

Data Structure

자료구조



힙 트리



한국기술교육대학교
온라인평생교육원

학습내용

- 힙 트리의 구조와 특징
- 힙의 구현

학습목표

- 힙 트리의 구조와 특징을 설명할 수 있다.
- 힙의 삽입 연산과 삭제 연산을 구현할 수 있다.



힙 트리의 구조와 특징

힙 트리

힙 트리의 구조와 특징

1 힙(Heap)이란?

힙(Heap)

- ‘더미’
- 가장 큰(또는 작은) 값을 빠르게 찾아내도록 만들어진 자료구조
 - 완전 이진 트리
- 우선순위 큐를 구현하는 가장 좋은 방법
 - 삽입/삭제 연산의 시간 복잡도가 $O(\log n)$
- 느슨한 정렬 상태만을 유지
 - 큰 값이 상위 레벨, 작은 값이 하위 레벨에 있음

힙 트리

힙 트리의 구조와 특징

1 힙(Heap)이란?

● 힙의 종류

최대 힙(Max Heap)

부모의 키 값이
자식 노드의
키 값보다 **크거나 같은**
완전 이진 트리

최소 힙(Min Heap)

부모의 키 값이
자식 노드의
키 값보다 **작거나 같은**
완전 이진 트리



힙 트리의 구조와 특징

힙 트리

힙 트리의 구조와 특징

1 힙(Heap)이란?

● 힙의 종류



힙 트리

힙 트리의 구조와 특징

2 힙의 연산

조건

- 연산들은 힙의 특성을 유지하도록 동작하여야 함
- 구조적인 특성 : 완전 이진 트리
- 순서 특성 : 최대 힙 또는 최소 힙

▶ 힙의 연산

삽입 연산	<ul style="list-style-type: none"> ▪ 새로운 항목을 힙에 추가하는 연산 ▪ 추가 후 힙의 특성을 반드시 유지
삭제 연산	<ul style="list-style-type: none"> ▪ 루트 노드를 꺼내 반환하는 연산 ▪ 마찬가지로 힙의 특성을 유지

힙 트리의 구조와 특징



힙 트리

힙 트리의 구조와 특징

2 힙의 연산

● 삽입 연산

아이디어

- 회사에서 신입 사원이 들어오면 일단 말단 위치에 앉힘
 - 완전 이진 트리 구조 유지
- 신입 사원의 능력을 봐서 위로 승진(Upheap, Shift-up)
 - 순서 특성을 맞춤



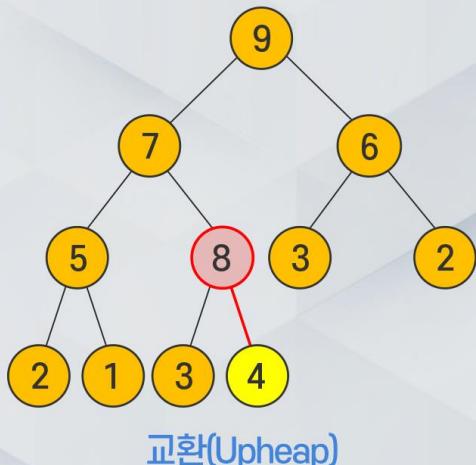
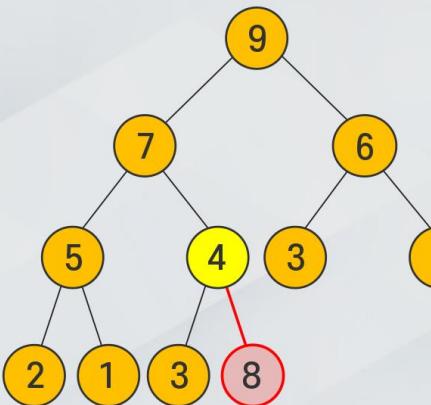
힙 트리

힙 트리의 구조와 특징

2 힙의 연산

● Upheap

► Upheap 과정 $O(\log n)$





힙 트리의 구조와 특징

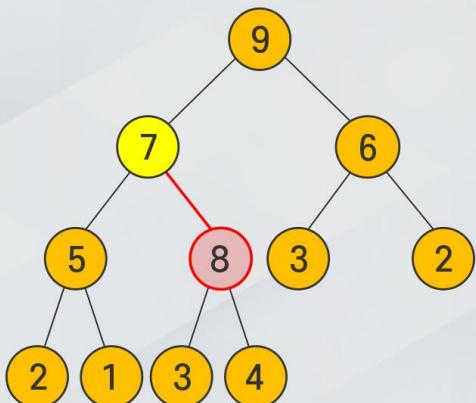
힙 트리

힙 트리의 구조와 특징

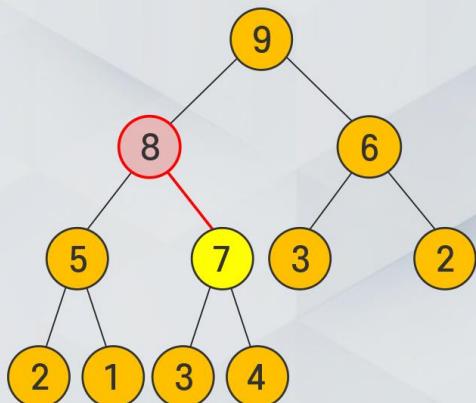
2 힙의 연산

◎ Upheap

► Upheap 과정 $O(\log n)$



부모 노드(4)와 비교



교환(Shift-up)

힙 트리

힙 트리의 구조와 특징

2 힙의 연산

◎ 삭제 연산

삭제 연산

- 힙에서는 루트 노드가 최대 또는 최소 우선 순위 노드
- 루트를 꺼냄 → 최대/최소 우선 순위 노드 삭제 → **우선 순위 큐**

삭제 연산 아이디어

- 루트 삭제
- 회사의 가장 말단 사원을 루트에 앉힘
 - 완전 이진 트리 구조 유지
- 능력을 보고 순차적으로 강등(downheap)
 - 순서 특성을 맞춤



힙 트리의 구조와 특징

힙 트리

힙 트리의 구조와 특징

2 힙의 연산

◎ downheap

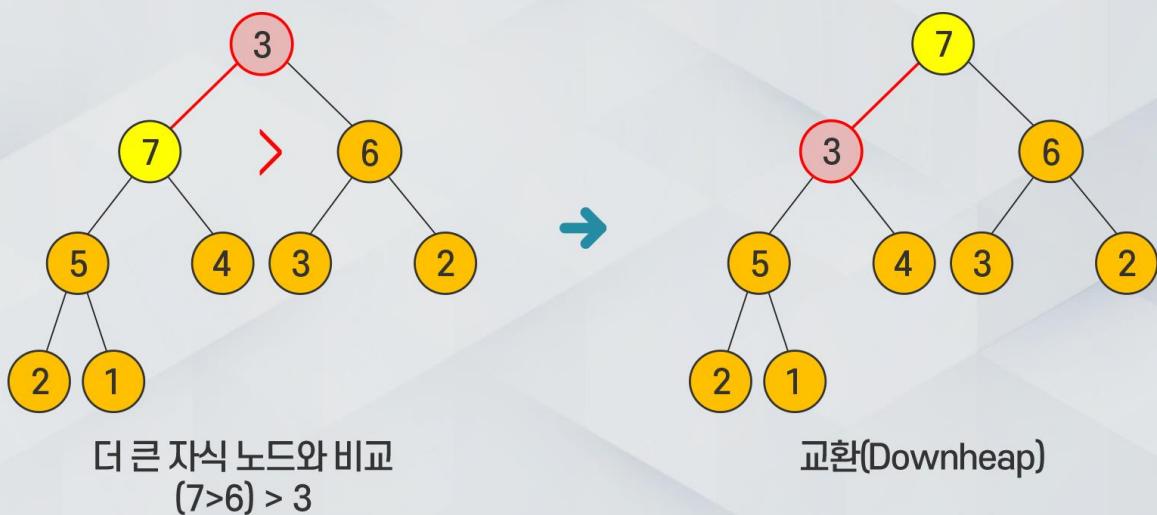


힙 트리

힙 트리의 구조와 특징

2 힙의 연산

◎ downheap





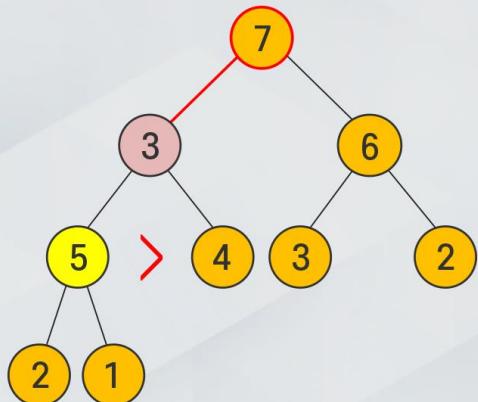
힙 트리의 구조와 특징

힙 트리

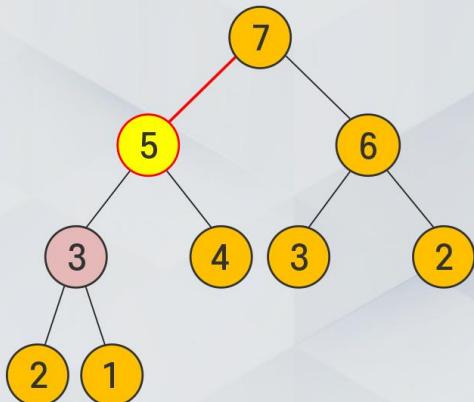
힙 트리의 구조와 특징

2 힙의 연산

- ◎ downheap



더 큰 자식 노드와 비교
(5>4) > 3



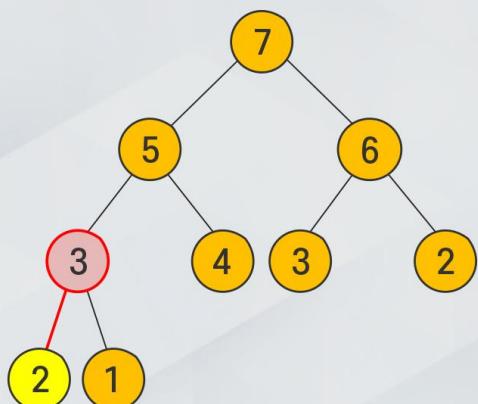
교환(Downheap)

힙 트리

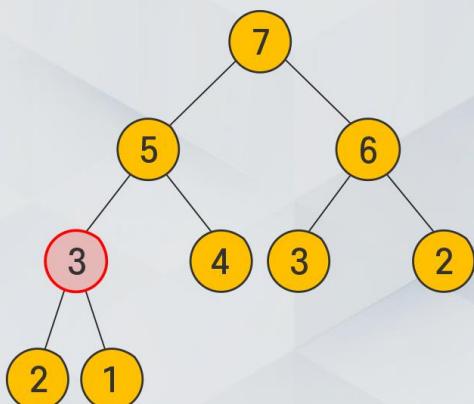
힙 트리의 구조와 특징

2 힙의 연산

- ◎ downheap



더 큰 자식 노드와 비교
(2>1) > 3



교환 없음. 종료



힙 트리의 구조와 특징

힙 트리

힙 트리의 구조와 특징

2 힙의 연산

◎ downheap

downheap의 복잡도

- 최악의 경우 트리의 높이 만큼의 강등이 발생
- 노드가 n 개인 완전 이진 트리의 높이 = $\log n$
- 시간 복잡도: $O(\log n)$





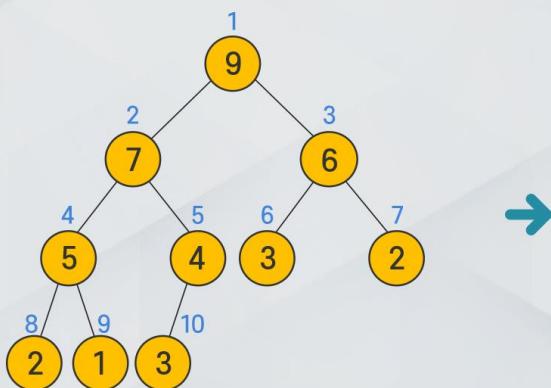
힙의 구현

힙 트리 힙의 구현

1 배열 구조의 힙 트리

힙은 보통 배열 구조로 구현

- 완전 이진 트리 → 각 노드에 번호를 붙임 → 배열의 인덱스
- 배열 중간에 빈 칸이 발생하지 않음



0	
1	9
2	7
3	6
4	5
5	4
6	3
7	2
8	2
9	1
10	3
11	
12	

인덱스 0은 사용하지 않음

완전 이진 트리이므로 중간에 빈 칸이 없음

힙 트리 힙의 구현

1 배열 구조의 힙 트리

- 부모 노드와 자식 노드의 관계

$$\text{왼쪽 자식의 인덱스} = (\text{부모의 인덱스}) * 2$$

$$\text{오른쪽 자식의 인덱스} = (\text{부모의 인덱스}) * 2 + 1$$

$$\text{부모의 인덱스} = (\text{자식의 인덱스}) / 2$$



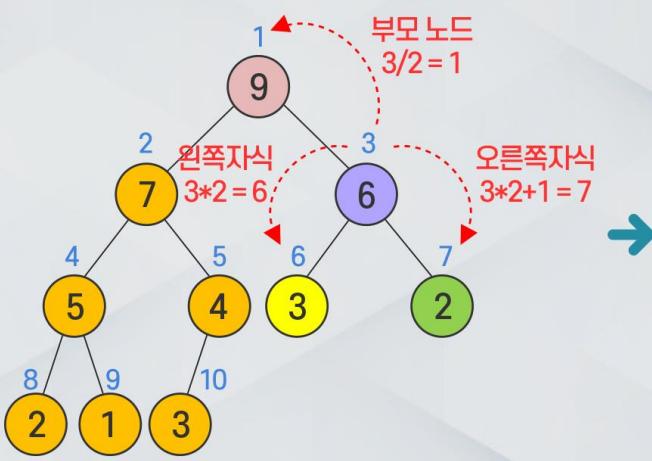


힙의 구현

힙 트리 힙의 구현

1 배열 구조의 힙 트리

- 부모 노드와 자식 노드의 관계



0	
1	9
2	7
3	6
4	5
5	4
6	3
7	2
8	2
9	1
10	3
11	
12	

힙 트리 힙의 구현

2 최대 힙의 구현

- 최대 힙 클래스(Max Heap)

- ▶ 고정된 크기의 배열 이용
- ▶ 힙에 저장될 레코드 자체가 우선 순위 값이라 가정

```
class MaxHeap: # 최대힙 클래스
    def __init__( self, capacity = 10 ): # 생성자 정의
        self.capacity = capacity # 힙의 용량
        self.heap = [None] * (capacity+1) # 배열구조의 힙
        self.hsize= 0 # 힙의 현재 원소 개수
```



힙의 구현

힙 트리

힙의 구현

2 최대 힙의 구현

● 기본 연산들

▶ 힙의 크기

```
def size(self) : # 힙의 크기
    return self.hsize
```

▶ 공백 상태 검사

```
def isEmpty(self) : # 공백상태 검사
    return self.size() == 0
```

▶ 포화 상태 검사

```
def isFull(self) : # 포화 상태 검사
    return self.size() == self.capacity
```

힙 트리

힙의 구현

2 최대 힙의 구현

● 노드 접근

▶ 부모, 자식 노드 접근

```
def Parent(self, i) : # 부모 노드 반환
    return self.heap[i//2]

def Left(self, i) : # 왼쪽 자식 반환
    return self.heap[i*2]

def Right(self, i) : # 오른쪽 자식 반환
    return self.heap[i*2+1]
```



힙의 구현

힙 트리

힙의 구현

2 최대 힙의 구현

● 삽입 연산

```

def insert(self, n) :
    self.hsize += 1
    self.heap[self.size()] = n
    i = self.size()
    while (i != 1 and n > self.Parent(i)):
        self.heap[i] = self.Parent(i)
        i = i // 2
    self.heap[i] = n

```

노드 개수 증가
 # 맨 마지막 노드로 일단 삽입
 # 노드 n의 위치
 # 부모보다 큰 동안 계속 업힙
 # 부모를 끌어내림
 # i를 부모의 인덱스로 올림
 # 마지막 위치에 n 삽입

힙 트리

힙의 구현

2 최대 힙의 구현

● 삭제 연산

```

def delete(self) :
    parent = 1
    child = 2
    if not self.isEmpty() :
        hroot = self.heap[1]
        last = self.heap[self.size()]
        while (child <= self.size()):
            # 만약 오른쪽 노드가 더 크면 child를 1 증가(기본은 왼쪽 노드)
            if child < self.size() and self.Left(parent) < self.Right(parent):
                child += 1
            if last >= self.heap[child] :
                # 자식이 더 작으면: 멈춤
                break
            # 아니면 down-heap 계속 진행

```

루트의 인덱스
 # 루트의 왼쪽 자식 인덱스
 # 힙이 공백이 아니면 삭제 가능
 # 삭제할 루트를 복사해 둠
 # 마지막 노드
 # 마지막 노드 이전까지 아래로 반복
 # 만약 오른쪽 노드가 더 크면 child를 1 증가(기본은 왼쪽 노드)
 # 자식이 더 작으면: 멈춤
 # down-heap 종료



힙의 구현

힙 트리 힙의 구현

2 최대 힙의 구현

● 삭제 연산

```

self.heap[parent] = self.heap[child]
parent = child
child *= 2;                                     # child를 부모에 복사
                                                    # parent, child 갱신

self.heap[parent] = last                         # last를 parent위치에 복사
self.hsize -= 1                                 # last를 삭제
return hroot                                    # 저장해두었던 루트를 반환

```

힙 트리 힙의 구현

2 최대 힙의 구현

● 기타 연산들

▶ peek() 연산

- 루트 노드 반환(인덱스 1)

```

def peek(self) :
    return self.heap[1]                           # 루트를 반환

```

▶ 화면 출력 연산

```

def display(self, msg = 'Heap: ') :
    print(msg, self.heap[1:self.size()+1])      # 슬라이싱 이용

```

힙의 구현



파일(F) 편집(E) 보기(V) 프로젝트(P) 디버그(D) 테스트(S) 놀이(N) 도구(T) 확장(X) 찾(W) 도움말(H) 검색(Ctrl+Q) 교재-파이썬

Max-heap.py BinaryTree.py BinTreeRep.py MapChain.py

```

1 class MaxHeap:           # 최대힙 클래스
2     def __init__(self, capacity = 10):    # 생성자 정의
3         self.capacity = capacity          # 힙의 용량
4         self.heap = [None] * (capacity + 1) # 배열구조의 힙
5         self.hsize = 0                     # 힙의 현재 원소 개수
6
7     def size(self):                   # 힙의 크기
8         return self.hsize
9     def isEmpty(self):               # 공백상태 검사
10        return self.size() == 0
11    def isFull(self):                # 포화상태 검사
12        return self.size() == self.capacity
13
14    def Parent(self, i):            # 부모노드 반환
15        return self.heap[i // 2]
16    def Left(self, i):              # 왼쪽 자식 반환
17        return self.heap[i * 2]
18    def Right(self, i):             # 오른쪽 자식 반환
19        return self.heap[i * 2 + 1]
20
21    def display(self, msg = 'Heap: '):
22        print(msg, self.heap[1:self.size() + 1])      # 슬라이싱 이용

```

206% 206% 206% 206%

출처 출처 보기 선택(S) 출처

오류 목록: 경고 0 출처

제작되었습니다.

실습 단계

최대 힙 클래스는 이와 같이 Maxheap이란 이름을 사용

생성자를 구현

끝날 때 반드시 루트는 반환

삽입을 할 때마다 현재 힙 트리를 화면에 출력

삭제를 한번 더 해 보고 현재 힙 상태를 출력 시도

Maxheap이 성공적으로 동작

기본적으로 최소 힙으로 동작

2를 삭제할 때마다 힙의 구조가 계속 바뀜

최종적으로 최소 힙을 출력하면