DOI: 10.3745/KIPSTA.2009.16-A.3.217

상수 삽입 전이 시간을 가지는 양단 우선순위 큐

정 해 재[†]

요 약

우선순위 큐는 스케줄링, 정렬, 유전자 검색과 같은 우선순위에 따른 검색, 최단거리 계산과 같은 응용에 사용될 수 있다.

본 논문에서 제안하는 배열을 이용한 양단 우선순위 큐 자료구조는 삽입과 삭제 연산에 각각 O(1) 전이시간과 O(logn) 시간이 걸린다. 본 저자가 알고 있는 한, 지금까지의 배열을 이용한 양단우선순위 큐 알고리즘은 삽입과 삭제에 모두 O(logn) 시간이 걸린다.

키워드 : 자료 구조, 양단 우선순위 큐, 힙, 전이 시간 복잡도

A Double-Ended Priority Queue with O(1) Insertion Amortized Time

Haejae Jung[†]

ABSTRACT

Priority queues can be used in applications such as scheduling, sorting, retrival based on a priority like gene searching, shortest paths computation.

This paper proposes a data structure using array representation for double-ended priority queue in which insertion and deletion takes O(1) amortized time and O(logn) time, respectively. To the author's knowledge, all the published array-based data structures for double ended priority queue support O(logn) time insertion and deletion operations.

Keywords : Data Structure, Double-Ended Priority Queue, Heap, Amortized Time Complexity

1. 서 론

양단 우선순위 큐(double-ended priority queue)는 어떤 데이터 집합 S에서 특정 우선순위에 따른 최소값과 최대값을 빨리 찾고자 고안된 동적 자료구조로서, 다음의 세 가지 연산을 기본으로 지원하며, 이 자료구조는 운영체제 스케줄링, 사건 시뮬레이션, 유전자 검색과 같은 우선순위에 따른 검색, 또는 정렬과 같은 응용에 이용될 수 있다.

- insert(e, S): 집합 S에 새로운 임의의 데이터 e를 추가.
- delmax(S): 집합 S로부터 우선순위가 가장 높은 데이터를 삭제.
- delmin(S): 집합 S로부터 우선순위가 가장 낮은 데이터를 삭제.

우선순위 큐는 부모-자식 관계를 나타내기 위해 포인터를 사용하거나 일차원 배열 인덱스를 사용하여 구현할 수 있다. 포인터를 사용하는 경우 부모 노드와 자식 노드를 포인터로 연결하여 부모-자식 관계를 나타내고, 배열을 사용하는 경우 인덱스 값에 관한 간단한 수식을 사용하여 부모-자식 관계를 묵시적으로 나타낸다.

배열을 이용한 단일 우선순위 큐(single-ended priority queue)로는 O(logn) 삽입 및 삭제 시간을 가진 전통적인 힙, 상수 삽입 전이시간(amortized time)과 O(logn) 삭제 시간 복잡도를 가지는 묵시 이항(binomial) 큐, 및 M-힙의 배열 버전인 MA-힙 등이 있다 [1-4]. MA-힙은 최대 O(logn)개의 내부힙으로 구성되며, 삽입은 O(1) 전이 시간 복잡도(amortized time complexity)를 가지고, 삭제는 O(logn) 시간 복잡도를 가진다[1]. MA-힙은 묵시 이항 큐보다 자료구조가 훨씬 단순하여 구현하기가 상당히 용이하다.

본 논문에서 관심이 있는 배열을 이용한 양단 우선순위 큐에는 최대-최소 힙(min-max heap), 딥(deap), d-딥(d-deap), 대칭 최대-최소 힙(symmetric min-max heap), 구간 힙(interval heap) 등이 있으며, 이들은 모두 O(logn) 삽입 및 삭제 시간 복잡도를 가진다 [5-8]. 이 중 구간 힙의 구조는 전통적인 힙과 같은 완전 이진 트리(complete binary tree) 형태를 가지지만, 제일 마지막 노드를 제외한 각 노드에는 구간의 좌단값과 우단값을 나타내는 두 개의 값 <lval, rval>을 유지

※ 이 논문은 2006년도 안동대학교 특성화 추진 지원사업에 의하여 연구되었음.
† 종신회원 : 안동대학교 정보통신공학과 교수
논문접수 : 2009년 1월 22일
수 정 일 : 1차 2009년 4월 21일
심사완료 : 2009년 4월 30일

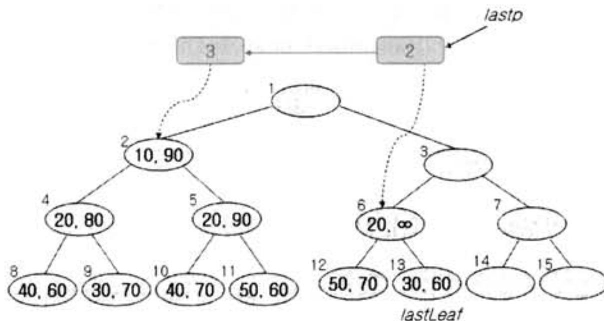
하고, lval <= rval의 관계를 가진다. 삽입, 삭제 연산을 통하여 부모 노드의 구간은 자식 노드의 구간을 항상 포함하도록 유지한다 [5, 6].

본 논문에서는 상수 삽입 전이 시간과 O(logn) 최대값/최소값 삭제 시간을 가지는 배열을 이용한 양단 우선순위 큐 MI-힙을 제안하며, 제안된 자료 구조는 MA-힙의 자료구조의 특성과 구간힙의 특성을 기반으로 하여 고안된 구조이다. 본 저자가 알고 있는 한, 지금까지의 배열을 이용한 모든 양단 우선순위 큐에 대한 모든 자료구조는 O(logn) 삽입과 O(logn) 최대값/최소값 삭제 시간 복잡도를 가진다.

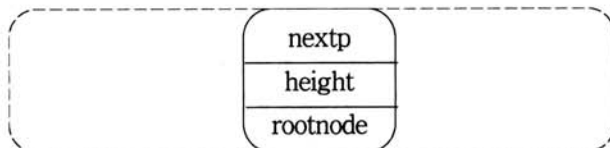
2. MI-힙 자료 구조

MI-힙은 최대 n 개의 데이터를 저장하기 위해 $2^{\lfloor \log n \rfloor + 1} - 1$ 개의 노드를 가지는 포화 이진 트리 (full binary tree)로 표현되며, 최대 O(logn) 개의 내부힙으로 구성된다 [3]. 데이터를 가지고 있는 내부힙의 각 루트 노드는 연결 리스트를 통하여 연결된다. (그림 1)은 노드 2와 노드 6에 루트를 가진 두 개의 내부힙으로 구성된 MI-힙의 예를 나타내며, 각 내부힙의 루트 노드는 연결 리스트의 노드에 의해 참조되고 연결 리스트의 각 노드는 lastp로부터 접근될 수 있다. MI-힙에서는 삽입과 삭제 연산을 하기 위해서 두 개의 변수 lastLeaf와 lastp를 유지한다. lastLeaf는 데이터를 가진 노드 중 가장 오른쪽 리프 노드를 나타내고, lastp는 가장 오른쪽 연결 리스트 노드를 참조하여 마지막 내부힙 즉 가장 오른쪽 내부힙의 루트 노드를 접근할 수 있도록 하고 있다.

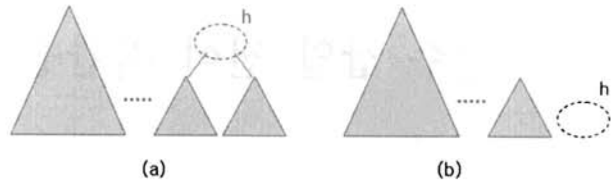
MI-힙에서의는 각 노드에 좌단값(lval)과 우단값(rval)의 두 값 <lval, rval>을 저장하여 구간을 나타내는데, 그 두 값의 관계는 lval <= rval 이다. MI-힙에서의 부모 노드의 구간은 항상 자식 노드의 구간을 포함하는 관계를 가진다. 예를 들면, (그림 1)에서 노드 4와 5의 구간은 노드 2의 구간에 포함된다.



(그림 1) MI-힙의 예



(그림 2) 연결리스트 노드 구조



(그림 3) MI-힙에서의 연산

lastp를 통하여 연결된 연결 리스트의 각 노드는 (그림 3)과 같은 구조를 가진다. nextp는 연결 리스트의 다음 노드를 가리키고, rootnode는 내부힙의 루트 노드 인덱스 값을 저장하며, height는 rootnode에 의해 참조되는 내부힙의 높이를 나타낸다. (그림 1)에서의 연결리스트 노드에 있는 값은 height 값이다.

MI-힙에서의 실질적인 데이터 삽입과 삭제는 MI-힙에서의 가장 오른쪽에 있는 마지막 내부힙의 루트 노드에서 이루어지며, 그 노드의 rval은 실제 저장되는 어떠한 값보다도 크게 정의된 값((그림 1)에서 ∞로 표시)이 저장될 수 있다. (그림 3)은 연결 리스트를 생략한 MI-힙의 구조를 나타내고 있다. 삽입 연산 시 마지막 내부힙 루트 노드의 rval이 ∞인 경우, 새 데이터로 ∞ 값을 대체한다. 그렇지 않은 경우 마지막 두 내부힙의 높이를 비교한다. 높이가 같은 경우, (그림 3)의 (a)와 같이 마지막 두 내부힙의 루트 노드의 부모 노드 h에 새로운 데이터와 ∞ 값를 삽입하고, 노드 h를 새로운 마지막 내부힙의 루트로 만든다. 높이가 같지 않을 경우, (그림 3)의 (b)와 같이 하나의 노드로 된 새로운 내부힙 h에 새 데이터와 ∞ 값을 삽입하고, 노드 h를 마지막 내부힙의 루트 노드로 만든다. 삭제의 경우 실제 데이터의 삭제는 마지막 내부힙의 루트 노드에서 일어난다. 즉, 먼저 각 내부힙의 루트 노드를 조사하여 최대 또는 최소 값을 찾고, 그 값을 마지막 내부힙의 rval (rval이 ∞이면 lval)로 대체한다. 물론 기술한 삽입 또는 대체 후 힙조정(heapify)이 적절히 이루어진다.

3. MI-힙 연산

3.1 초기화

MA-힙에서는 n 개의 데이터 저장을 위해 크기가 $M = 2^{\lfloor \log n \rfloor + 1}$ 인 배열 A[M]를 할당하여, 포화 이진 트리가 되도록 한다. 참조 변

```
void initialize( numMax )  // numMax : max. number of elements
{
MinLeaf = 2⌊ log2(numMax)⌋ ;        // leftmost leaf node
   MaxLeaf = 2* MinLeaf - 1;  // rightmost leaf node

   // allocate array A[] and
   // initialize lastLeaf and lastp.
A = new Node [MaxLeaf + 1]; // array of nodes
lastLeaf = MinLeaf - 1;
ListNode lastp = null;  // reference of right most linked list node
}
```

(그림 4) MI-힙 초기화

수 lastp는 NULL로 초기화되고, 변수 lastLeaf는 MI-힙의 제일 왼쪽 리프 노드 인덱스보다 1 적은 값으로 초기화 한다.

3.2 힙조정

MI-힙에 새로운 데이터가 삽입되면 더 이상 MI-힙의 성질을 만족하지 못하게 되므로 힙조정을 통하여 MI-힙 성질을 만족하도록 만든다. MI-힙에서는 새로운 데이터가 어떤 내부힙의 루트 노드의 좌단값 또는 우단값으로 삽입될 수 있으며, 힙 조정은 하향식으로 이루어진다. 새로운 데이터가 우단값으로 상입될 경우, 우단값으로만 형성되는 최대 힙를 따라 하향식으로 조정이 이루어지는데, 그 알고리즘은 (그림 5)에 나타나 있다. 좌단값으로만 구성되는 최소힙을 따라 하향식으로 힙조정을 하는 heapify_lheap( )도 이와 유사하게 작성할 수 있다.

함수 heapify_rheap()의 인수 k는 새로운 우단값을 가지고 있는 노드 인덱스를 나타낸다. 함수 cmp_swap()는 특정 노드의 좌단값과 우단값을 비교하여 좌단값이 우단 값보다 크면 서로 교환해 주는 함수이다. heapify_rheap() 함수에서 노드 k의 두 자식 노드의 우단값 중 큰 값을 가지는 노드를 max로 두고, 노드 max가 부모 노드 k의 우단값보다 크면 두 우단값을 서로 교환한다. 이러한 과정은 부모 노드의 우단값이 자식노드의 우단값 중 큰 값보다 크면 while 반복문을 빠져나와 힙조정을 종료한다.

```
// k : root node to start to heapify
void heapify_rheap( k )
{
   cmp_swap( A[k].lval, A[k].rval );
   left = 2*k;  // left child of k
   while( left < MaxLeaf ) {  // 자식이 있는 동안
      max = left;
      if( A[max].rval < A[left+1].rval )
         max = left + 1;     // right child of k.
      if( A[k].rval > A[max].rval ) break;
      swap( A[k].rval, A[max].rval );
      cmp_swap( A[max].lval, A[max].rval );
      k = max;  left = 2*k;
   }
}
```

(그림 5) 우단 힙조정 알고리즘

3.3 삽입

(그림 6)은 MI-힙 삽입 알고리즘을 보여주고 있다. 인수로 전달된 thedata를 삽입하기 위해 마지막 내부힙의 루트 노드가 한 개의 데이터만 가지고 있는지 알아본다. 이 경우 인수로 전달된 데이터로 rval인 INF(∞)값을 대체하고 heapify_rheap() 함수를 호출하여 힙조정을 한다.

마지막 두 내부힙의 높이가 같은 경우에는 마지막 내부힙 루트노드의 부모노드에 인수 thedata와 INF 값을 각각 lval과 rval로 저장하고, heapify_lheap()을 호출하여 힙조정을 한다. 마지막 리스트 노드의 왼쪽 노드는 삭제하고, 마지막 리스트 노드의 rootnode와 height 필드를 수정한다.

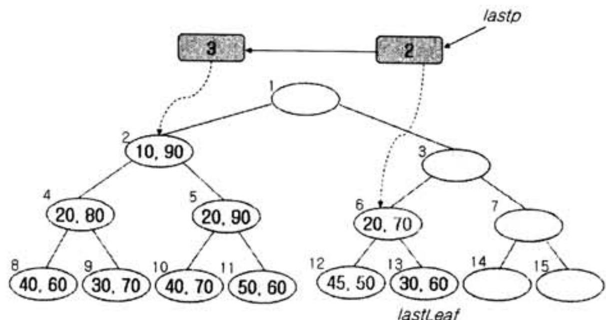
마지막 두 내부힙의 높이가 다른 경우, lastLeaf를 1 증가하여 새로운 내부힙을 생성하고, 이 노드의 lval과 rval 필드에 thedata와 INF 값을 각각 삽입한다. 이 경우 새로운 리스트 노드를 하나 생성하여 마지막 리스트 노드로 만들고, 이 노드의 rootnode와 height 값은 lastLeaf와 1이 된다.

(그림 7)과 (그림 8)은 (그림 1)에 데이터 45와 35를 연속으로 삽입한 후의 MI-힙을 보여주고 있다. (그림 7)은 45로 INF값을 대체한 후 힙조정한 결과이고, (그림 8)은 35를 새로운 내부힙 루트 노드 14에 삽입한 결과이다.

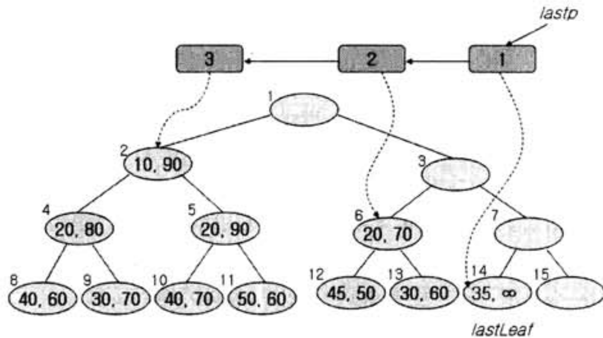
```
bool insert( Element thedata )
{
   // check if the rval of the last root is infinite value.
   if( lastp != NULL && A[lastp.rootnode].rval == INF ) {
      lastroot = lastp.rootnode;
      A[lastroot].rval = thedata;
      heapify_rheap( lastroot );
      return TRUE;
   }

   // Now, heap is empty, or the last root node has two values.
   h1 = -1;  h2 = -2;
   if( lastp ) {
      if( lastp.rootnode == 1 ) return FALSE;  // MI-힙 is full.
      h1 = lastp.height;     // 마지막 내부힙 높이
      sp = lastp.nextp;      // 두 번째 리스트 노드
      if( sp ) h2 = sp.height; // 마지막 왼쪽 내부힙 높이
   }
   if( h1 == h2 ) {
      parnode = lastp.rootnode / 2;
      A[parnode].lval = thedata;  A[parnode].rval = INF;
      heapify_lheap( parnode );
      lastp.rootnode = parnode;  lastp.height++;
      lastp.nextp = sp.nextp;
      delete sp;  // 두 번째 리스트 노드 삭제
   } else {
      lastLeaf++;
      A[lastLeaf].lval = thedata;  A[lastLeaf].rval = INF;
      np = new ListNode;
      np.rootnode = lastLeaf;  np.height = 1;
      np.nextp = lastp;        lastp = np;
   }
   return TRUE;
}
```

(그림 6) 삽입 알고리즘



(그림 7) (그림 1)에 45를 삽입한 후



(그림 8) (그림 7)에 35를 삽입한 후

[정리 1] n 개의 데이터를 가지고 있는 MI-힙에서, 삽입 연산은 $O(1)$ 전이 시간 복잡도를 가진다.

(증명) 리스트 노드의 삽입 및 삭제와 변수 lastLeaf의 증가는 상수 시간 걸리고, m개의 노드를 가진 내부힙의 힙조정은 트리 높이인 $O(\log m)$ 시간이 걸린다.

삽입 전이 시간 복잡도를 구하기 위해, 일련의 삽입 연산이 이루어진다고 하자. 일련의 삽입 중, 두 개의 내부힙으로 구성된 MI-힙에 새로운 데이터를 삽입할 경우 하나의 내부힙만으로 구성된 MI-힙이 되는데, 이때 최대 비용이 소요된다. 삽입 시의 힙조정은 항상 이웃하는 두 내부힙의 부모 노드에서 시작하여 리프 노드 쪽으로 이루어지므로, 그 부모 노드의 높이 만큼의 시간이 걸린다. 따라서, 하나의 내부힙으로 구성된 MI-힙의 높이가 h라 할 때, 일련의 삽입 총 비용을 계산하면 $T=\sum_{i=1}^{h} O(2i)2^{h-i}$가 된다. 즉, 트리 레벨 i 에는 2^{h-i}개의 노드가 있고, 각 노드에 대한 2개의 데이터 삽입 비용은 $O(2i)$가 된다. 따라서 총 비용을 계산하면, $T= O(2^{h+1}\sum_{i=1}^{h}(i/2^i)) = O(2^{h+1} \cdot 2) = O(n)$이 되므로, 각 데이터 삽입에 대한 전이 시간 복잡도는 $O(n)/n = O(1)$이 된다. ■

3.4 최대값 삭제

최대값은 내부힙들 중 어떤 한 내부힙의 루트 노드 또는 마지막 내부힙의 루트 노드의 자식 노드에 있을 수 있다. (그림 9)에서 보는 바와 같이 마지막 내부힙의 rval이 INF 값을 가지고 있는 경우 그 자식 노드의 값도 조사하여 최대값을 가진 노드 및 값을 각각 maxnode 및 maxval로 둔다. 마지막 내부힙을 제외한 내부힙에 대해서는 루트 노드의 rval만 비교하면 되며, while 문을 통하여 최대값을 가진 maxnode를 찾게 된다.

MI-힙에서의 최대값 삭제는 최대값을 가지고 있는 내부힙을 찾은 후, 그 최대값을 마지막 내부힙의 루트 노드 값으로 대체하고, 대체된 내부힙에 대해 힙조정을 함으로써 이루어진다. (그림 10)의 최대값 삭제 알고리즘은 우선 find_maxnode() 함수를 호출하여 최대값을 가진 노드 maxnode를 찾는다.

```
int find_maxnode( void )
{
    maxnode = lastp.rootnode;
    leftchild = 2 * maxnode;
    if( maxnode.rval == INF && leftchild <= MaxLeaf ) {
        rightchild = leftchild +1;
        maxnode = leftchild;
        if( A[leftchild].rval < A[rightchild].rval ) maxnode = rightchild;
    }
    maxval = A[maxnode].rval;
    if( A[maxnode].rval == INF )  maxval = A[maxnode].lval;

    tmp = lastp.nextp;
    while( tmp ) {
        if( A[tmp.rootnode].rval > maxval ) {
            maxnode = tmp.rootnode;
            maxval = A[maxnode].rval;
        }
        tmp = tmp.nextp;
    }

    return maxnode; // node index with maxval
}
```

(그림 9) 최대값을 가진 노드 찾기 함수

```
Element delmax( )
{
    if( lastp == NULL ) return -1; // MI-힙 is empty.

    maxnode = find_maxnode( ); // find the node with max. element
    retdata = A[maxnode].rval;
    if( A[maxnode].rval == INF ) { // maxnode == lastroot : leaf node
        retdata = A[maxnode].lval;
        lastLeaf--; tmp = lastp; lastp = lastp.nextp; delete tmp;
        return retdata;
    }

    lastroot = lastp.rootnode;
    if( A[lastroot].rval == INF ) { // lastroot has children, maxnode != lastroot
        A[maxnode].rval = A[lastroot].lval;
        heapify_rheap( maxnode );
        if( 2*lastroot < MaxLeaf ) split_heap( );
        else { lastLeaf--; tmp = lastp; lastp = lastp.nextp; delete tmp; }
        return retdata;
    }

    // Now, there is no INF value in MI-heap.
    if( lastroot != maxnode ) {
        A[maxnode].rval = A[lastroot].rval;
        heapify_rheap( maxnode );
    }
    A[lastroot].rval = INF;

    return retdata;
}
```

(그림 10) 최대값 삭제 알고리즘

maxnode의 rval이 INF이면 maxnode는 단 하나의 리프노드만을 가진 마지막 내부힙이므로 마지막 내부힙을 삭제하고 최대값을 리턴한다. 그렇지 않고 마지막 내부힙 루트노드의 rval이 INF이면, 그 노드의 lval을 maxnode의 rval로 대체한 후 힙조정을 한다. 힙 조정 후, 마지막 내부힙 루트 노드가

자식을 가지고 있으면 split_heap( ) 함수를 호출하여 마지막 내부힙의 루트 노드의 오른쪽 자식을 마지막 내부힙 루트 노드로 만들고, 자식이 없으면 마지막 내부힙을 제거한다. 위의 경우가 아니면, 마지막 내부힙의 rval을 maxnode의 rval로 복사한 후 maxnode로부터 하향식으로 힙조정을 한다. 마지막 내부힙 루트 노드인 lastroot의 rval에는 INF 값을 삽입한 후, 최대값을 리턴한다.

(그림 11)의 split_heap( ) 함수는 마지막 내부힙의 루트 노드에 더 이상 데이터가 없을 경우 그 자식들을 루트로 하는 두 개의 내부힙으로 분리하는 함수이다. 이 때, 마지막 두 내부힙의 높이는 1 감소한다.

(그림 12)는 기술한 알고리즘에 따라 (그림 8)로부터 최대값 90을 삭제한 결과를 보여주고 있다.

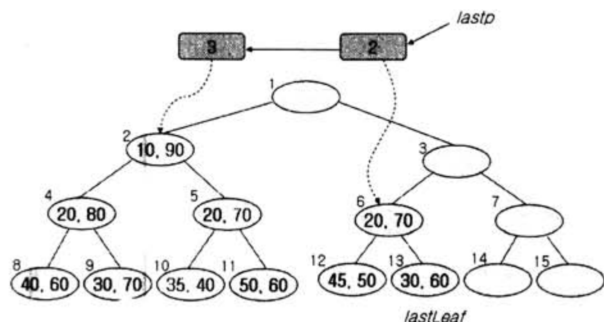
[정리 2] n 개의 데이터를 가지고 있는 MI-힙에서, 최대값 삭제 연산은 $O(\log n)$ 시간 복잡도를 가진다.

(증명) 힙조정 알고리즘 heapify_rheap( )은 힙의 높이 만큼의 시간이 걸리므로 $O(\log n)$ 시간이 걸리고, MI-힙에 최대 O(logn) 개의 내부힙이 존재할 수 있으므로 최대값을 가진 노드를 찾는 함수 find_maxnode( ) 또한 $O(\log n)$ 시간이 걸린다. 그 외의 삭제 알고리즘에서 사용하는 함수 split_heap( )과 문장은 상수 시간이 걸린다. 따라서 최대값 삭제 알고리즘은 $O(\log n)$ 시간 복잡도를 가진다. ■

O(logn) 시간 복잡도를 가지는 최소값 삭제 알고리즘도 최대값 삭제 알고리즘과 유사하며, 구체적인 알고리즘은 [부록]의 (그림 13)와 (그림 14)에 나타나 있다.

```
void split_heap( void )  // 마지막 내부힙 루트가 자식을 가지고 있음.
{
    sp = new ListNode;
    sp.nextp = lastp.nextp;  lastp.nextp = sp;
    leftchild = 2 * lastp.rootnode;
    lastp.rootnode = leftchild+1; lastp.height--;
    sp.rootnode = leftchild;    sp.height = lastp.height;
}
```

(그림 11) 마지막 내부힙의 분할



(그림 12) (그림 8)로부터 최대값 삭제 후 MI-힙

4. 결 론

본 고에서 제안한 MI-힙은 배열을 이용하여 묵시적으로 부모-자식 관계를 표현하는 양단 우선순위 큐 자료구조이다. 배열을 이용한 기존의 양단 우선순위 큐 알고리즘은 삽입과 삭제를 하는데 모두 O(logn) 시간이 걸렸지만, 본 논문에서 제안한 MI-힙에서는 삽입과 최대값/최소값 삭제 연산에 각각 상수 전이 시간과 O(logn) 시간 복잡도를 가진다.

참 고 문 헌

[1] 정해재, "배열 표현을 이용한 M-힙에서의 삽입/삭제 알고리즘", 정보처리학회논문지, 13-A(3), pp.261-266, Jun., 2006.

[2] D. Mehta, and S. Sahni(ed.), Handbook of Data Structures and Applications, Chapman & Hall/CRC, New York, 2005.

[3] 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.

[4] S. Carlsson, J. Munro, and P. Poblete, "An implicit binomial queue with constant insertion time," "Proceedings of the 1st Scandinavian Workshop on Algorithm Theory," Lecture Notes in Computer Science, 318, pp.1-13, July, 1988.

[5] J. van Leeuwen and D. Wood, "Interval heaps," The Computer Journal, 36(3), 209-216, 1993.

[6] Y. Ding and M. Weiss, "On the Complexity of Building an Interval Heap," Information Processing Letters, 50, 143-144, 1994.

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

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

(부 록) 최소값 삭제 알고리즘

```
int find_minnode( void )
{
   minroot = lastp.rootnode;

   tmp = lastp.nextp;
   while( tmp ) {
      if( A[tmp.rootnode].lval < A[minroot].lval ) {
          minroot = tmp.rootroot;
      }
      tmp = tmp.nextp;
   }

   return  minroot;  // node index with minval
}
```

(그림 13) 최소값을 가진 노드 찾기 함수

```
Element delmin( )
{
   if( lastp == NULL ) return -1; // MI-힙 is empty.

   minroot = find_minnode( ); // find the node with min. element
   retdata = A[minroot].lval;
   lastroot = lastp.rootnode;
   A[minroot].lval = A[lastroot].lval;
   heapify_lheap( minroot );

   if( A[lastroot].rval != INF ) {
      A[lastroot].lval = A[lastroot].rval;   A[lastroot].rval = INF;
      heapify_lheap( lastroot );
      return retdata;
   }

   // A[lastroot].rval is INF.
   leftchild = 2 * lastroot;
   if( leftchild > MaxLeaf ) { // lastroot is a leaf node.
      lastLeaf--;
      tmp = lastp;  lastp = lastp.nextp;  delete tmp;
   } else {   // lastroot has children
      split_heap( );
   }
   return retdata;
}
```

(그림 14) 최소값 삭제 알고리즘

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 정보통신공학과 교수

관심분야 : 알고리즘, 계산기하, 웹 데이터베이스, 웹 보안