Heap

# Index	3
CreatedAt	@September 28, 2022
Person	A Ally Hyeseong Kim
Status	In Progress
Tags	Heap Java Python
UpdatedAt	@September 28, 2022

References

파이썬 알고리즘 인터뷰

2021 세종도서 학술부문 선정작. 현업과 실무에 유용한 주요 알고리즘 이론을 깊숙이 이해하고, 파이썬의 핵심 기능과 문법까지 상세하게 이해할 수 있는 취업용 코딩 테스트를 위한 완벽

https://www.aladin.co.kr/shop/wproduct.aspx?ItemId=245495826

References
1. Heap
2. Heap Operation
3. Binary Heap vs Binary Search Tree

1. Heap

> *`Heap`은 `Heap`의 특성(`Min Heap`에서는 부모가 항상 자식보다 크거나 같다)을 만족하는 Almost Complete Tree인 특수한 `Tree` 기반의 자료구조이다.*

- Python `heapq` module: `Min Heap`
  - 가장 작은 값: root of `heap`
- `Priority Queue`: `Heap`으로 구현
  - `Heap`은 주로 배열로 구현한다.

- `Heap`은 부모 node와 자식 node 간의 관계만 정의(좌우 node의 관계는 정의하지 않음)하므로 정렬된 구조가 아니다.
- `Binary Heap`: 자식이 둘인 `Heap`

---

2. Heap Operation

```
class BinaryHeap(object):
    def __init__(self):
        self.items = [None]

    def __len__(self):
        return len(self.items) - 1
```

2.1. Insertion

- 시간 복잡도: $O(logn)$

`Heap`에 element를 삽입하는 과정

1. element를 가장 하위 레벨의 최대한 왼쪽에 삽입한다.(배열에서는 가장 마지막)
2. `Up-Heap`: 계속 부모 값과 비교해서 더 작은 경우 위치를 변경한다.
   → 가장 작은 값은 root에 삽입된다.

```
def _percolate_up(self):
    i = len(self)
    parent = i / 2
    while parent > 0:
        if self.items[i] < self.items[parent]:
            self.items[parent], self.items[i] = self.items[i], self.items[parent]
        i = parent
        parent = i // 2

def insert(self, k):
    self.items.append(k)
    self._percolate_up()
```

2.2. Extraction

- 시간 복잡도: $O(logn)$

👩‍💻 `Heap`에서 element를 추출하는 과정

1. root에 있는 element를 추출한다.
2. `Down-Heap`: 계속 자식 값과 비교해서 더 큰 경우 위치를 변경한다.
   → `Heap`을 재정렬한다.

```
def _percolate_down(self, idx):
    left = idx * 2
    right = idx * 2 + 1
    smallest = idx

    if left <= len(self) and self.items[left] < self.items[smallest]:
        smallest = left

    if right <= len(self) and self.items[right] < self.items[smallest]:
        smallest = right

    if smallest != idx:
        self.items[idx], self.items[smallest] = self.items[smallest], self.items[idx]
        self._percolate_down(smallest)

def extract(self):
    extracted = self.items[1]
    self.items[1] = self.items[len(self)]
    self.items.pop()
    self._percolate_down(1)
    return extracted
```

---

3. Binary Heap vs Binary Search Tree

- `Binary Heap`: 상/하 관계를 보장한다.
  - `Min Heap`: 부모 < 자식
  - 가장 큰 값을 추출(`Max Heap`)하거나 가장 작은 값을 추출(`Min Heap`)할 때 사용한다.($O(1)$)
- `Binary Search Tree`: 좌/우 관계를 보장한다.
  - 왼쪽 자식 < 부모 ≤ 오른쪽 자식
  - 탐색, 삽입의 시간 복잡도: $O(logn)$
  - 모든 값이 정렬되어야 할 때 사용한다.