11주차 3차시 히프

【학습목표】

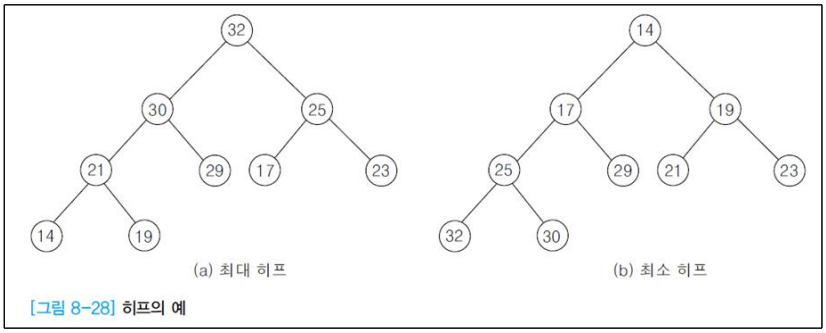
1. 히프를 예를 들어 설명할 수 있다.
2. 최대 히프와 최소 히프를 구분할 수 있다.

학습내용1 : 히프와 히프의 추상자료형

1. 히프(Heap)

* 히프란?
- 완전 이진 트리에 있는 노드 중에서 킷값이 가장 큰 노드나 킷값이 가장 작은 노드를 찾기 위해서 만든 자료구조
- 최대 히프
    - 킷값이 가장 큰 노드를 찾기 위한 완전 이진 트리
    - {부모노드의 킷값 ≥ 자식노드의 킷값}
    - 루트 노드 : 킷값이 가장 큰 노드
- 최소 히프(min heap)
    - 킷값이 가장 작은 노드를 찾기 위한 완전 이진 트리
    - {부모노드의 킷값 ≤ 자식노드의 킷값}
    - 루트 노드 : 킷값이 가장 작은 노드

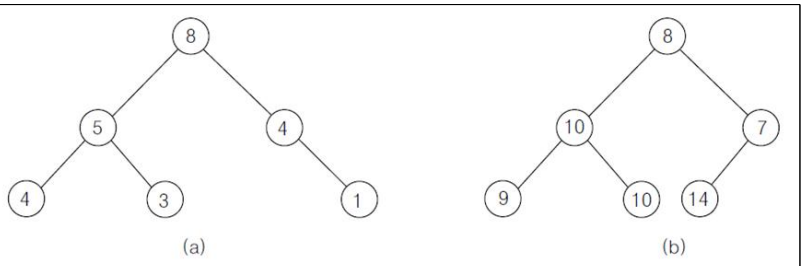
* 히프의 예



[그림 8-28] 히프의 예

* 히프가 아닌 이진 트리의 예

2. 히프의 추상 자료형



[그림 8-29] 히프가 아닌 이진 트리의 예

```
ADT Heap
  데이터 : n개의 원소로 구성된  완전 이진 트리로서 각 노드의 킷값은
          그의 자식 노드의 킷값보다 크거나 같다.
          (부모 노드의 킷값 ≥ 자식 노드의 킷값)
  연산 :
      heap∈Heap; item∈Element;
      createHeap() ::= create an empty heap;
            // 공백 히프의 생성 연산

      isEmpty(heap) ::= if (heap is empty) then  return  true;
                        else return false;
            // 히프가 공백인지를 검사하는 연산

      insertHeap(heap, item) ::= insert item into heap;
            // 히프의 적당한 위치에 원소(item)를 삽입하는 연산

      deleteHeap(heap) ::= if (isEmpty(heap)) then return error;
                           else {
                                 item ← 히프에서  가장 큰 원소;
                                 remove {히프에서 가장 큰 원소};
                                 return  item;
                           }
            // 히프에서 킷값이 가장 큰 원소를 삭제하고 반환하는 연산
End Heap()
```

학습내용2 : 히프의 삽입, 삭제 연산

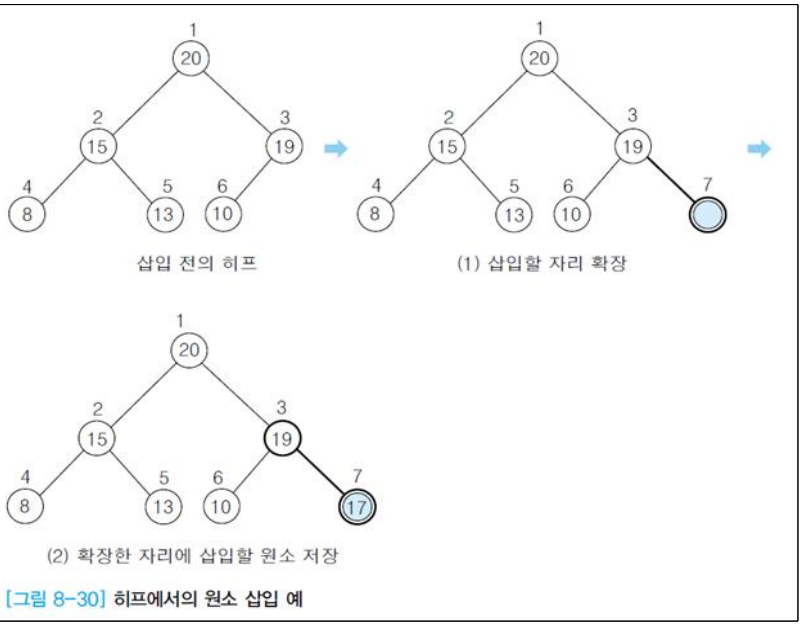
1. 히프에서의 삽입 연산

* 1단계 : 완전 이진 트리를 유지하면서 확장한 노드에 삽입할 원소를 임시 저장
- 노드가 n개인 완전 이진 트리에서 다음 노드의 확장 자리는 n+1번의 노드가 된다.
- n+1번 자리에 노드를 확장하고, 그 자리에 삽입할 원소를 임시 저장한다.

* 2단계 : 만들어진 완전 이진 트리 내에서 삽입 원소의 제자리를 찾는다.
- 현재 위치에서 부모노드와 비교하여 크기 관계를 확인한다.
- {현재 부모노드의 킷값 ≥ 삽입 원소의 킷값}의 관계가 성립하지 않으면, 현재 부모노드의 원소와 삽입 원소의 자리를 서로 바꾼다.

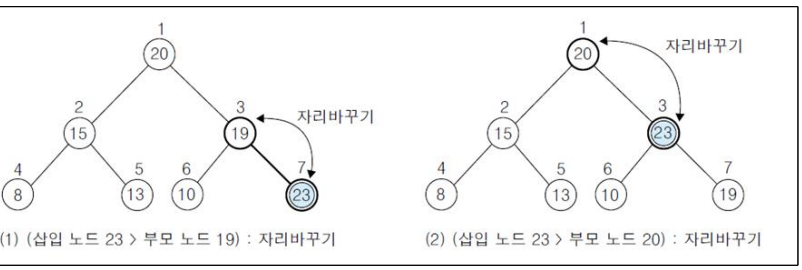
2. 히프 삽입 연산 예제

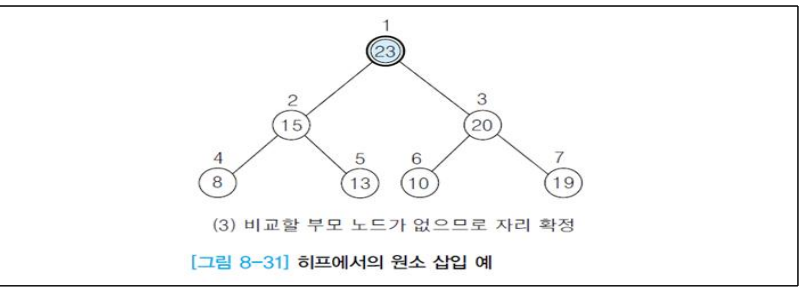
* 예제1 : 17을 삽입하는 경우
- 노드를 확장하여 임시로 저장한 위치에서의 부모 노드와 크기를 비교하여 히프의 크기 관계가 성립하므로 현재 위치를 삽입 원소위 자리로 확정



[그림 8-30] 히프에서의 원소 삽입 예

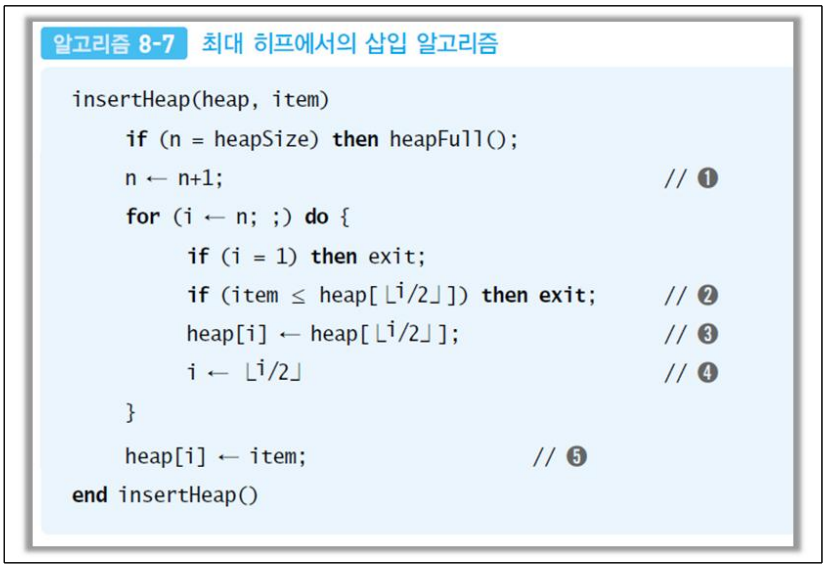
* 예제2 : 23을 삽입하는 경우





[그림 8-31] 히프에서의 원소 삽입 예

3. 히프에서의 삽입 연산 알고리즘



알고리즘 8-7 최대 히프에서의 삽입 알고리즘

```
insertHeap(heap, item)
    if (n = heapSize) then heapFull();
    n ← n+1;                                          // ❶
    for (i ← n; ;) do {
        if (i = 1) then exit;
        if (item ≤ heap[⌊i/2⌋]) then exit;            // ❷
        heap[i] ← heap[⌊i/2⌋];                        // ❸
        i ← ⌊i/2⌋                                     // ❹
    }
    heap[i] ← item;                                   // ❺
end insertHeap()
```

① 현재 히프의 크기를 하나 증가시켜서 노드 위치를 확장하고, 확장한 노드 번호가 현재의 삽입 위치 i가 된다.
② 삽입할 원소 item과 부모 노드 heap[⌊i/2⌋]를 비교하여 item이 부모 노드 보다 작거나 같으면 현재의 삽입 위치 i를 삽입 원소의 위치로 확정한다.
③ 만약 삽입할 원소 item이 부모 노드보다 크면, 부모 노드와 자식 노드의 자리를 바꾸어 최대 히프의 관계를 만들어야 하므로 부모 노드 heap[⌊i/2⌋]를 현재의 삽입 위치 heap[i]에 저장하고,
④ i/2를 삽입 위치 i로 하여, ②~④를 반복하면서 item을 삽입할 위치를 찾는다.
⑤ 찾은 위치에 삽입할 노드 item을 저장하면, 최대 히프의 재구성 작업이 완성되므로 삽입 연산을 종료한다.

4. 히프에서의 삭제 연산

* 히프에서는 루트 노드의 원소만을 삭제할 수 있다
- 1단계 : 루트 노드의 원소를 삭제하여 반환한다.
- 2단계 : 원소의 개수가 n-1개로 줄었으므로, 노드의 수가 n-1인 완전 이진 트리로 조정한다.
    - 노드가 n개인 완전 이진 트리에서 노드 수 n-1개의 완전 이진 트리가 되기 위해서 마지막 노드, 즉 n번 노드를 삭제한다.
    - 삭제된 n번 노드에 있던 원소는 비어있는 루트노드에 임시 저장한다.
- 3단계 : 완전 이진 트리 내에서 루트에 임시 저장된 원소의 제자리를 찾는다.
    - 현재 위치에서 자식노드와 비교하여 크기 관계를 확인한다.
    - {임시 저장 원소의 킷값 ≥ 현재 자식노드의 킷값 }의 관계가 성립하지 않으면, 현재 자식노드의 원소와 임시 저장 원소의 자리를 서로 바꾼다

5. 히프에서의 삭제 연산 예제

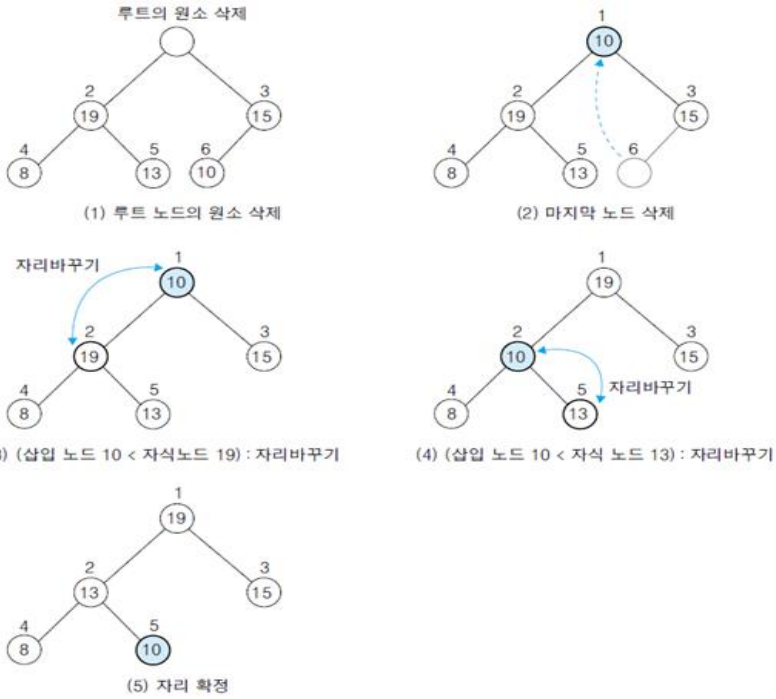


그림 8-32] 히프에서의 원소 삭제 예

6. 히프에서의 삭제 연산 알고리즘

알고리즘 8-8 최대 히프에서의 삭제 알고리즘

```
deleteHeap(heap)
    if (n = 0) then return error;
    item ← heap[1];                                  // ❶
    temp ← heap[n];                                  // ❷
    n ← n-1;                                         // ❸
    i ← 1;                                           // ❹
    j ← 2;
    while (j ≤ n) do {
        if (j < n) then
            if (heap[j] < heap[j+1]) then j ← j+1;   // ❺
        if (temp ≥ heap[j]) then exit;
        heap[i] ← heap[j];
        i ← j;                                       // ❻
        j ← j*2;
    }
    heap[i] ← temp;                                  // ❼
    return item;                                     // ❽
end deleteHeap()
```

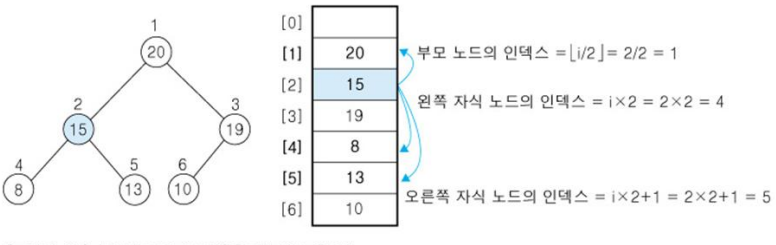
① 루트노드 heap[1]을 변수 item에 저장하고,
② 마지막 노드의 원소 heap[n]을 변수temp에 임시 저장한 후에,
③ 마지막 노드를 삭제하고 히프배열의 원소 개수를 하나 감소한다.
④ 마지막 노드의 원소였던 temp의 임시 저장위치 i는 루트노드의 자리인 1번이 된다.
⑤ 현재 저장위치에서 왼쪽 자식 노드 heap[j]와 오른쪽 자식 노드 heap[j+1]이 있을 때, 둘 중에서 킷값이 큰 자식

노드의 킷값과 temp를 비교하여, temp가 크거나 같으면 현재 위치가 temp의 자리로 확정된다.
⑥ 만약 temp가 자식노드보다 작으면, 자식노드와 자리를 바꾸고 다시 ⑤~⑥을 반복하면서 temp의 자리를 찾는다.
⑦ 찾은 위치에 temp를 저장하여 최대 히프의 재구성 작업을 완성하고
⑧ 루트노드를 저장한 item을 반환하는 것으로 삭제 연산을 종료한다.

학습내용3 : 순차 자료구조를 이용한 히프의 구현

1. 히프의 구현

- 부모 노드와 자식 노드를 찾기 쉬운 1차원 배열의 순차 자료 구조 이용
- 1차원 배열을 이용한 히프의 표현 예



[그림 8-33] 순차 자료구조를 이용한 히프의 표현 예

2. 순차 자료구조를 이용한 C 프로그램

* 공백 히프에 원소 10, 45, 19, 11, 96을 차례로 삽입하면서 최대 히프를 구성하고, 삭제연산을 수행하여 삭제된 원소를 출력하는 프로그램
- 최대 히프의 루트 노드는 히프에서 가장 큰 노드가 되므로, 원소 개수 만큼 삭제 연산을 수행하면서 출력하면 큰 값부터 작은 값의 내림차순으로 출력되는 것을 확인할 수 있다

* [예제] 순차 자료구조를 이용한 히프 프로그램

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #define MAX_ELEMENT 100
04  typedef struct { // 히프에 대한 1차원 배열과 히프 원소의 갯수를 구조체로 묶어서 선언
05          int heap[MAX_ELEMENT];
06          int heap_size;
07  } heapType;
08
09  heapType* createHeap() // 공백 히프를 생성하는 연산
10  {
11          heapType *h = (heapType *)malloc(sizeof(heapType));
12          h->heap_size=0;
13          return h;
14  }
15
16  void insertHeap(heapType *h, int item) // 히프에item을 삽입하는 연산
17  {
18          int i;
19          h->heap_size = h->heap_size +1;
20          i = h->heap_size;
21          while((i!=1) && (item > h->heap[i/2])){
22             h->heap[i] = h->heap[i/2];
23             i/=2;
24          }
25          h->heap[i] = item;
26  }
```

```
28  int deleteHeap(heapType *h) // 히프의 루트를 삭제하여 반환하는 연산
29  {
30          int parent, child;
31          int item, temp;
32          item = h->heap[1];
33          temp = h->heap[h->heap_size];
34          h->heap_size = h->heap_size -1;
35          parent = 1;
36          child = 2;
37          while(child <= h->heap_size) {
38             if((child < h->heap_size) && (h->heap[child]) < h->heap[child+1])
39                child++;
40             if (temp >= h->heap[child]) break;
41             else {
42                h->heap[parent] = h->heap[child];
43                parent = child;
44                child = child*2;
45             }
46          }
47          h->heap[parent] = temp;
48          return item;
49   }
50
51  printHeap(heapType *h) // 1차원 배열 히프의 내용을 출력하는 연산
52  {
53          int i;
54          printf("Heap : ");
```

```
55          for(i=1; i<= h->heap_size; i++)
56          printf("[%d] ", h->heap[i]);
57  }
58
59  void main()
60  {
61          int i, n, item;
62          heapType *heap = createHeap();
63          insertHeap(heap, 10);
64          insertHeap(heap, 45);
65          insertHeap(heap, 19);
66          insertHeap(heap, 11);
67          insertHeap(heap, 96);
68
69          printHeap(heap);
70
71          n= heap->heap_size;
72          for(i=1; i<=n; i++){
73             item = deleteHeap(heap);
74             printf("₩n delete : [%d] ", item);
75          }
76
77          getchar();
78  }
```

* 실행 결과

【학습정리】

1. 히프는 완전 이진 트리에 있는 노드 중에서 킷값이 가장 큰 노드나 가장 작은 노드를 찾기 위해서 만든 자료구조이다.

2. 최대 히프는 {부모 노드의 키값≥ 자식 노드의 킷값} 의 관계를 가지는 노드들의 완전 이진 트리이고, 최소 히프는 {부토 노드의 킷값 ≤자식 노드의 킷값}의 관계를 가지는 노드들의 완전 이진 트리이다.

3. 일반적으로 히프는 최대 히프를 의미하여 같은 킷값의 노드가 중복되어 있을 수 있다.