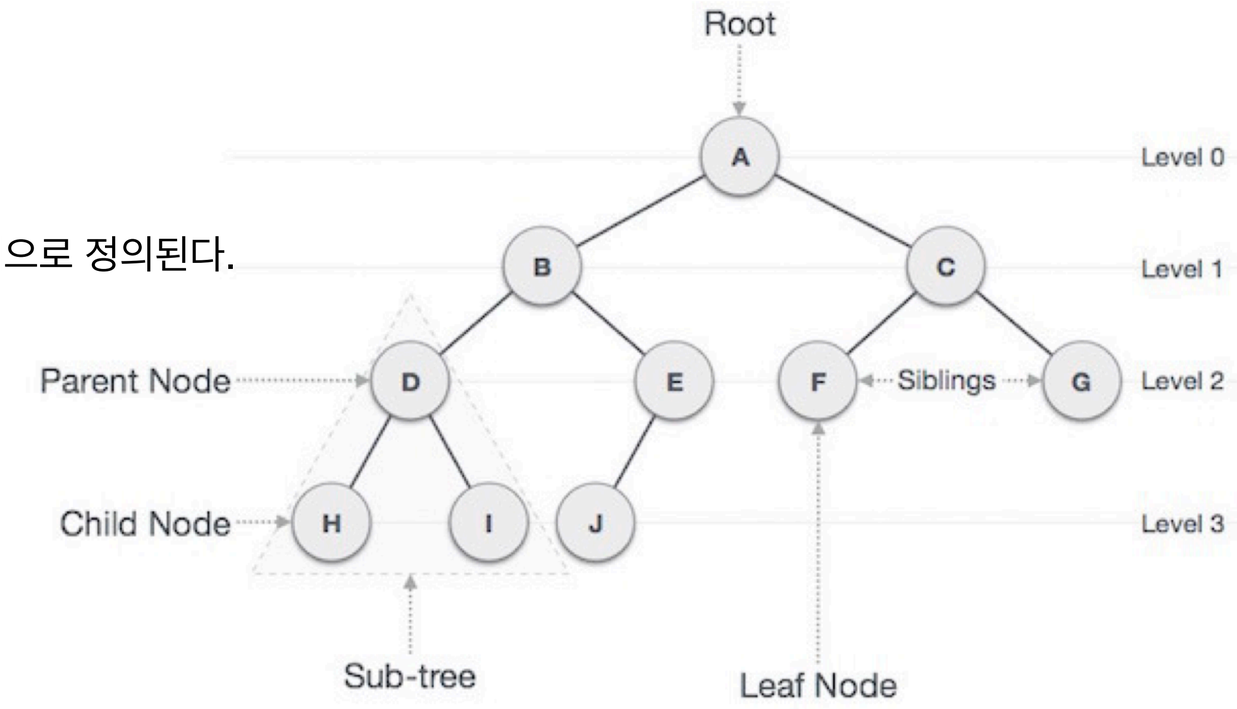
트리, 힙

알고리즘 스터디#5

2021.09.07

트리의 개념

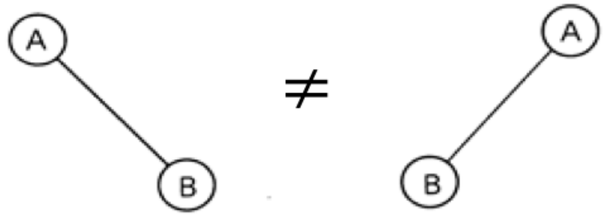
- 트리는 노드로 이루어진 자료 구조
  - 트리는 하나의 루트 노드를 갖는다.
  - 루트 노드는 0개 이상의 자식 노드를 갖고 있다.
  - 그 자식 노드 또한 0개 이상의 자식 노드를 갖고 있고, 이는 반복적으로 정의된다.
- 노드(node)들과 노드들을 연결하는 간선(edge)들로 구성되어 있다.
  - 트리에는 사이클(cycle)이 존재할 수 없다.
  - 노드들은 특정 순서로 나열될 수도 있고 그럴 수 없을 수도 있다.
  - 각 노드는 부모 노드로의 연결이 있을 수도 있고 없을 수도 있다.
  - 각 노드는 어떤 자료형으로도 표현 가능하다.

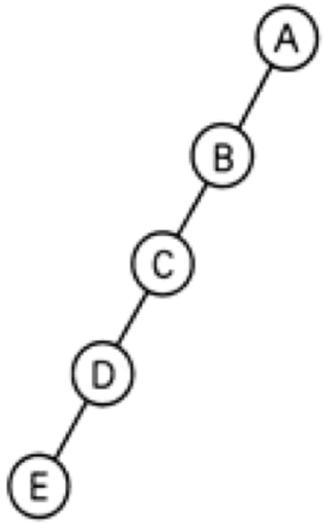


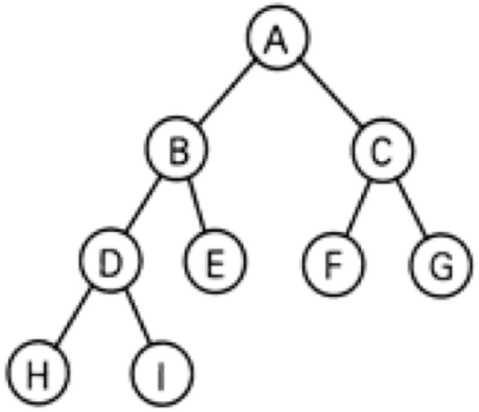
- 비선형 자료구조로 계층적 관계를 표현한다. ex) 디렉터리 구조, 조직도
- 사이클(cycle)이 없는 하나의 연결 그래프(Connected Graph) 또는 DAG(Directed Acyclic Graph, 방향성이 있는 비순환 그래프)의 한 종류이다.

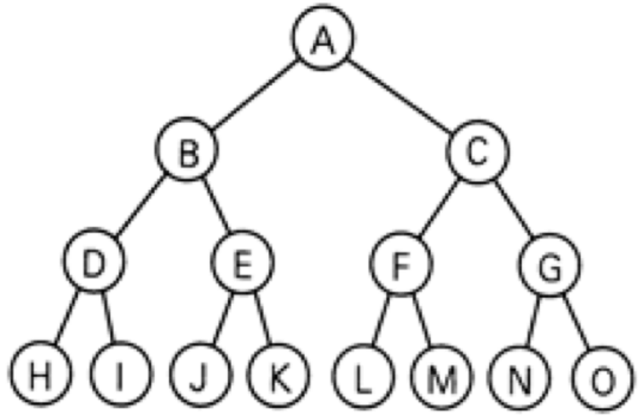
이진 트리(Binary Tree)

- 각 노드가 최대 두 개의 자식을 갖는 트리
- 모든 트리가 이진 트리는 아니다.
- 이진 트리에서는 왼쪽 부트리와 오른쪽 부트리를 구분함
- 이진 트리 종류
    - 경사 이진 트리
    - 완전 이진 트리
    - 포화 이진 트리
- 이진 트리 성질
    - 레벨 i에 있는 최대 노드 개수 2^i개
    - 깊이 k인 이진 트리의 최대 노드 개수 2^(k+1)-1개



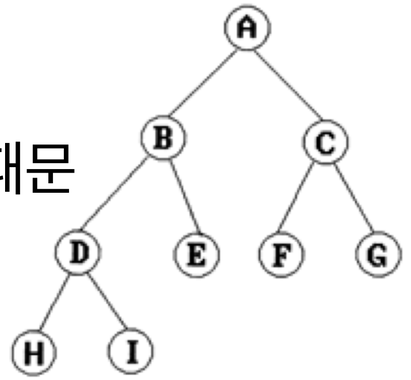


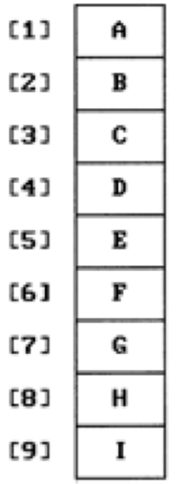


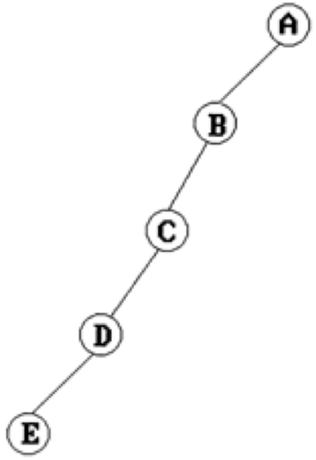


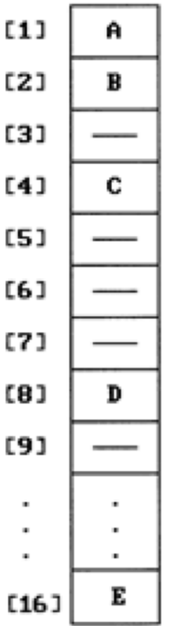
이진 트리 구현 방법1 : 인접 배열 이용

- 1차원 배열에 자신의 부모 노드만 저장하는 방법
  - 트리는 부모 노드를 0개 또는 1개를 가지기 때문
  - 부모 노드를 0개: 루트 노드





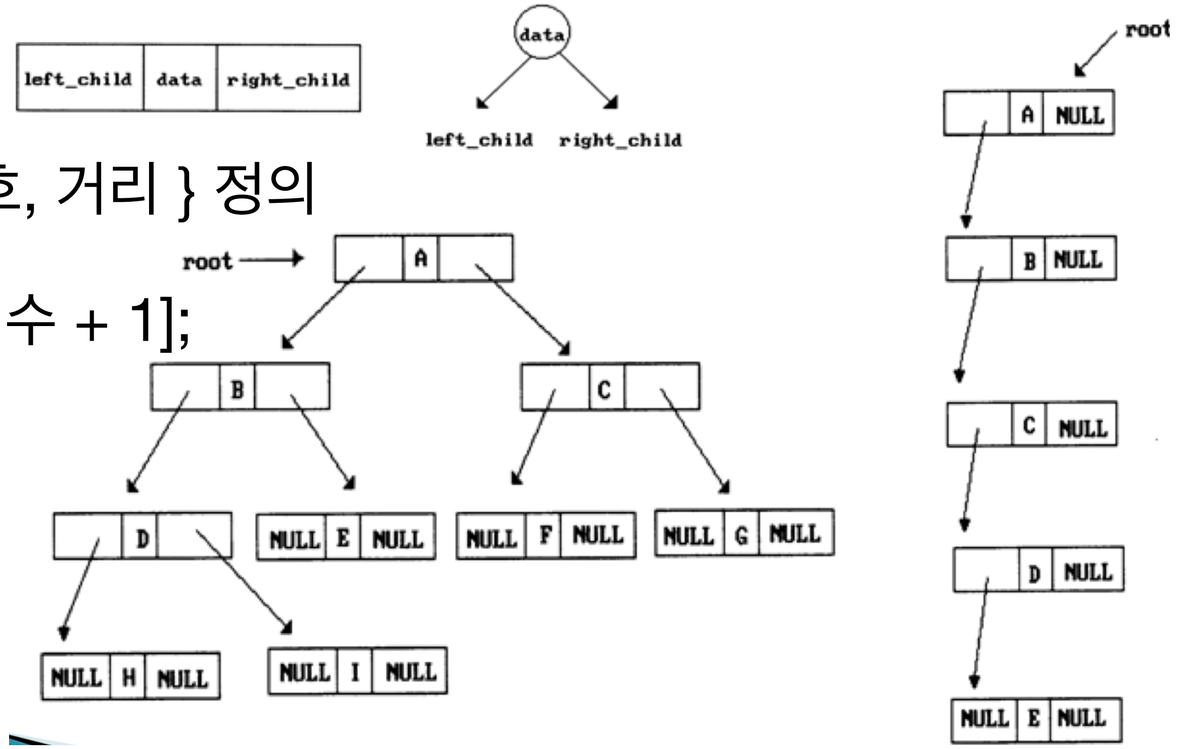




- 이진 트리의 경우, 2차원 배열에 자식 노드를 저장하는 방법
  - 이진 트리는 각 노드가 최대 두 개의 자식을 갖는 트리이기 때문
  - Ex) A[i][0]: 왼쪽 자식 노드, A[i][1]: 오른쪽 자식 노드
- 모든 이진 트리를 배열을 이용하여 표현할 수 있지만, 대부분 메모리 낭비 심함

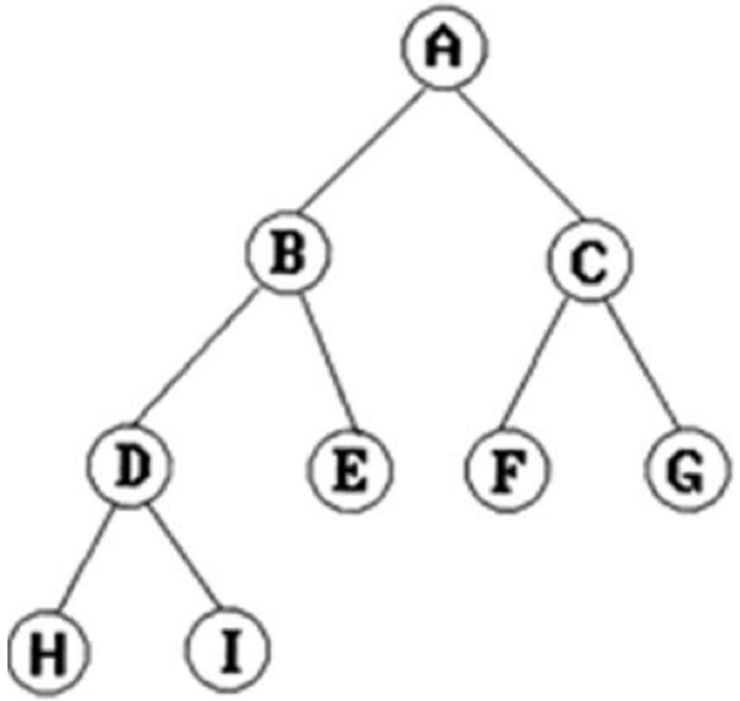
이진 트리 구현 방법2 : 인접 리스트 이용

- 가중치가 없는 트리의 경우
  - ArrayList< ArrayList > list = new ArrayList<>();
- 가중치가 있는 트리의 경우
  - 1) class Node { int num, dist; // 노드 번호, 거리 } 정의
  - 2) ArrayList[] list = new ArrayList[정점의 수 + 1];



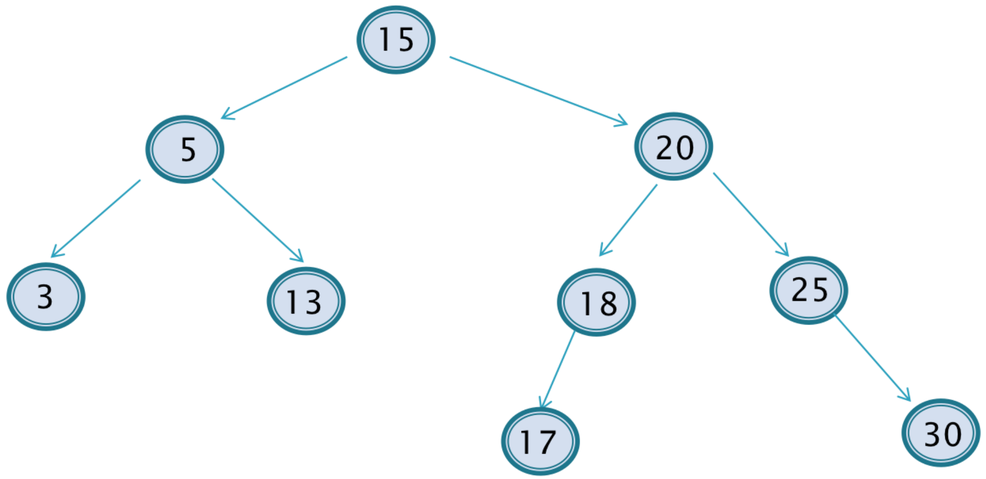
이진 트리 순회

- 트리의 모든 노드를 한번씩 방문하는 연산 - 중간 노드 기준
  - 중순위 순회(inorder traversal) -> 왼 - 중 - 오
    - H D I B E A F C G
  - 후순위 순회(postorder traversal) -> 왼 - 오 - 중
    - H I D E B F G C
  - 전순위 순회(preorder traversal) -> 중 - 왼 - 오
    - A B D H I E C F G
  - 레벨순위 순회(level order traversal)
    - A B C D E F G H I



이진 탐색 트리(Binary Search Tree)

- 특정 key값에 대해서 삽입, 삭제, 탐색이 가능한 자료구조 (사전(dictionary) 구조)
- 모든 노드 n에 대하여 왼쪽 자식 <= 부모 노드 < 오른쪽 자식이 성립
- 중위 순회(inorder traversal)를 하면, 오름차순으로 정렬된 순서로 Key값을 얻을 수 있음
- 평균 높이 O(logn)
- 시간 복잡도
    - 탐색 O(h)
    - 삽입 O(h)
    - 삭제 O(h)



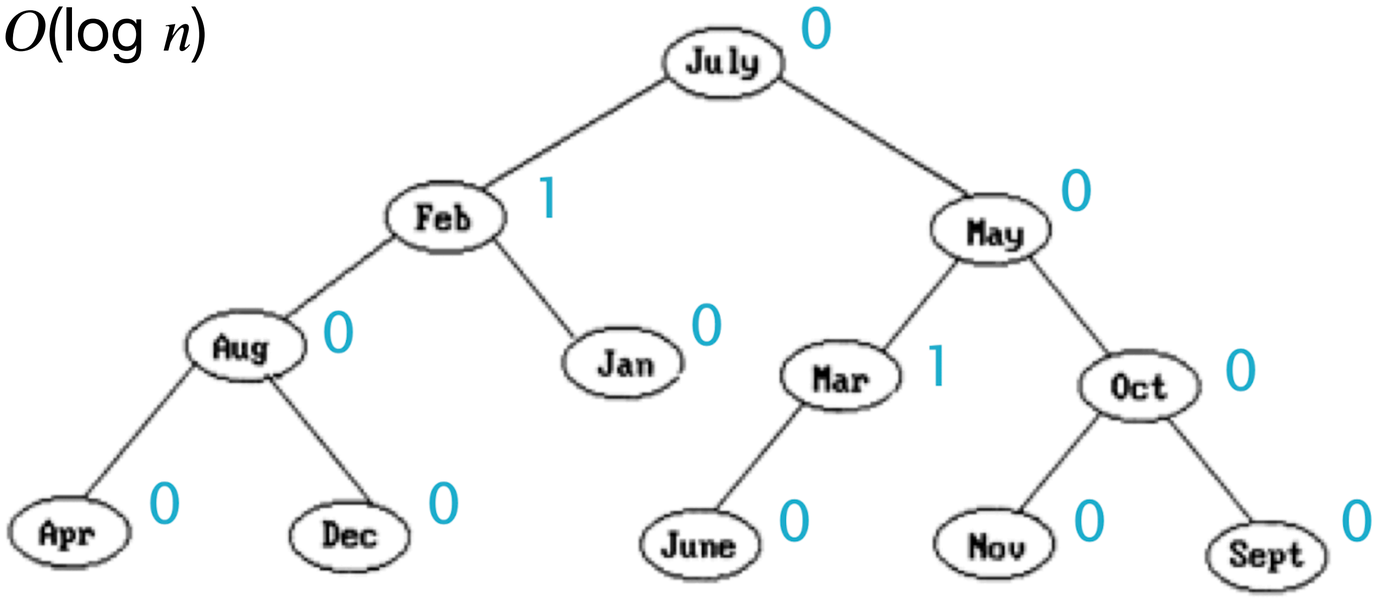
Search (18)

균형 이진 탐색 트리(Balanced BST)

- **균형 탐색 트리** : n 개 데이터 항목에 대한 동적 집합을 구현할 때높이 $O(\log n)$이 보장되는 탐색-트리 자료구조 (ex) AVL 트리, 레드-블랙 트리, 2-3 트리, 2-3-4 트리, B-트리)
- 이진 탐색 트리에서 탐색(Search), 삽입(Insert), 삭제(Delete) 등의 연산은 트리의 높이 h에 비례하는 시간 즉, $O(h)$ 시간이 소요된다.
- 이진 탐색 트리의 높이를 $O(\log n)$으로 제한할 수 있으면, 탐색, 삽입, 삭제 연산 모두 $O(\log n)$ 시간에 수행할 수 있다.
- 높이가 $O(\log n)$인 이진 탐색 트리를 균형 이진 탐색 트리라고 한다.
- ex) AVL 트리, 레드-블랙 트리

균형 이진 탐색 트리 1 : AVL 트리

- AVL 트리는 모든 노드에 대해서 왼쪽 부분트리와 오른쪽 부분트리의 높이 차가 1 이하인 이진 탐색 트리
- 왼쪽 부분트리와 오른쪽 부분트리의 높이를 h(l), h(r)이라고 할 때 h(l)–h(r)을 균형 인자(balance factor)라고 한다. AVL 트리에서 노드의 균형 인자는 -1, 0, 1이다.
- 높이, 탐색, 삭제, 삽입 $O(\log n)$
- 삽입 연산
  - LL, LR, RL, RR



균형 이진 탐색 트리 2: 레드-블랙 트리

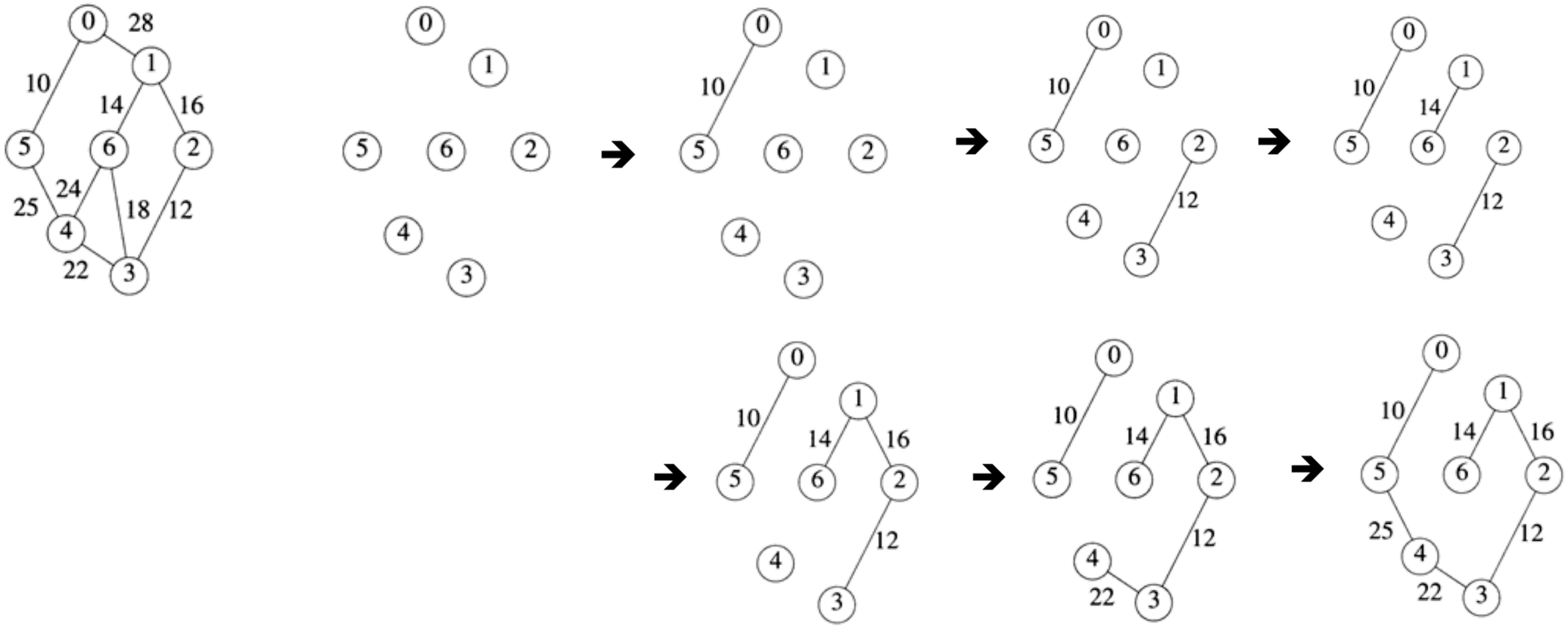
- 각 노드에 한 비트의 color 필드가 추가로 있으며 아래 레드-블랙 특성을 모두 만족하는 이진 탐색 트리
- 레드-블랙 특성
  - 1. 각 노드 color는 ‘레드’이거나 ‘블랙’이다.
  - 2. 루트와 외부 노드(NIL로 표현)의 color는 블랙이다.
  - 3. 레드 노드의 부모 노드는 반드시 블랙이다.
  - 4. 임의의 노드 x에서 후손 리프까지 가는 경로에 있는 블랙 노드의 개수는 같다. 이것을 black-height(x)라고한다.

최소 신장 트리

- 연결된 그래프 $G(V,E)$의 신장 트리란 G의 부분 그래프로서 G의 모든 정점을 포함하는 트리를 말한다. (여기서 트리는 사이클이 없고 연결된 그래프이다.)
- 에지에 가중치가 주어진 그래프 G에서 G의 신장 트리 중에서 에지 가중치 합이 최소가 되는 신장 트리를 최소 신장 트리라고 한다.
- 대표적인 최소 신장 트리 알고리즘
    - Kruskal의 알고리즘
    - Prim 알고리즘
    - => 둘 다 탐욕 방법 (greedy method)
- 비용이 최소인 통신 네트워크의 설계 등에 응용

Kruscal 알고리즘

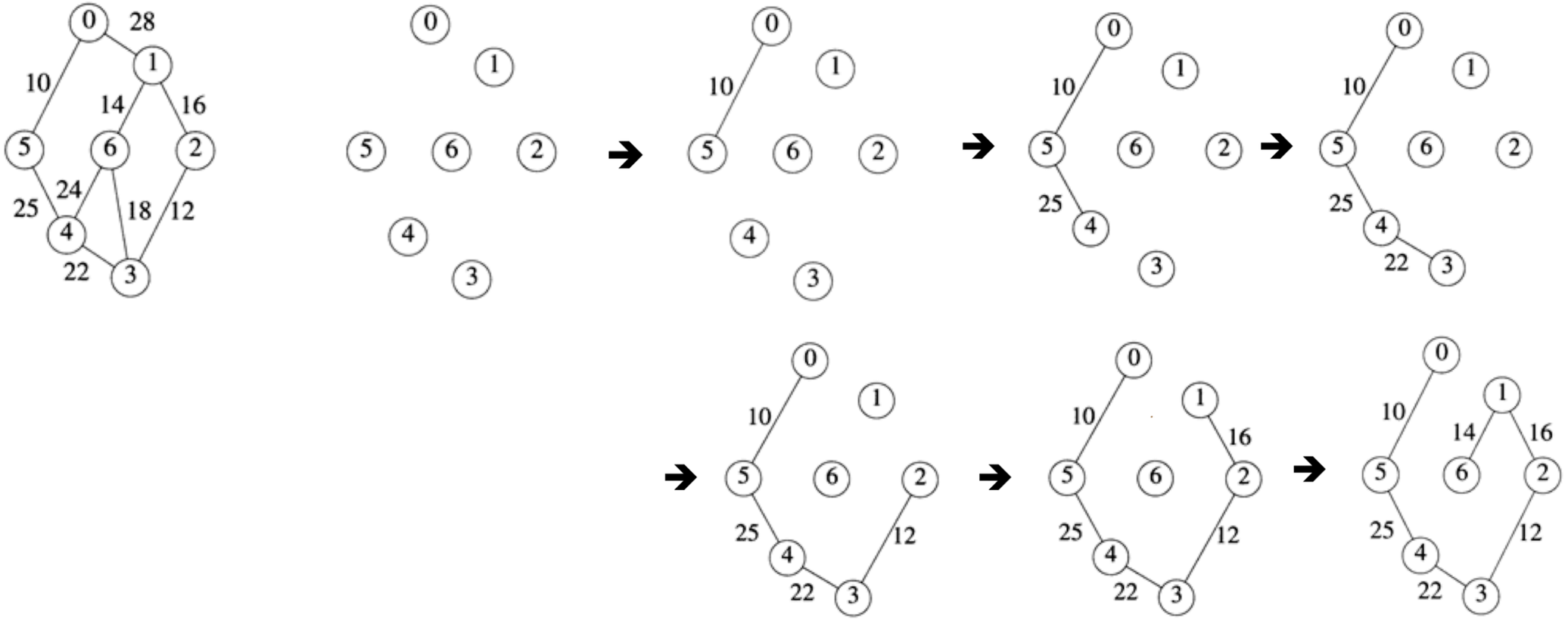
- 에지를 하나씩 추가하는 방식으로 최소 신장 트리를 만들어 감
- 에지의 가중치가 감소하지 않는 순서로 에지의 추가 여부를 결정하는데, 에지를 추가하였을 때 사이클을 형성하지 않으면 추가



Prim 알고리즘

- Prim 알고리즘은 그래프 G의 정점 하나로 이루어진 부분 트리에서 시작하여, 각 반복 과정마다 이미 구성된 부분 트리에 정점과 에지를 하나씩 추가함

- 이 때 부분 트리에 속한 정점과 인접한 정점 사이의 에지 중 가중치가 최소인 에지를 선택

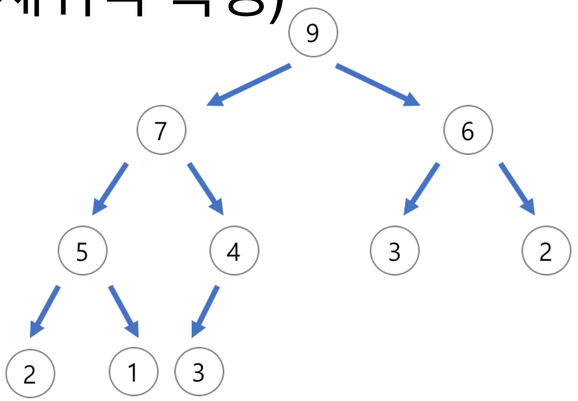


그래프와 트리의 차이

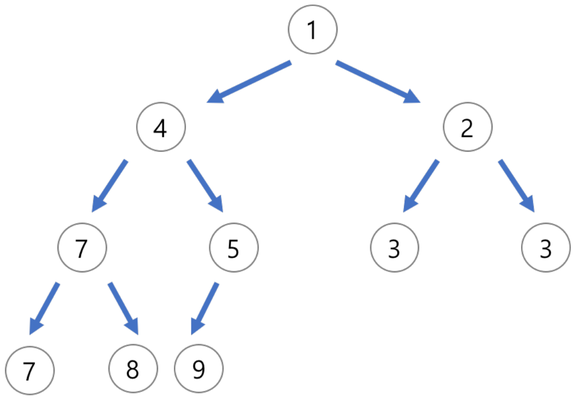
	그래프	트리
정의	노드(node)와 그 노드를 연결하는 간선(edge)을 하나로 모아 놓은 자료 구조	그래프의 한 종류 DAG(Directed Acyclic Graph, 방향성이 있는 비순환 그래프)의 한 종류
방향성	방향 그래프(Directed), 무방향 그래프(Undirected) 모두 존재	방향 그래프(Directed Graph)
사이클	사이클(Cycle) 가능, 자체 간선(self-loop)도 가능, 순환 그래프(Cyclic), 비순환 그래프(Acyclic) 모두 존재	사이클(Cycle) 불가능, 자체 간선(self-loop)도 불가능, 비순환 그래프(Acyclic Graph)
루트 노드	루트 노드의 개념이 없음	한 개의 루트 노드만이 존재, 모든 자식 노드는 한 개의 부모 노드 만을 가짐
부모-자식	부모-자식의 개념이 없음	부모-자식 관계 top-bottom 또는 bottom-top으로 이루어짐
모델	네트워크 모델	계층 모델
순회	DFS, BFS	DFS, BFS안의 Pre-, In-, Post-order
간선의 수	그래프에 따라 간선의 수가 다름, 간선이 없을 수도 있음	노드가 N인 트리는 항상 N-1의 간선을 가짐
경로	-	임의의 두 노드 간의 경로는 유일
예시 및 종류	지도, 지하철 노선도의 최단 경로, 전기 회로의 소자들, 도로(교차점과 일방 통행길), 선수 과목	이진 트리, 이진 탐색 트리, 균형 트리(AVL 트리, red-black 트리), 이진 힙(최대힙, 최소힙) 등

힙

- 우선순위 큐를 위하여 만들어진 자료구조
- 여러 개의 값들 중에서 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조
- 힙은 각 노드에 저장되어 있는 값이 자식 노드에 저장되어 있는 값보다 크거나(작거나) 같은 완전 이진 트리
- 힙 트리에서는 중복된 값을 허용한다. (이진 탐색 트리에서는 중복된 값을 허용하지 않는다.)
- 힙의 왼쪽 부트리, 오른쪽 부트리는 모두 힙이다. (재귀적 특성)
- N개가 힙에 들어가 있으면 높이는 log(N)이다.
- 최대 힙(max heap) / 최소 힙(min heap)



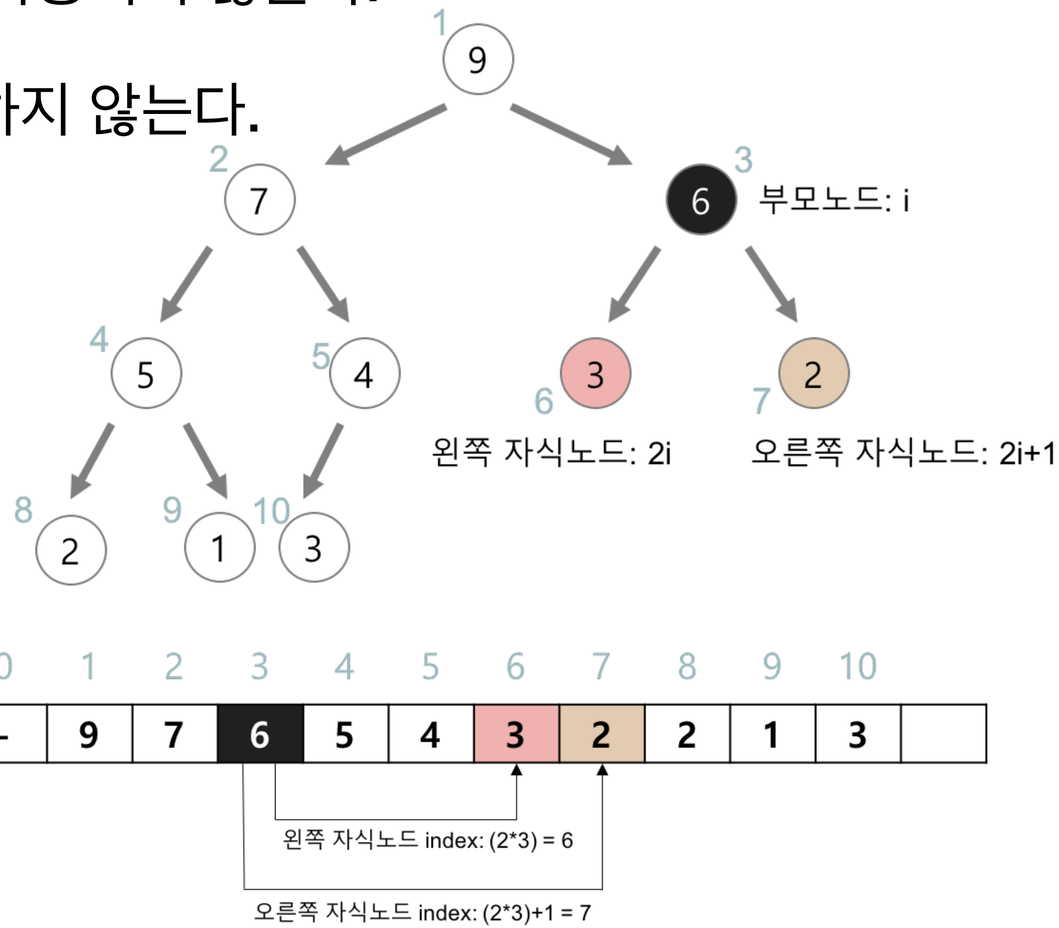
-최대 힙(max heap)-



-최소 힙(min heap)-

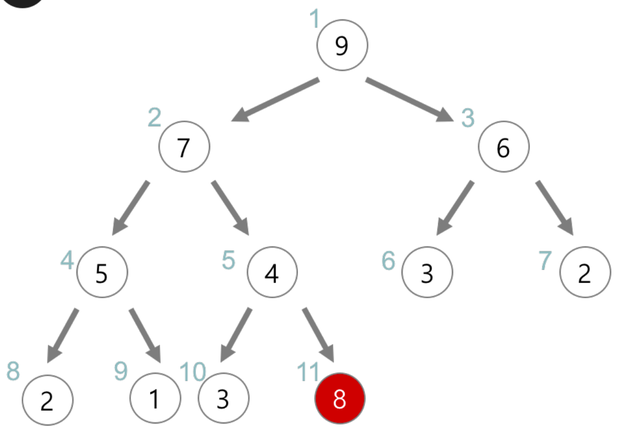
힙의 구현

- 힙을 저장하는 표준적인 자료구조는 배열이다.
- 구현을 쉽게 하기 위하여 배열의 첫 번째 인덱스인 0은 사용되지 않는다.
- 특정 위치의 노드 번호는 새로운 노드가 추가되어도 변하지 않는다.
- 힙에서의 부모 노드와 자식 노드의 관계
  - 왼쪽 자식의 인덱스 = (부모의 인덱스) * 2
  - 오른쪽 자식의 인덱스 = (부모의 인덱스) * 2 + 1
  - 부모의 인덱스 = (자식의 인덱스) / 2

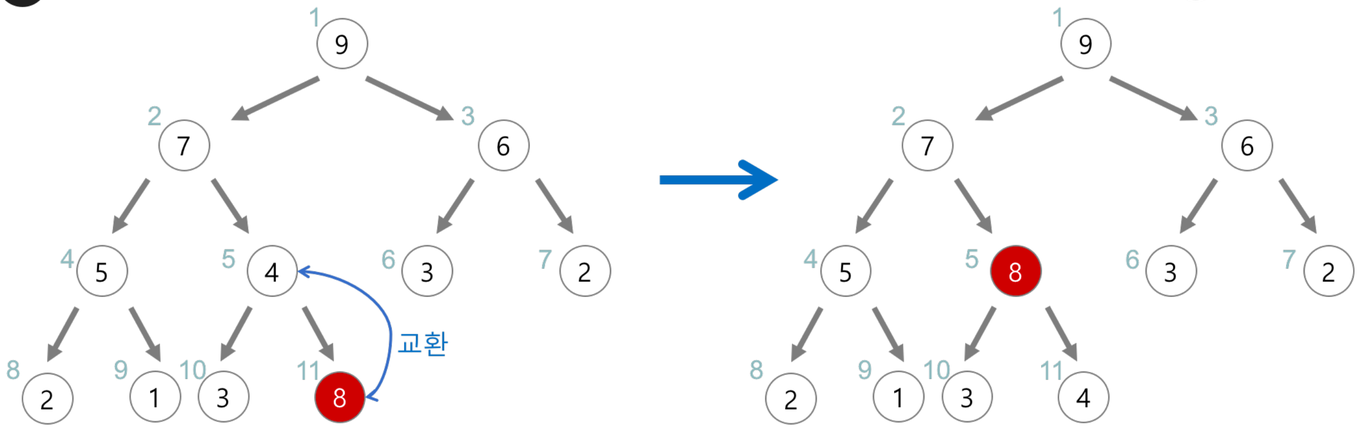


삽입 연산

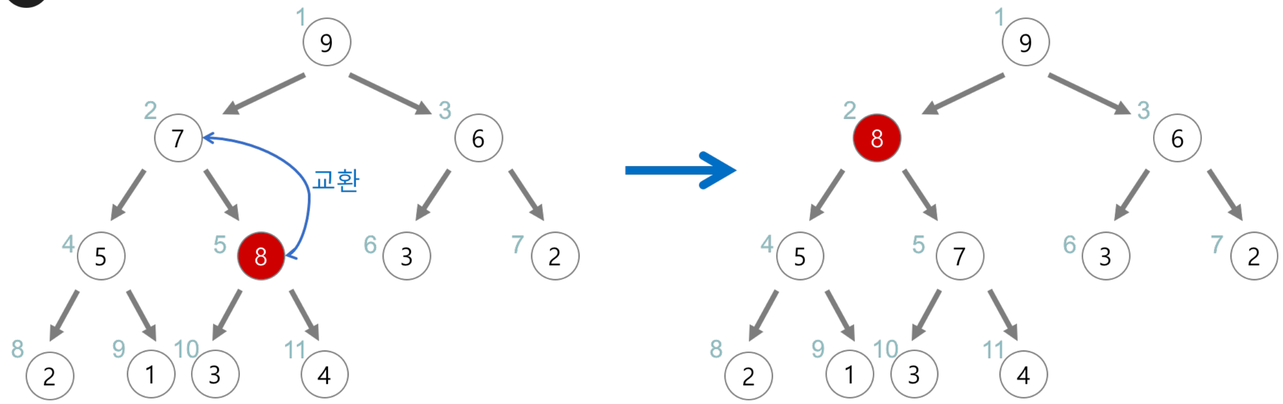
1 인덱스순으로 가장 마지막 위치에 이어서 새로운 요소 8을 삽입



2 부모 노드 4 < 삽입 노드 8 이므로 서로 교환



3 부모 노드 7 < 삽입 노드 8 이므로 서로 교환



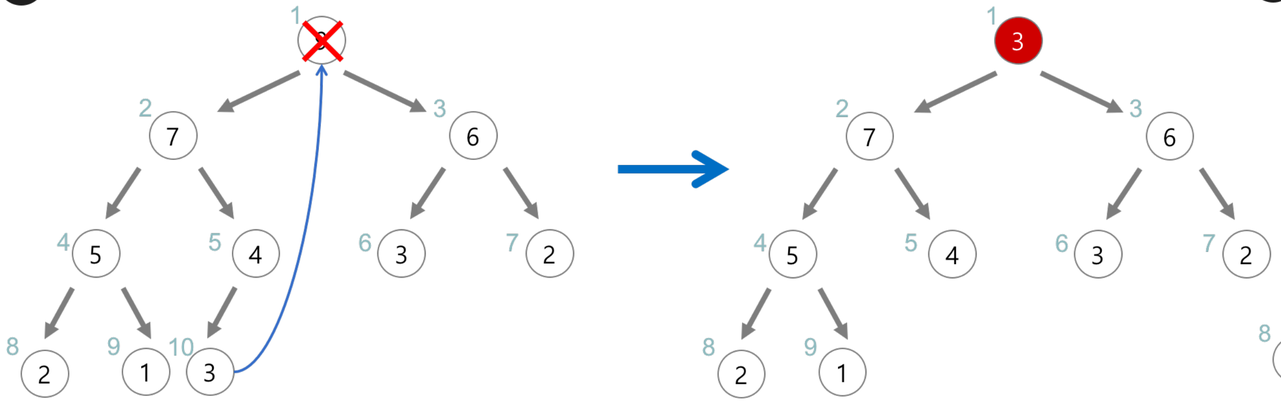
4 부모 노드 9 > 삽입 노드 8 이므로 더 이상 교환하지 않는다.

삽입 연산

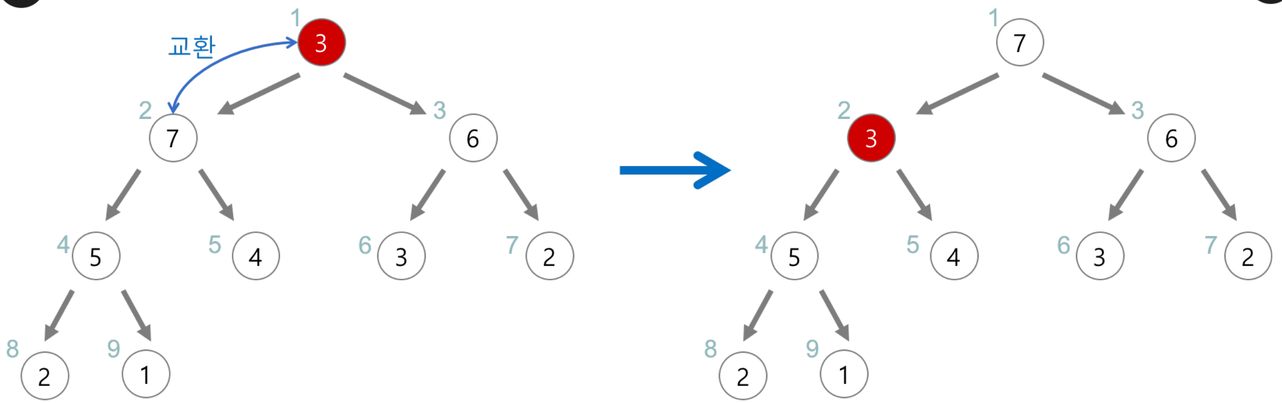
```c
/* 현재 요소의 개수가 heap_size인 힙 h에 item을 삽입한다. */
// 최대 힙(max heap) 삽입 함수
void insert_max_heap(HeapType *h, element item){
  int i;
  i = ++(h->heap_size); // 힙 크기를 하나 증가

  /* 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정 */
  // i가 루트 노트(index: 1)이 아니고, 삽입할 item의 값이 i의 부모 노드(index: i/2)보
  while((i != 1) && (item.key > h->heap[i/2].key)){
    // i번째 노드와 부모 노드를 교환환다.
    h->heap[i] = h->heap[i/2];
    // 한 레벨 위로 올라단다.
    i /= 2;
  }
  h->heap[i] = item; // 새로운 노드를 삽입
}
```

최댓값 삭제 연산

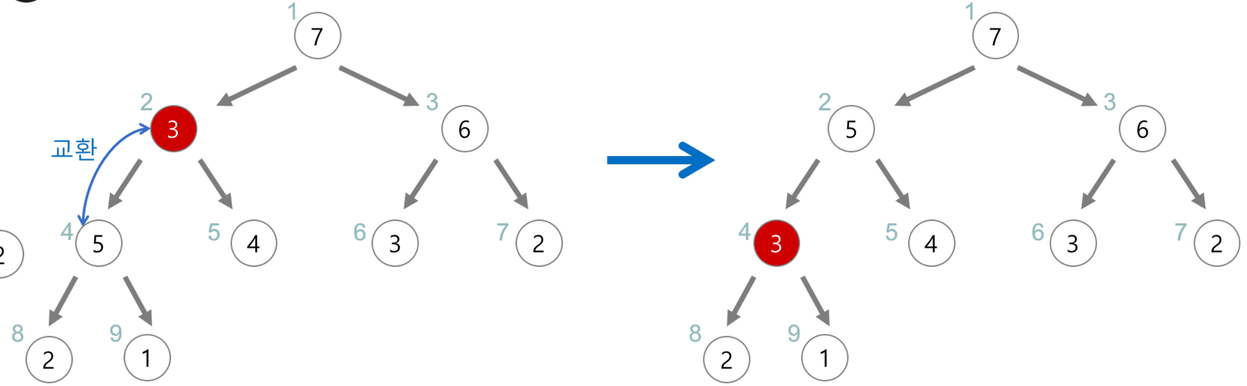
1 최댓값인 루트 노드 9를 삭제. (빈자리에는 최대 힙의 마지막 노드를 가져온다.)



2 삽입 노드와 자식 노드를 비교. 자식 노드 중 더 큰 값과 교환. (자식 노드 7 > 삽입 노드 3 이므로 서로 교환)



3 삽입 노드와 더 큰 값의 자식 노드를 비교. 자식 노드 5 > 삽입 노드 3 이므로 서로 교환

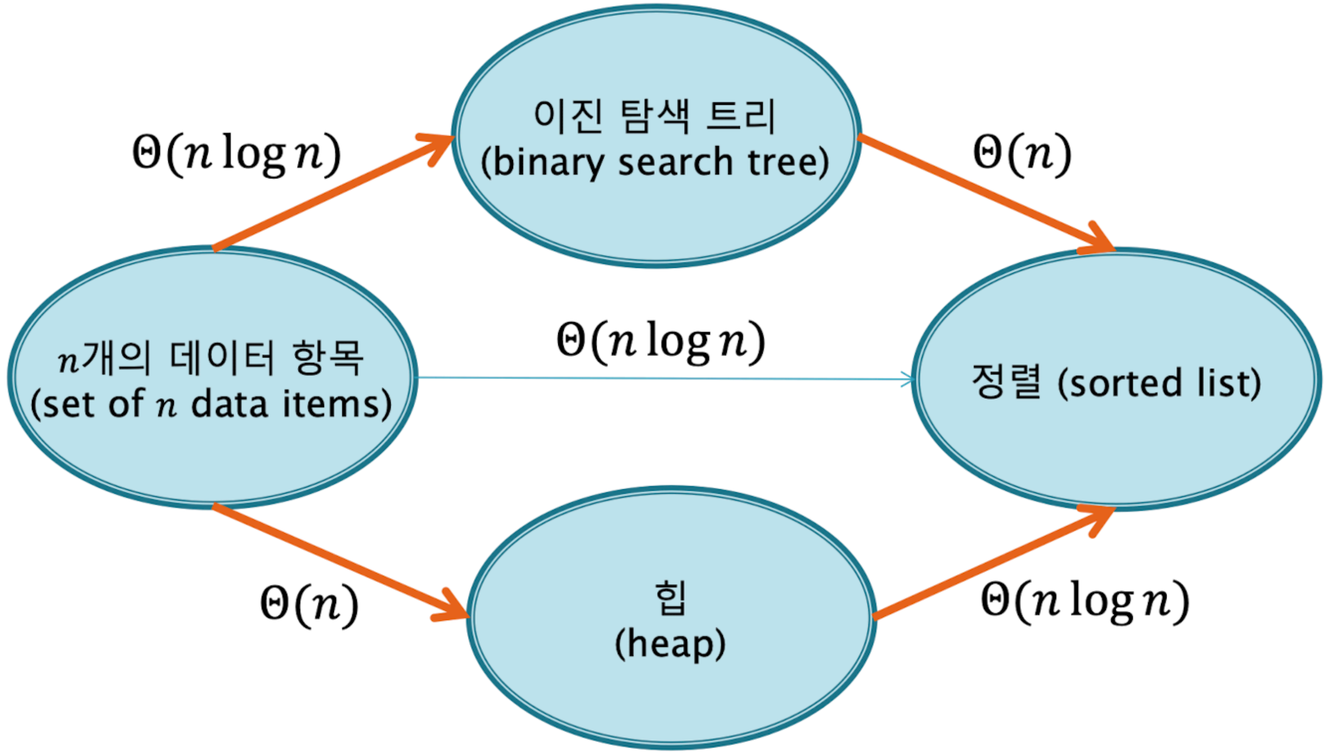


4 자식 노드 1, 2 < 삽입 노드 3 이므로 더 이상 교환하지 않는다.

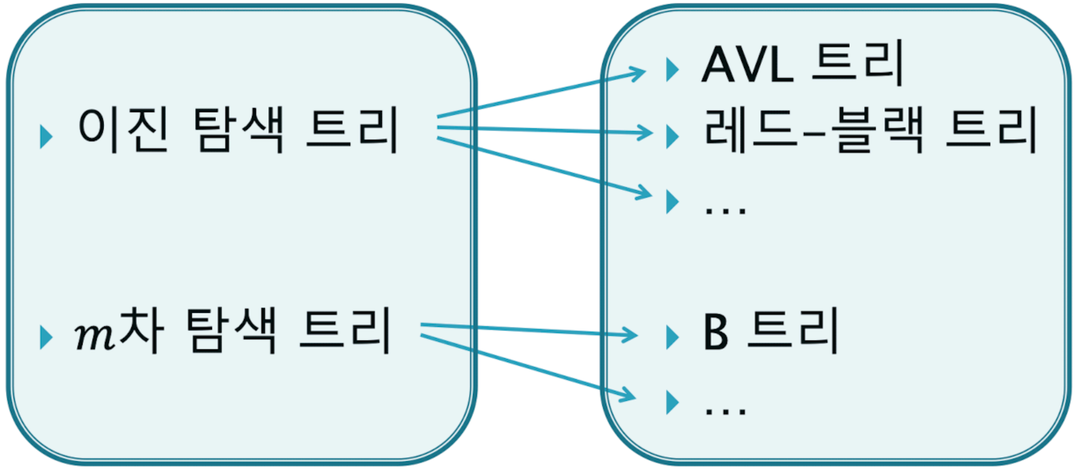
최댓값 삭제 연산

```c
// 최대 힙(max heap) 삭제 함수
element delete_max_heap(HeapType *h){
  int parent, child;
  element item, temp;

  item = h->heap[1]; // 루트 노드 값을 반환하기 위해 item에 할당
  temp = h->heap[(h->heap_size)--]; // 마지막 노드를 temp에 할당하고 힙 크기를 하나
  parent = 1;
  child = 2;

  while(child <= h->heap_size){
    // 현재 노드의 자식 노드 중 더 큰 자식 노드를 찾는다. (루트 노드의 왼쪽 자식 노드(index
    if( (child < h->heap_size) && ((h->heap[child].key) < h->heap[child+1
      child++;
    }
    // 더 큰 자식 노드보다 마지막 노드가 크면, while문 중지
    if( temp.key >= h->heap[child].key ){
      break;
    }

    // 더 큰 자식 노드보다 마지막 노드가 작으면, 부모 노드와 더 큰 자식 노드를 교환
    h->heap[parent] = h->heap[child];
    // 한 단계 아래로 이동
    parent = child;
    child *= 2;
  }

  // 마지막 노드를 재구성한 위치에 삽입
  h->heap[parent] = temp;
  // 최댓값(루트 노드 값)을 반환
  return item;
}
```

트리, 힙



탐색 트리 **균형 탐색 트리**



풀어올 문제

- https://programmers.co.kr/learn/courses/30/lessons/42627
- 프로그래머스 - 디스크 컨트롤러

자료 출처

- https://gmlwjd9405.github.io/2018/08/12/data-structure-tree.html
- 울 교수님 자료…