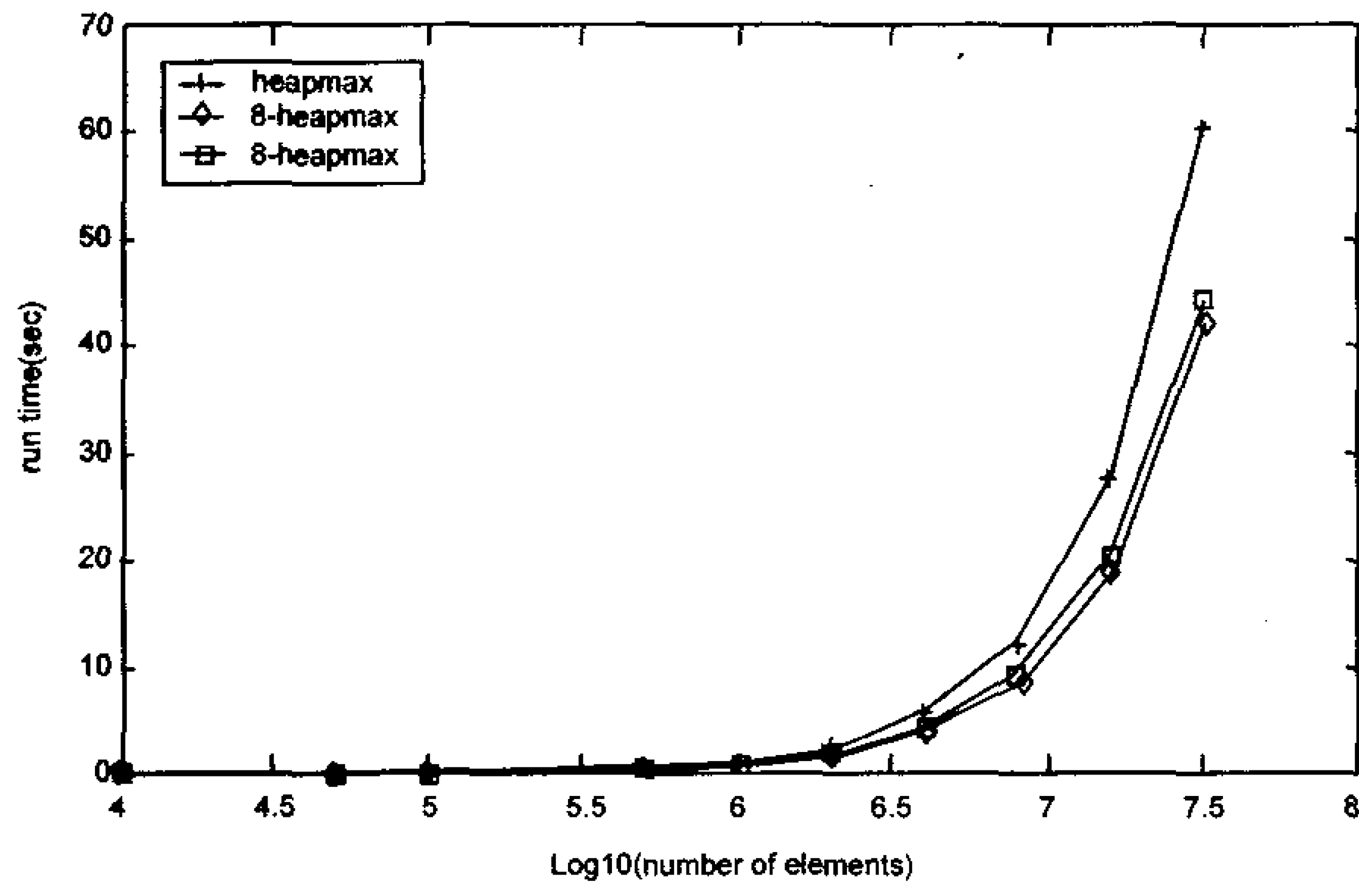
- 홀드(hold) 모델[7] : n/2개의 무작위 데이터로 초기화한 후, n개의 무작위 데이터를 삽입과 삭제를 혼합하여 시험한다. 이때, 삽입과 삭제는 각각 50%가 되게 하고, 자료 구조는 대략 n/2개의 데이터를 유지하도록 하였다. 시간 측정은 n개의 혼합된 삽입/삭제 연산에 대해 이루어졌다.
- 정렬 모델 : 자료 구조가 비어있는 상태에서 시작하여, n개의 무작위로 발생된 데이터를 삽입하고, 삽입된 모든 데이터를 내림차순으로 삭제한다. n개의 삽입과 n개의 삭제가 실행된 시간이 측정되었다.

위의 각 실험 모델에 대해 n = 10,000(10K), 50K, 100K, 500K, 1M(백만), 2M, 4M, 16M, 및 32M 개의 데이터에 대한 실행 시간이 측정되었는데, 다음 (그림 4)와 (그림 5)에 나타난 실행 시간은 15회 실행의 평균값이다.

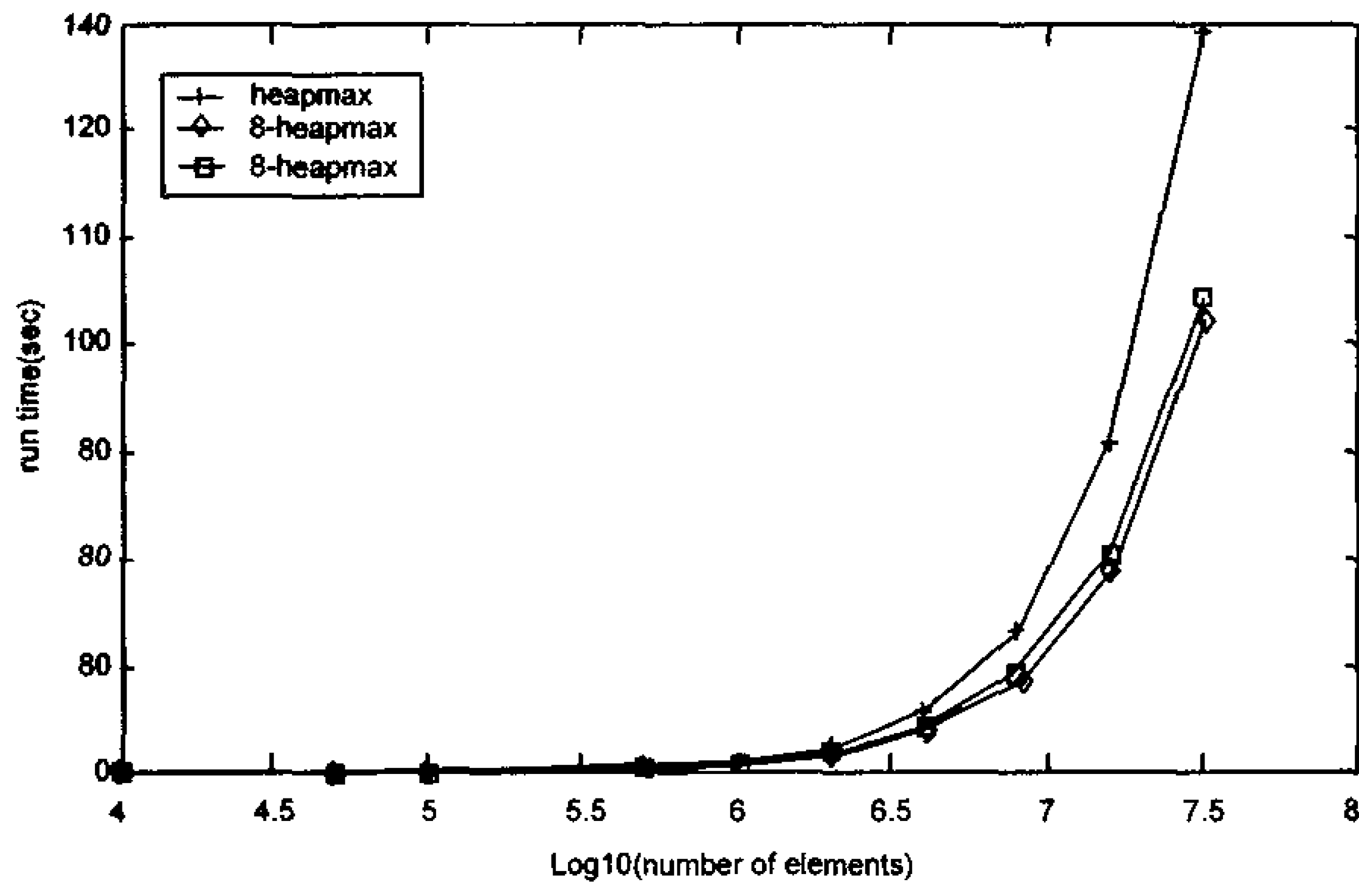
(그림 4)는 홀드 시험 모델에서 측정된 실행 시간을 보여준다. 그림에서 보인 것처럼 8-힙*(8-heapmax*로 표기)가 2-힙(heapmax로 표기)뿐만 아니라 8-힙(8-heapmax로 표기)보다도 더 빠른 것으로 나타났다. n=32M 데이터의 경우 2-힙보다 약 50%의 성능 개선이 이루어졌다.

(그림 5)는 정렬 시험 모델에서 측정된 실험 결과를 보여주고 있다. 홀드 모델에서와 비슷하게, 실험에 사용된 데이

터 수 n이 증가함에 따라 실행 속도 차가 증가함을 볼 수 있다. n = 32M 데이터에 대해 2-힙보다 약 65%의 성능이 개선되었다.



(그림 4) 홀드 모델에서 측정된 실행 시간



(그림 5) 정렬 모델에서 측정된 실행 시간

5. 결 론

본 논문에서는 8-원 트리에 근거한 새로운 묵시 우선순위 큐인 8-힙*를 제안했다. 8-힙*는 노드 인덱스와 배열 인덱스가 일치하기 때문에 8-힙보다 구현이 용이할 뿐만 아니라, 성능에 있어서도 기존의 묵시 힙보다 우수한 것으로 나타났다.

성능 비교는 전통적인 완전 이진 트리에 근거한 2-힙과 8원 트리에 근거한 8-힙에 대하여 이루어 졌고, 성능 측정에 많이 이용되는 홀드 시험 모델과 정렬 시험 모델이 사용되었다. 성능 측정 결과, 새로이 제안된 8-힙*가 실험 데이터 수가 증가함에 따라 기존의 2-힙 뿐만 아니라 8-힙보다도 빠른 것으로 나타났다.

참 고 문 헌

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

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

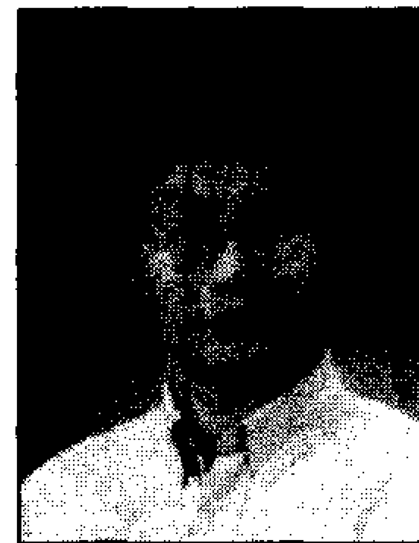
[3] T. J. Stasko and J. S. Vitter, "Pairing heaps : experiments and analysis," Communications of the ACM, 30(3), pp.234-249, 1987.

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

[5] A. LaMarca and R. Ladner, "The Influence of Caches on the Performance of Heaps," ACM Jounrnal of Experimental Algorithmics, 1(4), 1996.

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

[7] D. Jones, "An empirical comparison of priority-queue and event-set implementation," Communications of the Association for Computing Machinery, 29(4), pp.300-311, 1986.



정 해 재

e-mail : hjjung@cs.sungshin.ac.kr
1984년 경북대학교 전자공학과(전산전공)(공학사)
1987년 서울대학교 컴퓨터공학과(공학석사)
2000년 University of Florida 컴퓨터정보학과(공학박사)
1988년 1995년 한국전자통신연구원 선임연구원
2001년~2002년 Numerical Technologies Inc. USA, Staff Engineer
2002년~2003년 서울대학교 컴퓨터신기술연구소 객원연구원
2003년~현재 성신여자대학교 컴퓨터정보학부 초빙교수
관심분야 : 컴퓨터 알고리즘 및 자료 구조, 계산 기하, 컴퓨터 비젼