ICPC Sinchon

2022 Winter Algorithm Camp

9. 트리

서강대학교 임지환

2022 Winter Algorithm Camp

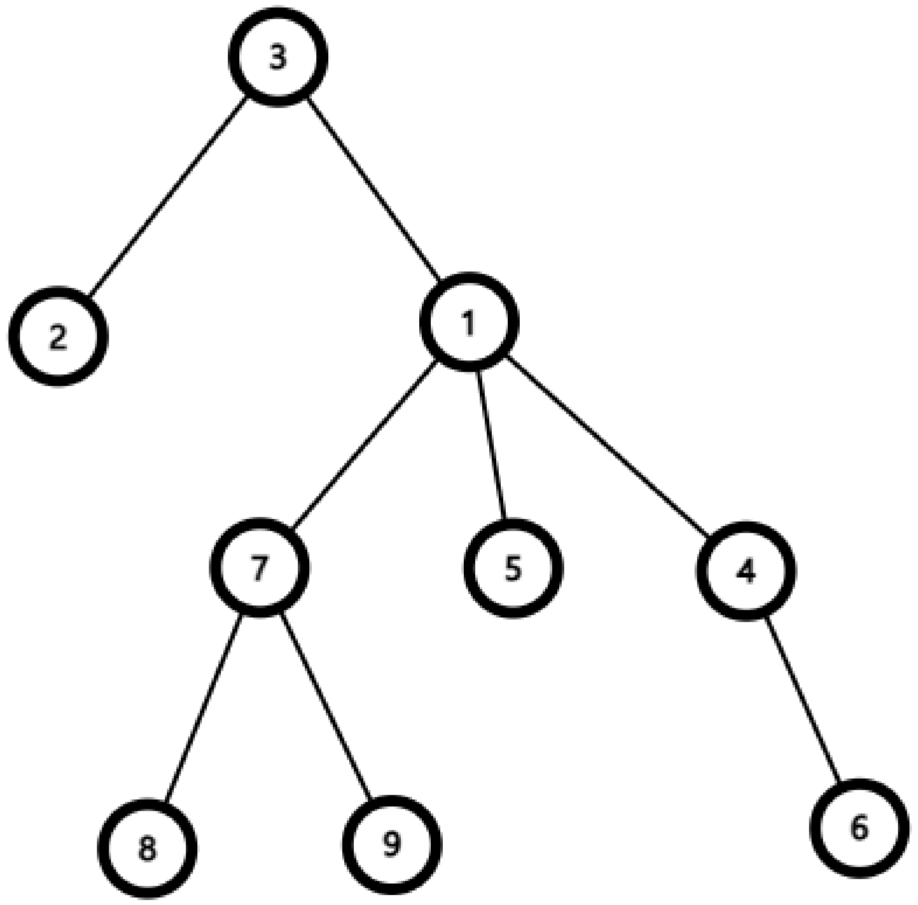
목차

240

Tree

* 정의

- Connected acyclic graph
- 계층형 자료구조



Tree

* 정의(recursive)

- A finite set of one or more nodes such that:

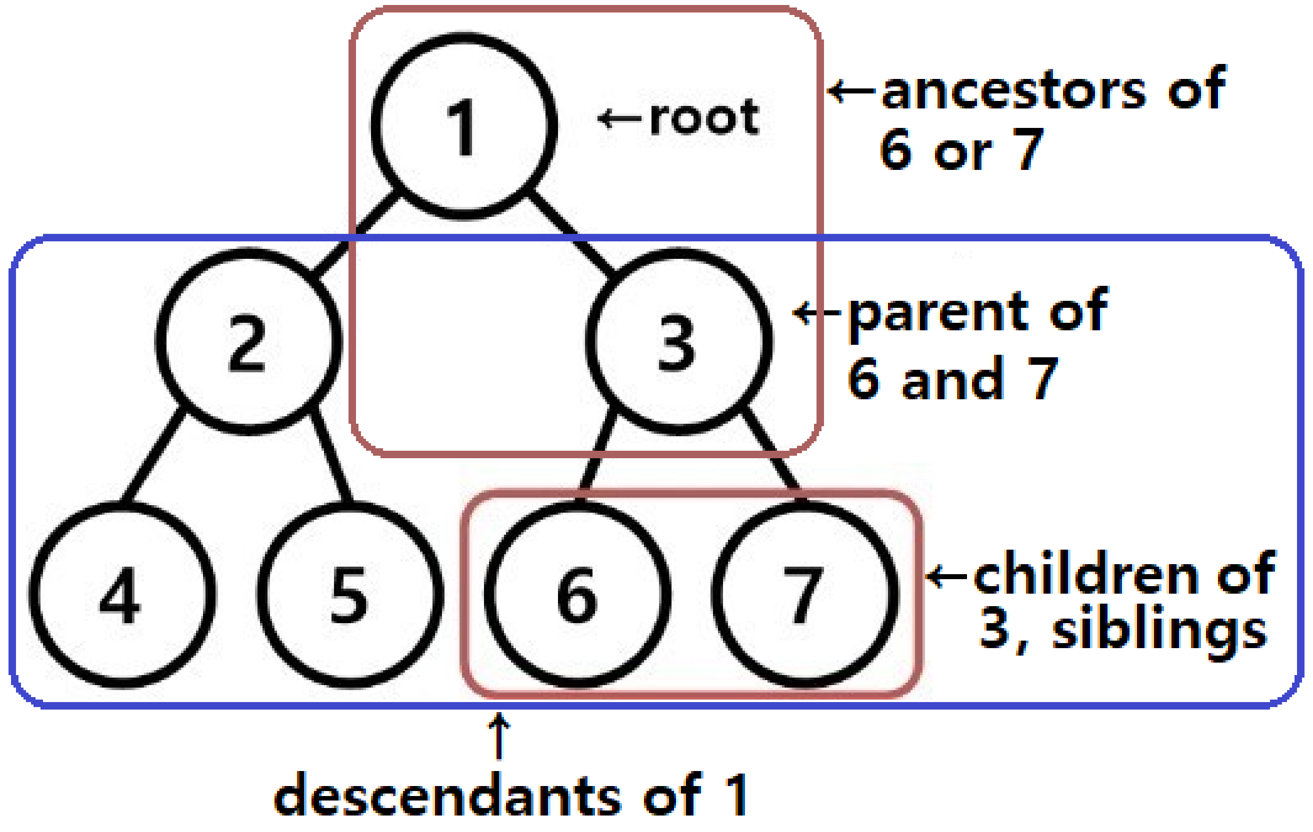
1) contains a specially designated node called root,
2) remaining nodes are partitioned into n(≥0) disjoint sets, $T_1, T_2, \ldots, T_n$, where each of these sets is a tree
3) $T_1, T_2, \ldots, T_n$ are called the subtrees of the root

Tree

* 용어 (At rooted tree)

상하(계층구조)가 정의되었을 때

- 부모(자식): 연결된 두 노드 중 상위(하위) 노드
- 형제: 부모 노드가 같은 두 노드의 관계
- 조상: 부모 노드들의 집합
- 자손: 자손 노드들의 집합
- 리프 노드: 자식이 없는 노드
- 서브 트리: 어떤 노드 t와 그 자손들로 구성된 트리
- 포레스트: 트리가 여러 개!



Tree

* 성질

- 간선의 개수 = 정점의 개수 - 1
- 임의의 서로 다른 두 정점 사이의 경로가 유일
- 각 노드들이 서로 다른 자식을 가짐
- 재귀적인 구조
- ...

Tree

* 구현

- 트리도 그래프입니다.
- 간선의 개수는 $|V| - 1$임이 보장되기 때문에 sparse graph이고, 인접리스트로 표현할 수 있습니다.

* 순회

- 루트로부터 순회 시 방문 체크 배열을 대신 파라미터를 하나 더 써서 해결할 수 있습니다.

```
vector<int> adj[mxn];

void dfs(int cur, int par){
  for(int &nxt:adj[cur]){
    if(nxt == par) continue;
    // can do sth
  }
}
```

15681. 트리와 쿼리

무방향 무가중치, 루트 있는 트리

N개의 정점과 그에 대한 간선 정보, 쿼리의 수 Q, 루트 노드의 번호 R($2 \le N \le 10^5$, $1 \le Q \le 10^5, 1 \le R \le N$)

query(x): x를 루트로 하는 서브트리에 속한 정점의 수

15681. 트리와 쿼리

1. 재귀적인 방식의 문제 해결

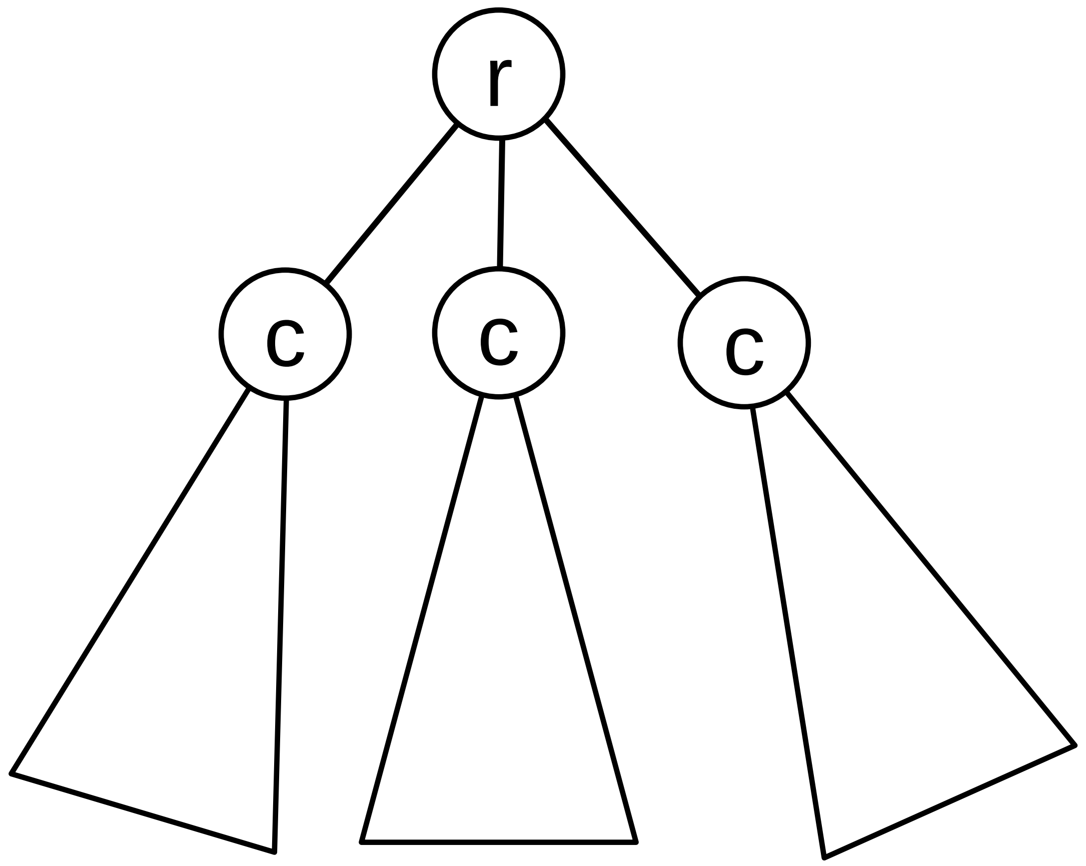
노드 T를 루트로 하는 서브트리의 크기를 sz(T)라 할 때,

노드 r과 자식들 c에 대하여 $sz(r) = sum(sz(c))$

```
sz(cur, prv) :
    cnt[cur] <- 1;
    for nxt in adj[cur]:
        if nxt == prv continue
        cnt[cur] += sz(nxt, cur)

    return cnt[cur]

```



15681. 트리와 쿼리

2. 시간복잡도

모든 노드들에 대한 서브트리 크기를 $O(|V| + |E|)$에 전처리

쿼리 당 $O(1)$, 최종 $O(N + Q)$

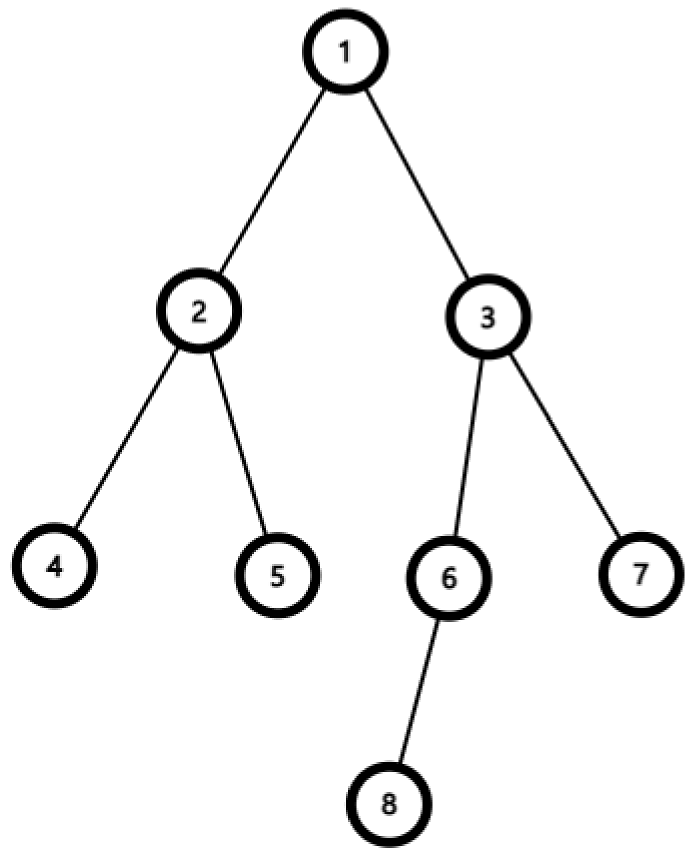
Binary Tree

* 정의

모든 정점이 2개 이하의 자식을 가지는 트리

* 성질

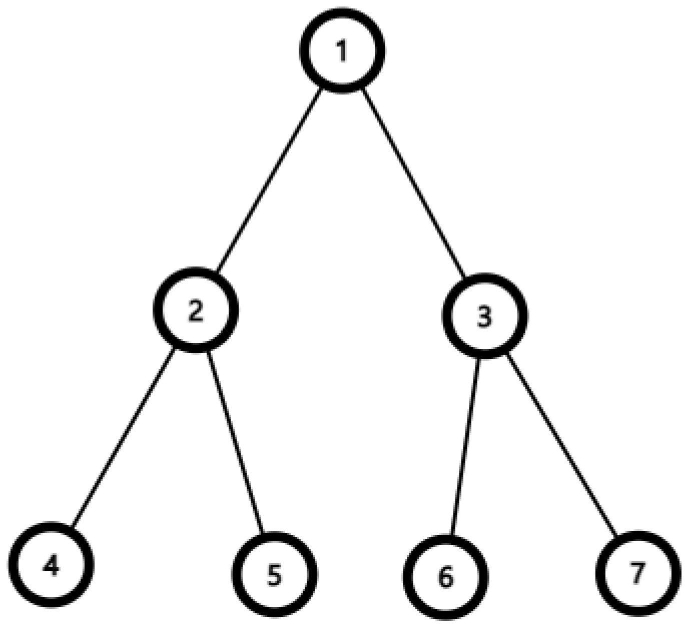
- 자식이 2개이기 때문에 방향이 정의된다. (Left, Right)
- 높이가 h인 이진트리의 최대 노드 수는 $2^h - 1$개이다.



Binary Tree

* 포화 이진 트리(Full binary tree)

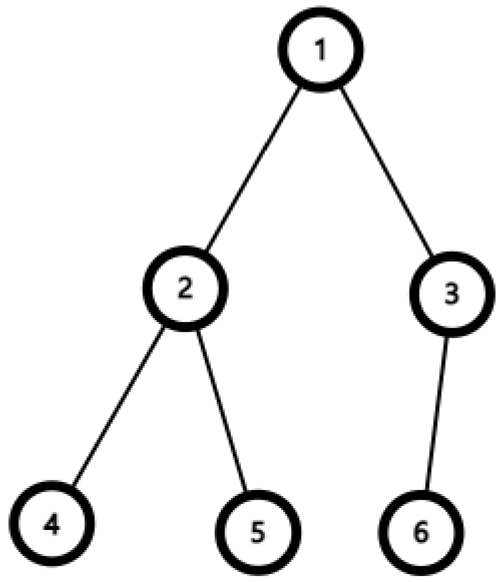
리프 노드를 제외한 모든 정점의 자식이 2개인 트리



* 완전 이진 트리(Complete binary tree)

마지막 레벨을 제외하고 모든 레벨이 완전히 채워져 있으며,

마지막 레벨은 왼쪽부터 채워진 트리



- 위쪽, 왼쪽부터 차례로 index를 1, 2, 3, …으로 부여했을 때
  부모와 자식의 index 관계 : $i \rightarrow (2i, 2i+1)$

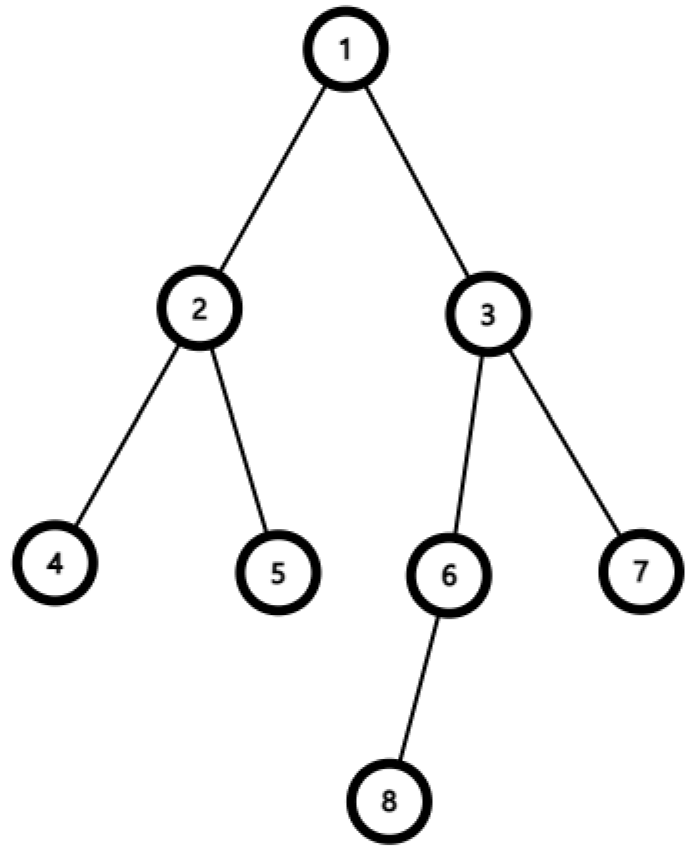
Binary Tree

* 이진 트리 순회

전위 순회(Preorder): 루트 – 왼쪽 서브트리 – 오른쪽 서브트리 순

중위 순회(Inorder): 왼쪽 서브트리 – 루트 – 오른쪽 서브트리 순

후위 순회(Postorder): 왼쪽 서브트리 –오른쪽 서브트리 – 루트 순




```
inorder(par) :
    if left_child exists:
        inorder(left_child)
    print(par)
    if right_child exists:
        inorder(right_child)
```

Priority queue

* 우선순위 큐

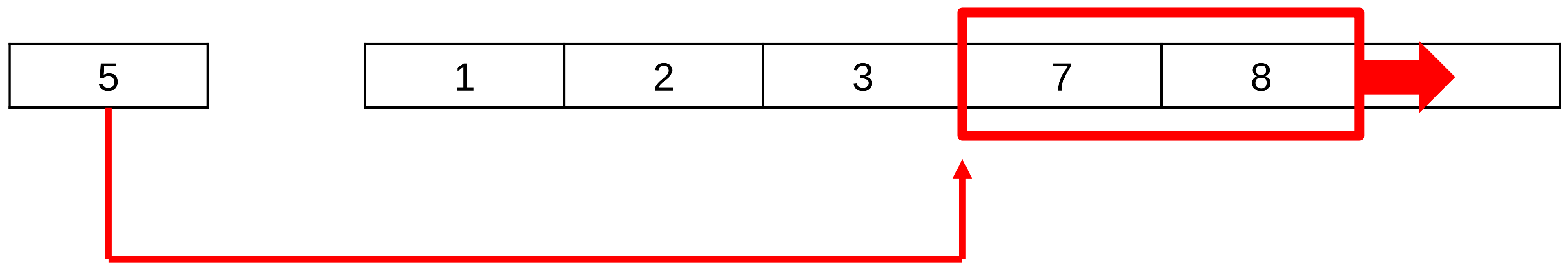
입력된 순서에 상관없이, **우선순위가 가장 높은 자료가 가장 먼저 꺼내지는** 자료구조

* Basic operations

- Top: pq에 있는 원소 중 우선순위가 가장 높은 원소 반환
- Push: pq에 원소 추가
- Pop: Top에 있는 원소 제거

Priority queue

* 우선순위 큐 구현..?



Priority queue

* 우선순위 큐 구현..?

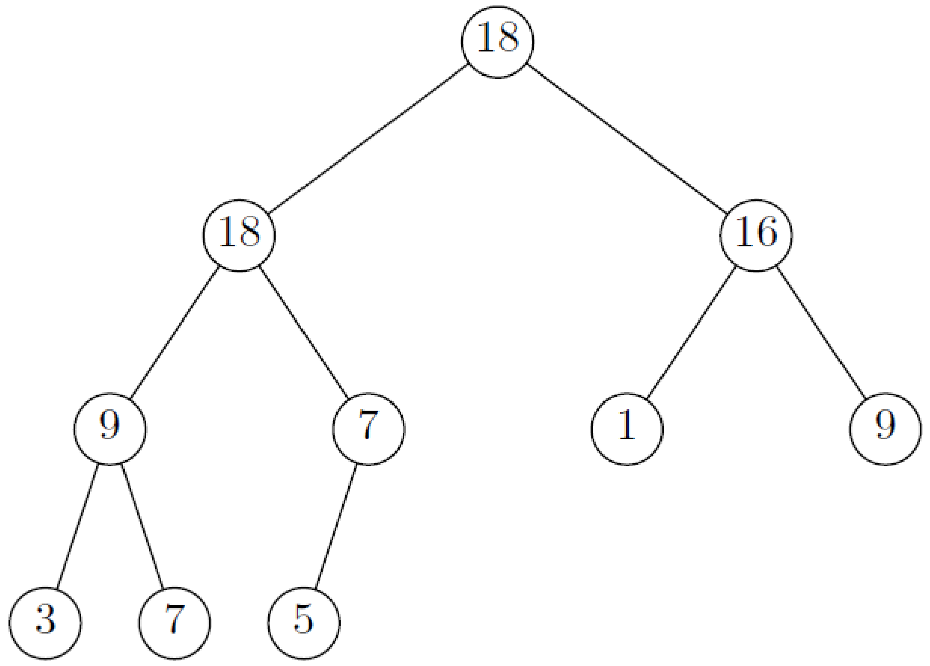
5

1	2	3	**5**	**7**	**8**

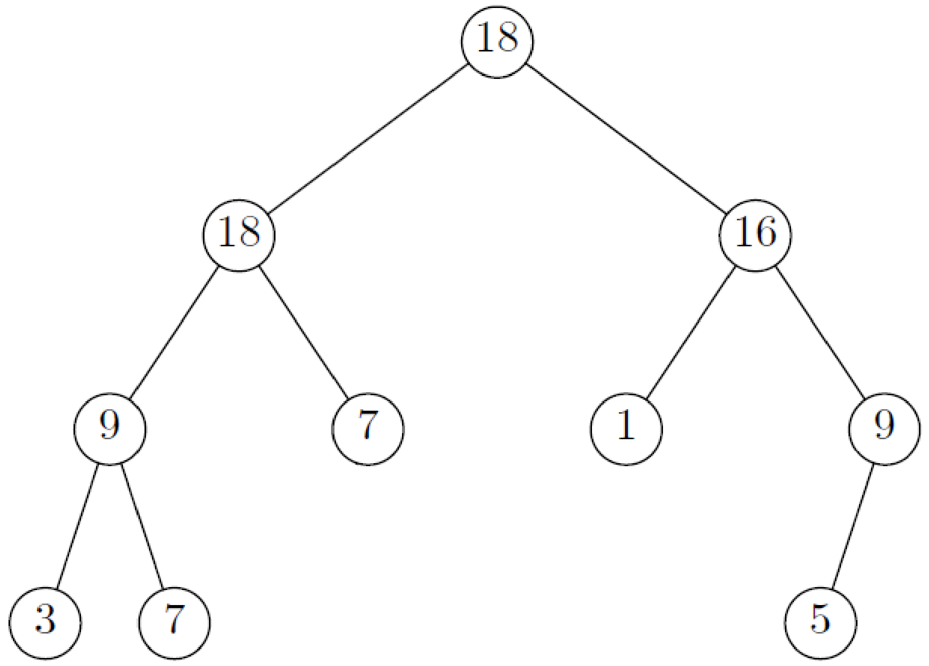
Heap – Partially Ordered Trees

* 정의와 속성

- 우선순위가 부여된 노드로 구성된 트리에 대하여
- 모든 서브트리의 루트 노드의 우선순위는 해당 서브트리의 어떤 노드들보다도 우선순위가 높다.
- Balanced POT: 완전 이진트리



Balanced POT

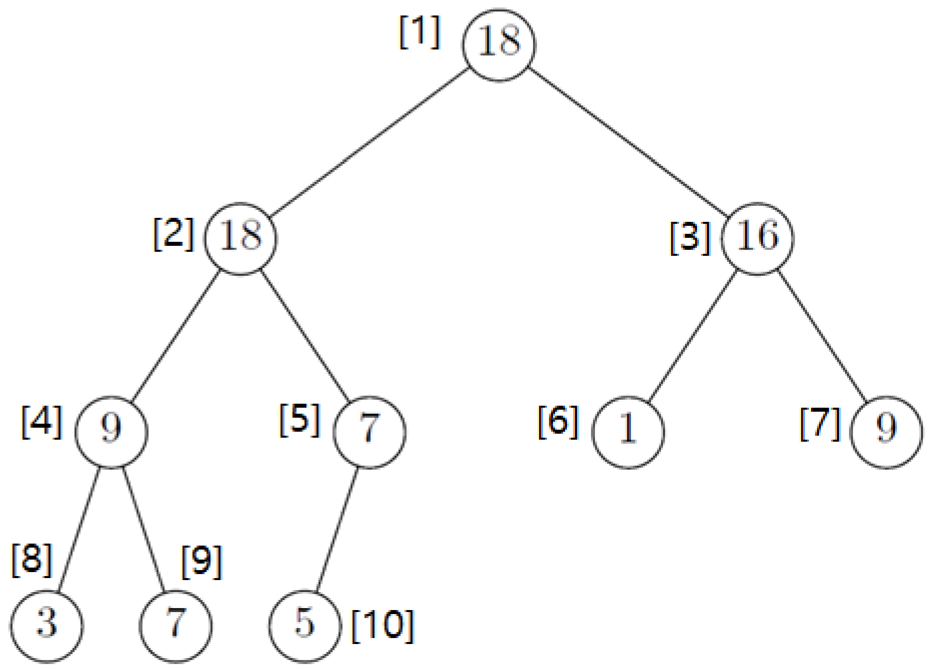


Unbalanced POT

Heap – Partially Ordered Trees

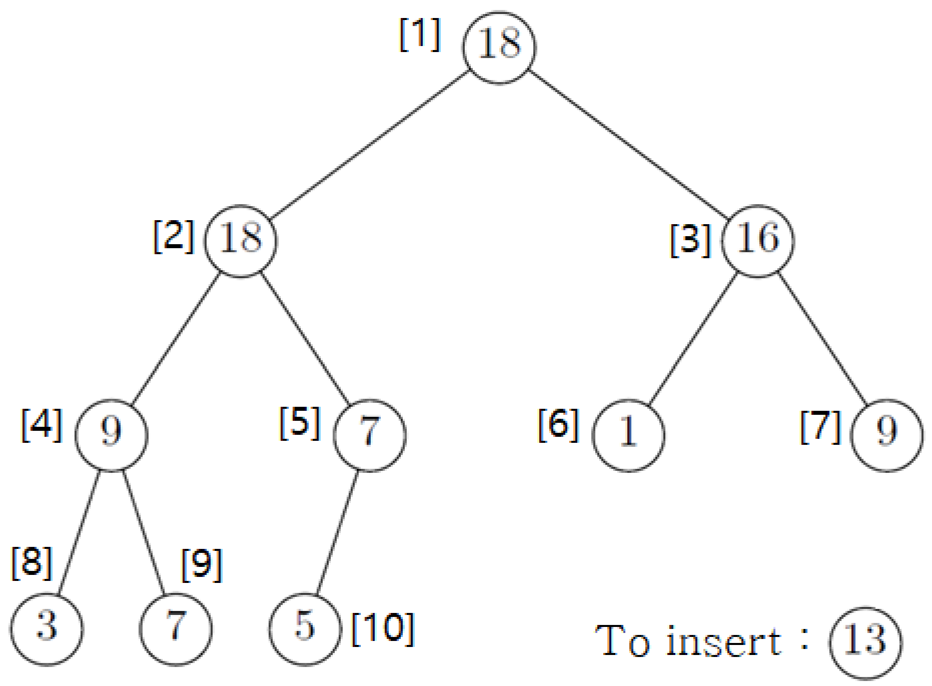
* Balanced POTs with array

1	2	3	4	5	6	7	8	9	10
18	18	16	9	7	1	9	3	7	5



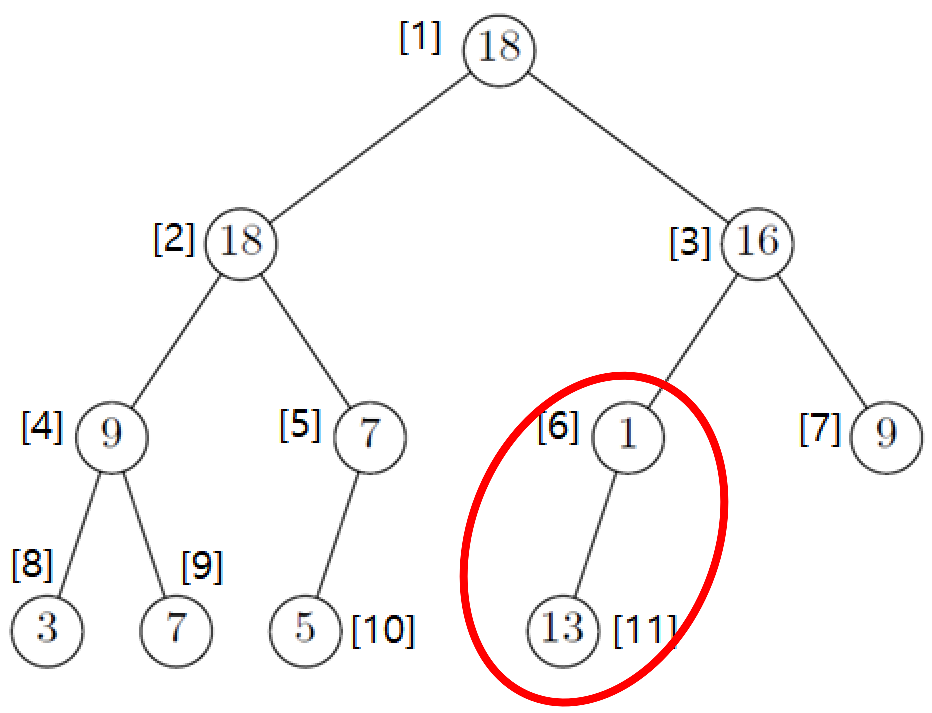
Heap – Operations

* Insertion



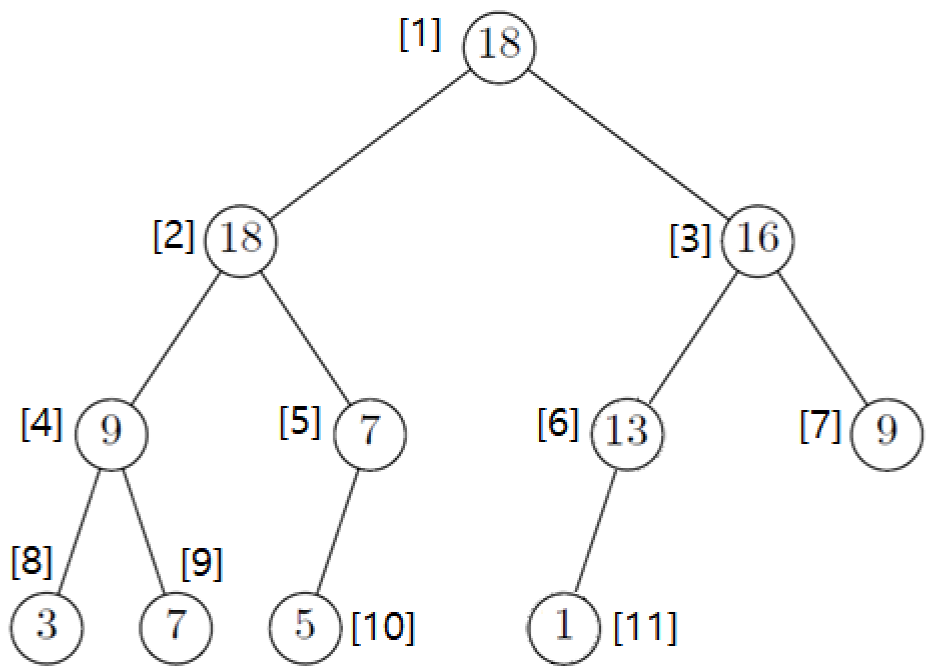
Heap – Operations

* Insertion



Heap – Operations

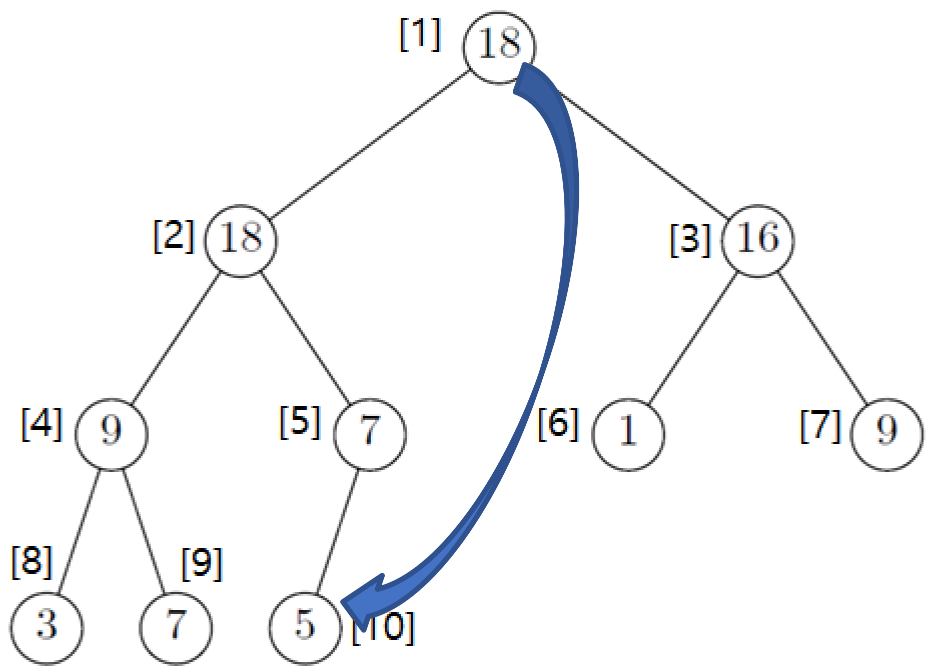
* Insertion



240

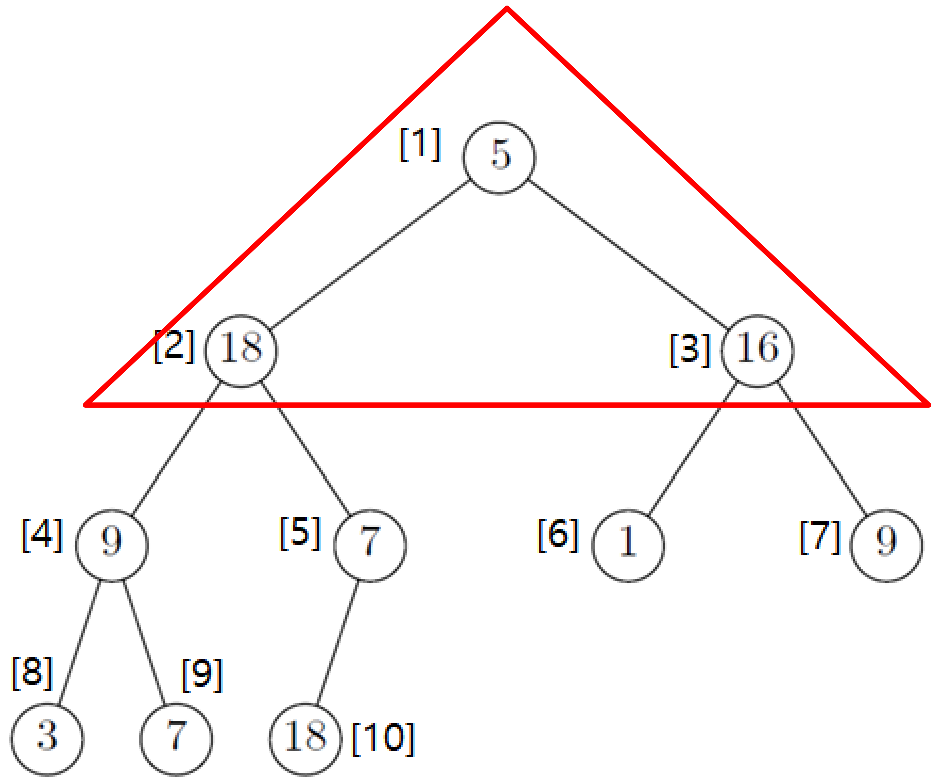
Heap – Operations

* Deletion



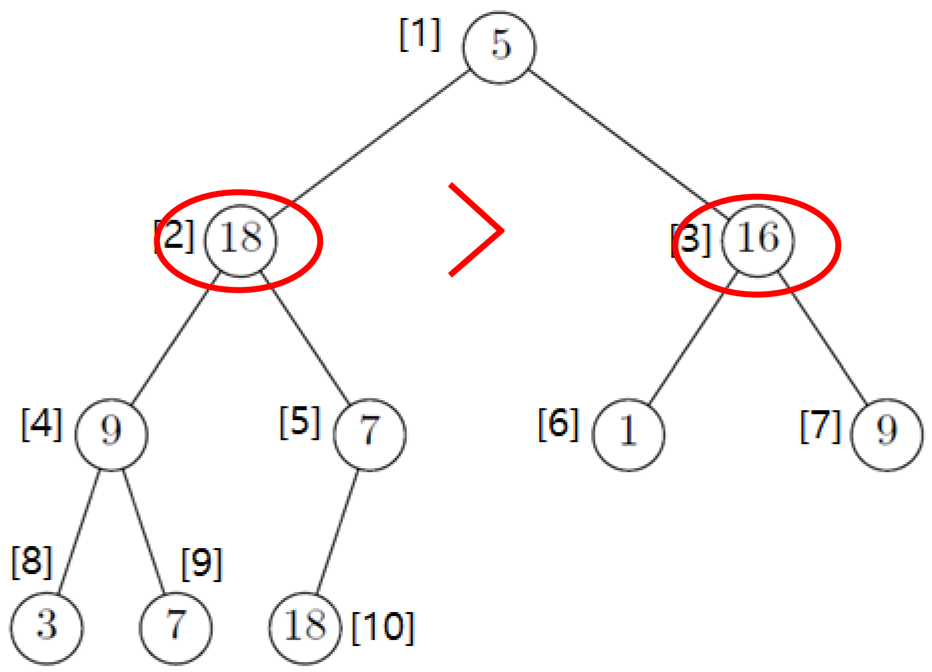
Heap – Operations

* Deletion



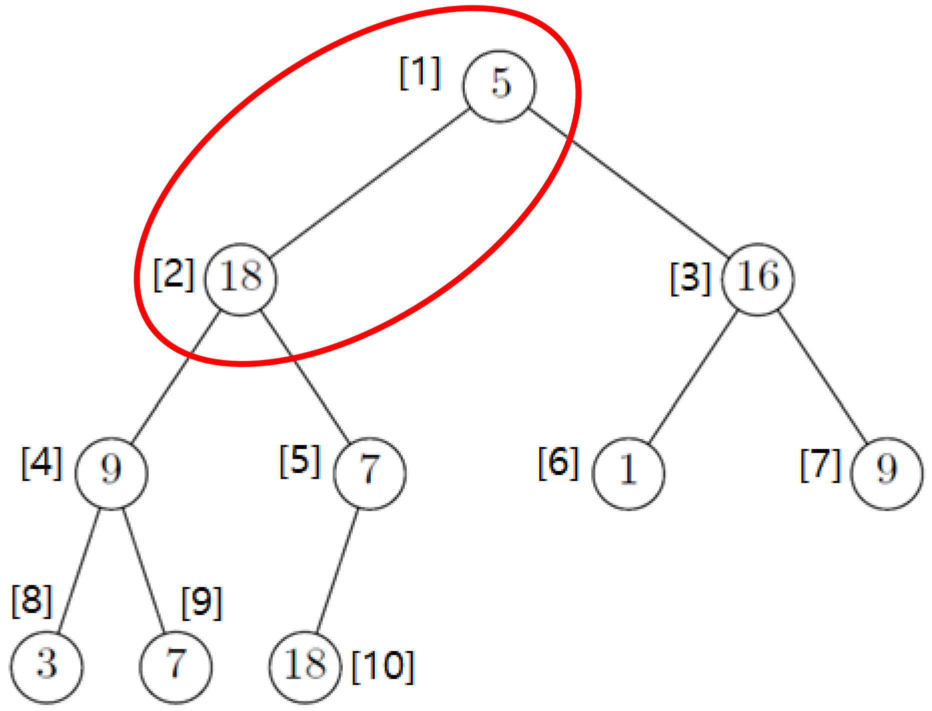
Heap – Operations

* Deletion



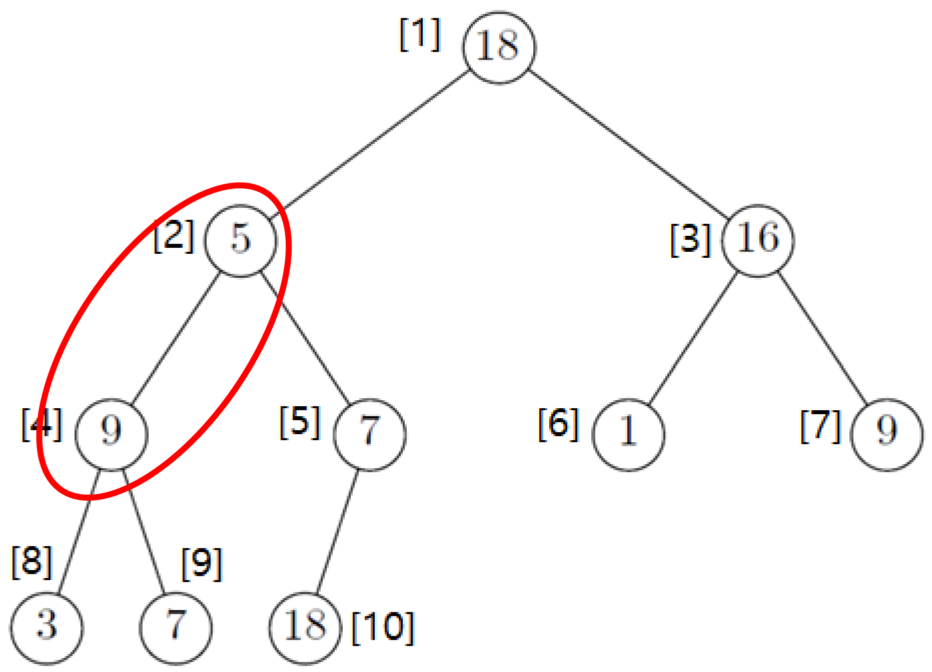
Heap – Operations

* Deletion



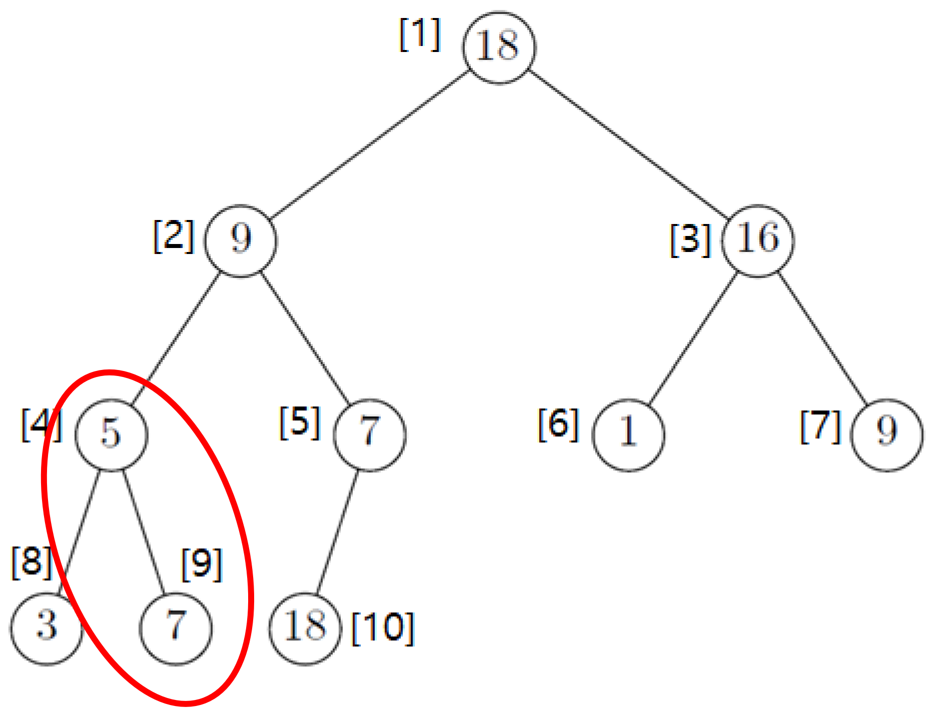
Heap – Operations

* Deletion



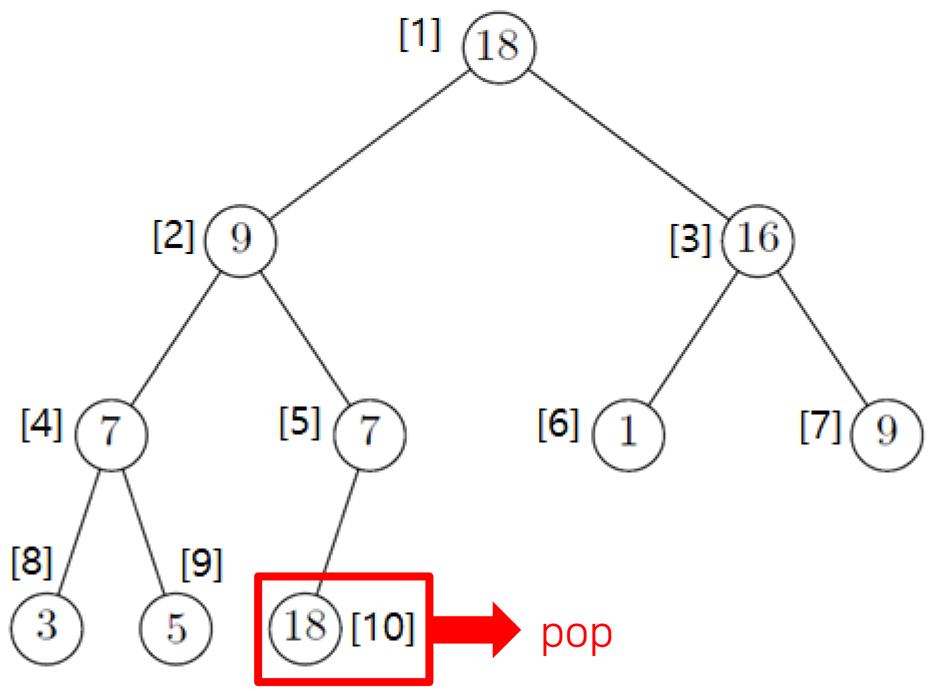
Heap – Operations

* Deletion



Heap – Operations

* Deletion



Heap – Time Complexity

* Time Complexity

- 트리의 높이: $O(logN)$
- Insertion시 참고하는 원소의 개수: $O(logN)$
- Deletion시 참고하는 원소의 개수: $O(logN)$
- $O(1)$ per Top access, $O(logN)$ per Insertion & Deletion

Heap – using standard template library

* std::priority_queue (link)

- Basically max heap
- to use min heap:

```
std::priority_queue<int, std::vector<int>, std::greater<int>> pq;
```

- 또는

```
std::priority_queue<int> pq;

...
// to push 1, 2, 3:
pq.push(-1);
pq.push(-2);
pq.push(-3);
```

15942. Thinking Heap

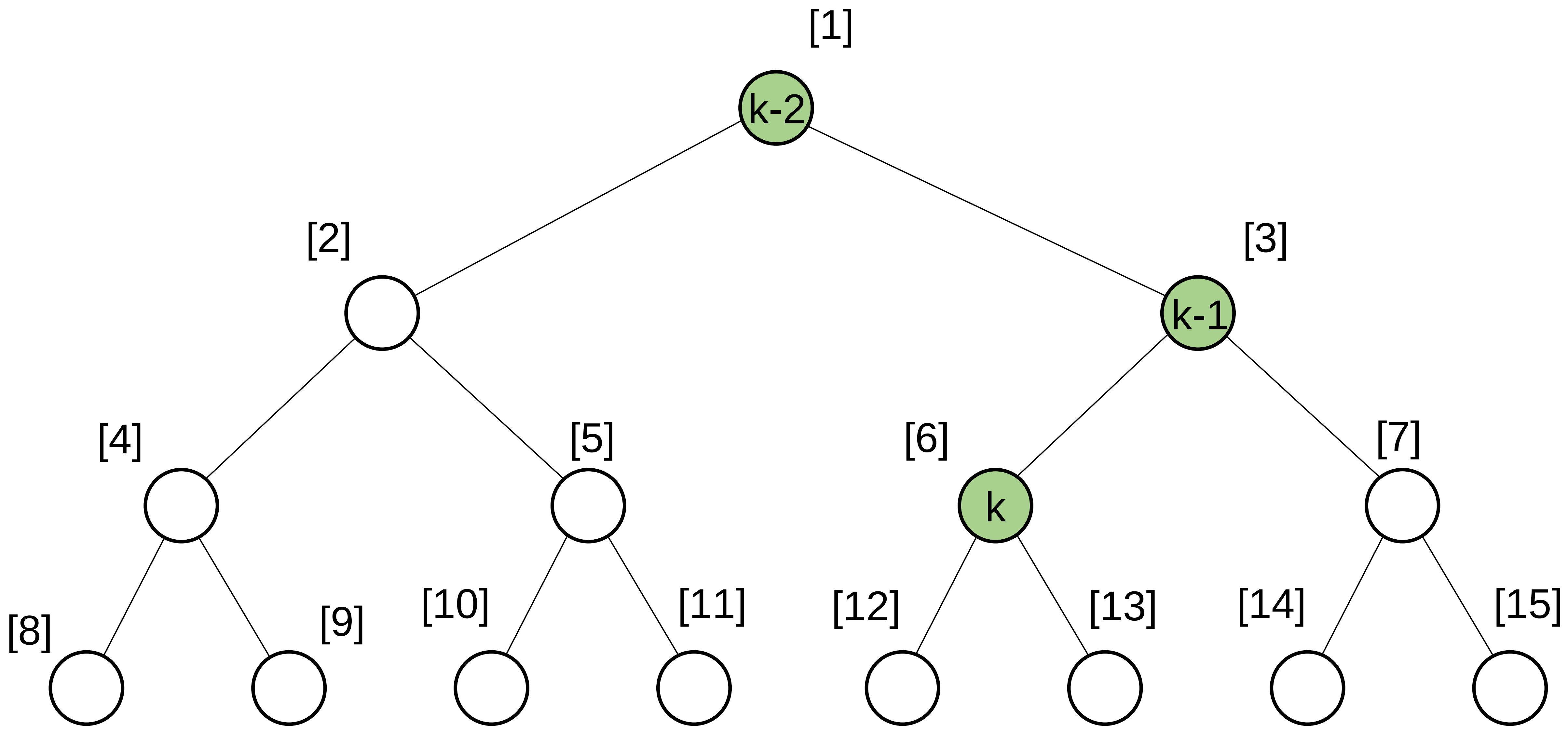
1~N인 서로 다른 자연수 N개의 원소를 Binary min heap를 넣으려 한다.$(1 \leq N \leq 200{,}000)$

N개의 자연수를 전부 넣은 후 자연수 k가 heap 배열의 p번째 위치하도록 하기 위한 원소 삽입 순서를 구해보자.

15942. Thinking Heap

1. p번째 위치의 조상들의 경우

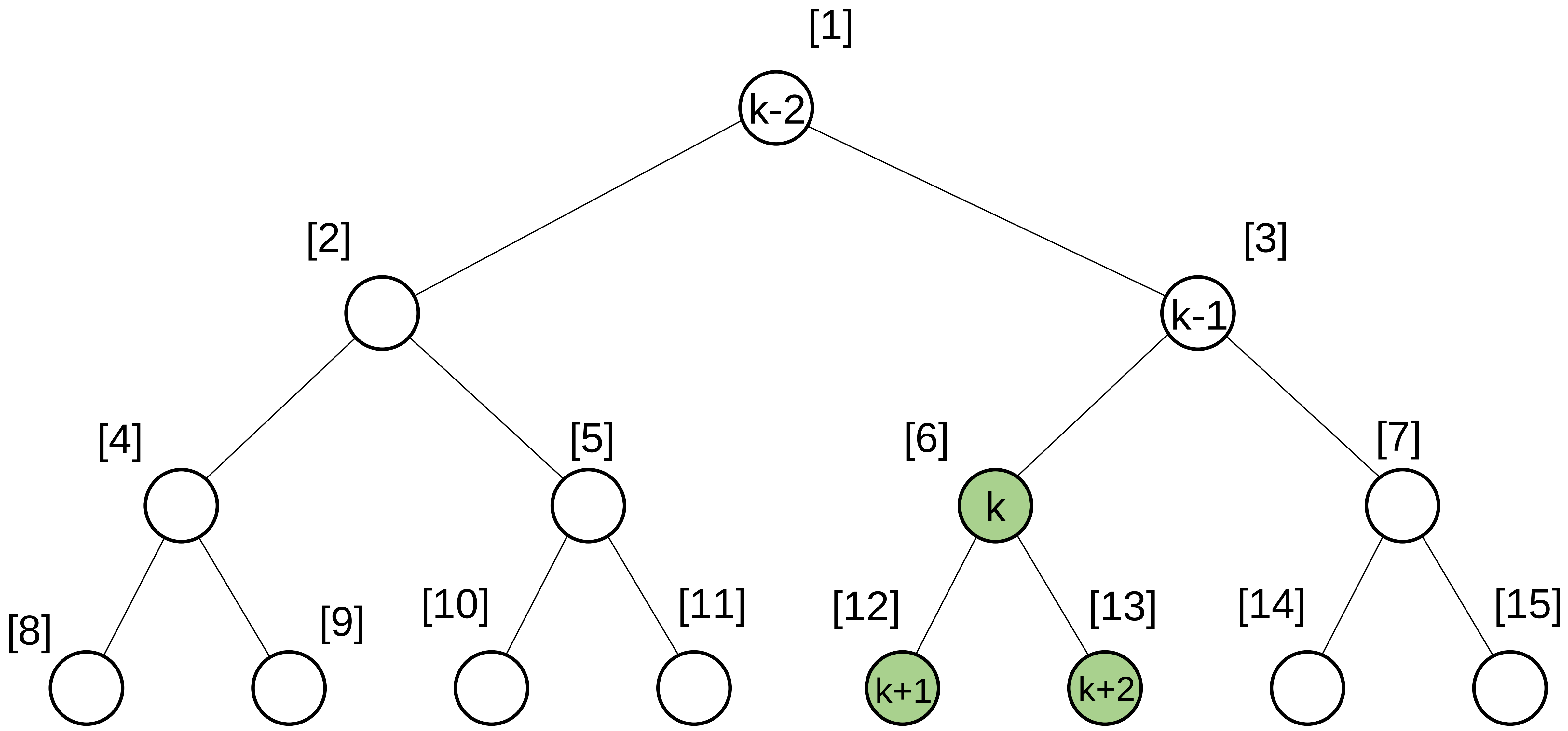
k미만의 수들로만 구성



15942. Thinking Heap

2. p번째 위치의 자손들의 경우

k초과의 수들로만 구성



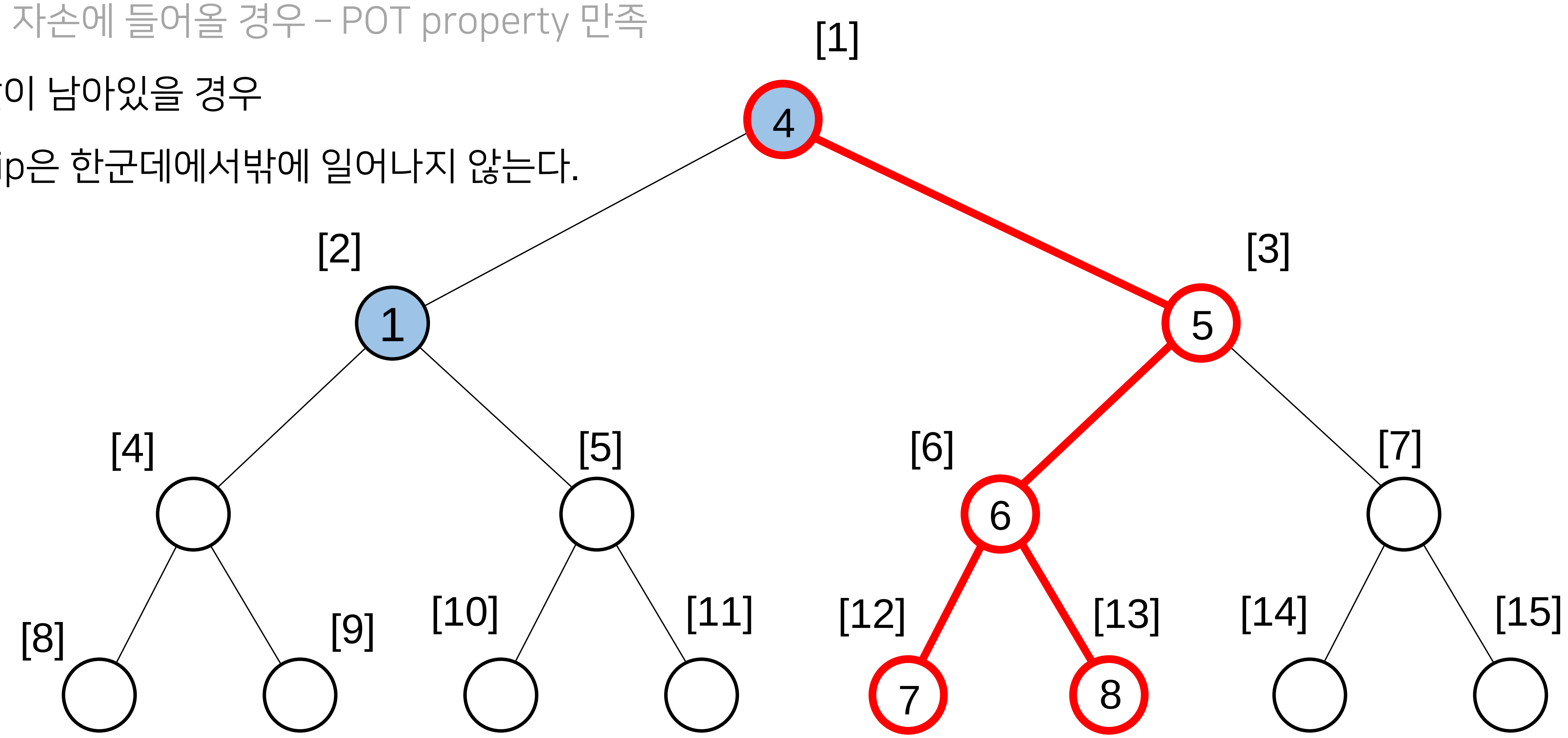
15942. Thinking Heap

3. 그 외 자리들은?

p번째 & p번째의 조상들 & p번째의 자손들만 flip없이 잘 고정이 된다면

1) 고정된 값보다 큰 값이 자손에 들어올 경우 – POT property 만족
2) 고정된 값보다 작은 값이 남아있을 경우

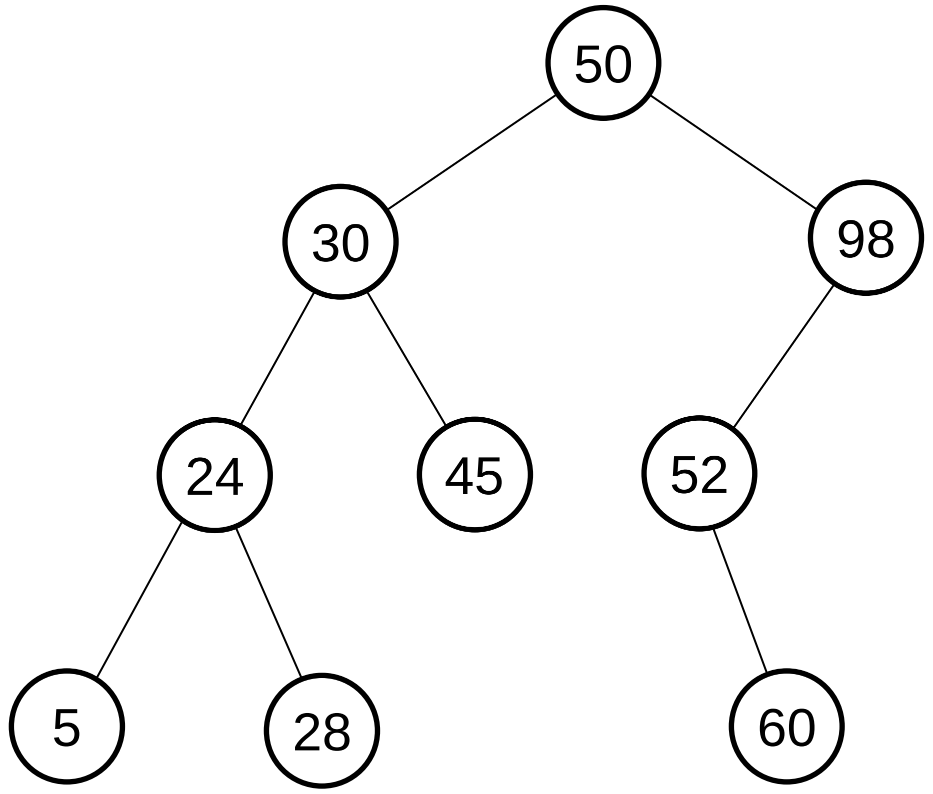
-> 고정된 값들과의 flip은 한군데에서밖에 일어나지 않는다.



Binary Search Tree

* 소개

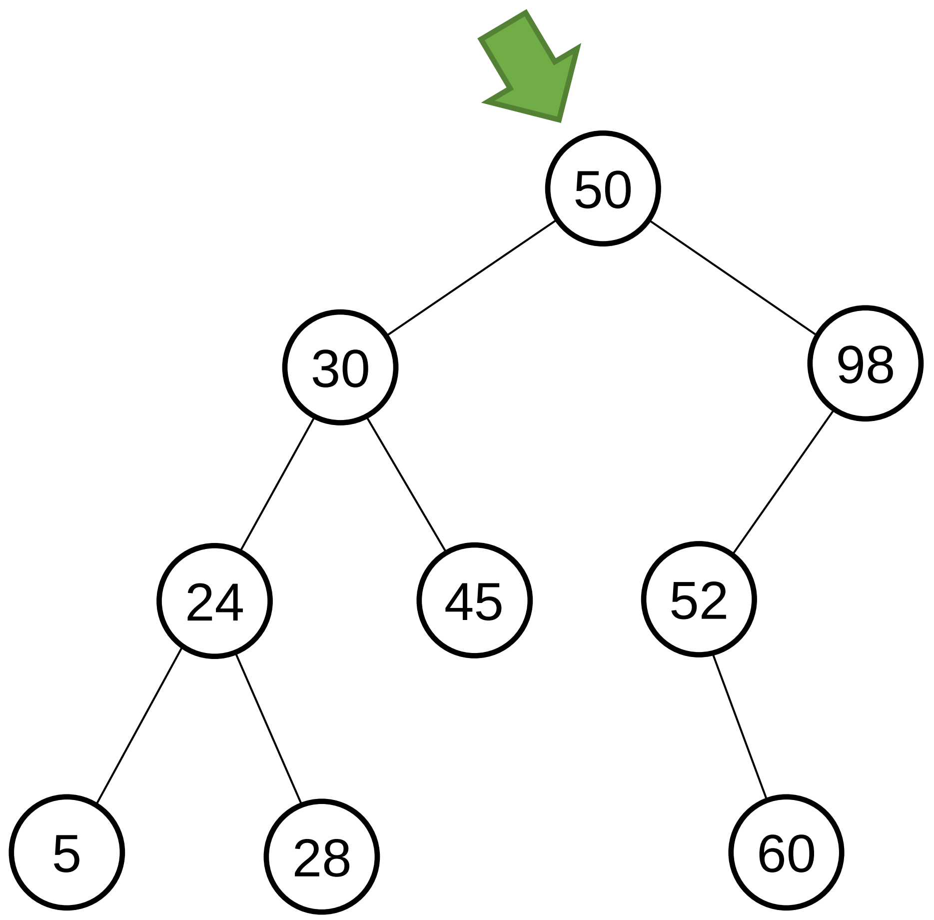
- 탐색형 자료구조로서의 트리 활용
- 왼쪽 서브트리에는 루트보다 작은 원소들을,
  오른쪽 서브트리에는 루트보다 큰 원소들을 배치시켜 특정 원소의 존재 여부 탐색



Binary Search Tree

ex) find 28,

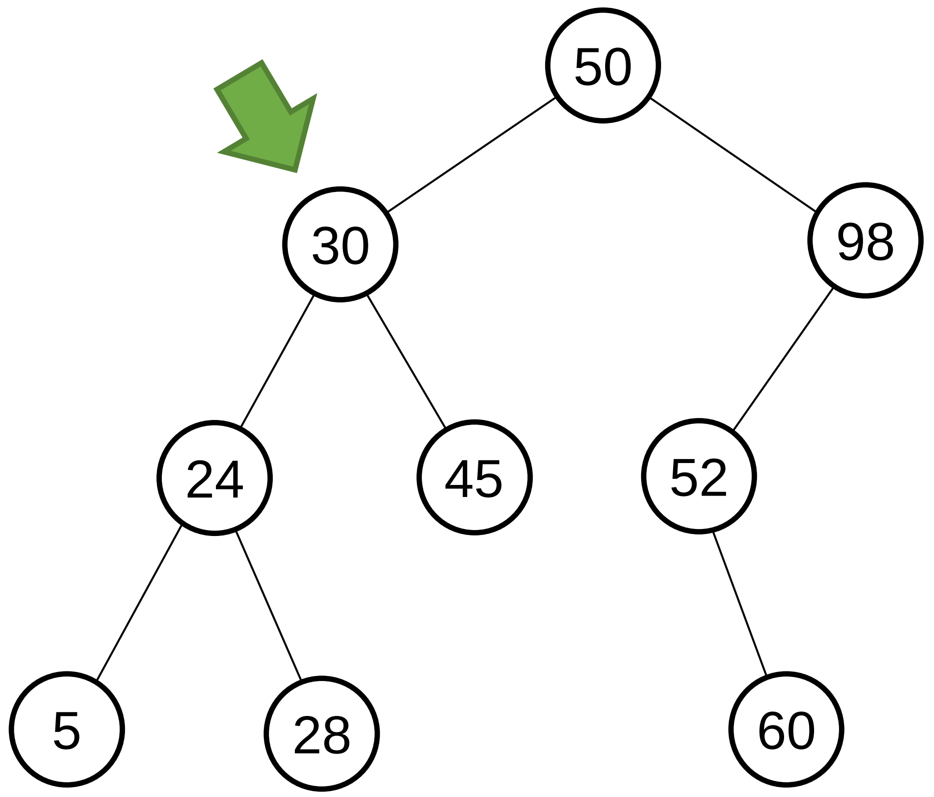
28 < 50, go left



Binary Search Tree

ex) find 28,

28 < 30, go left

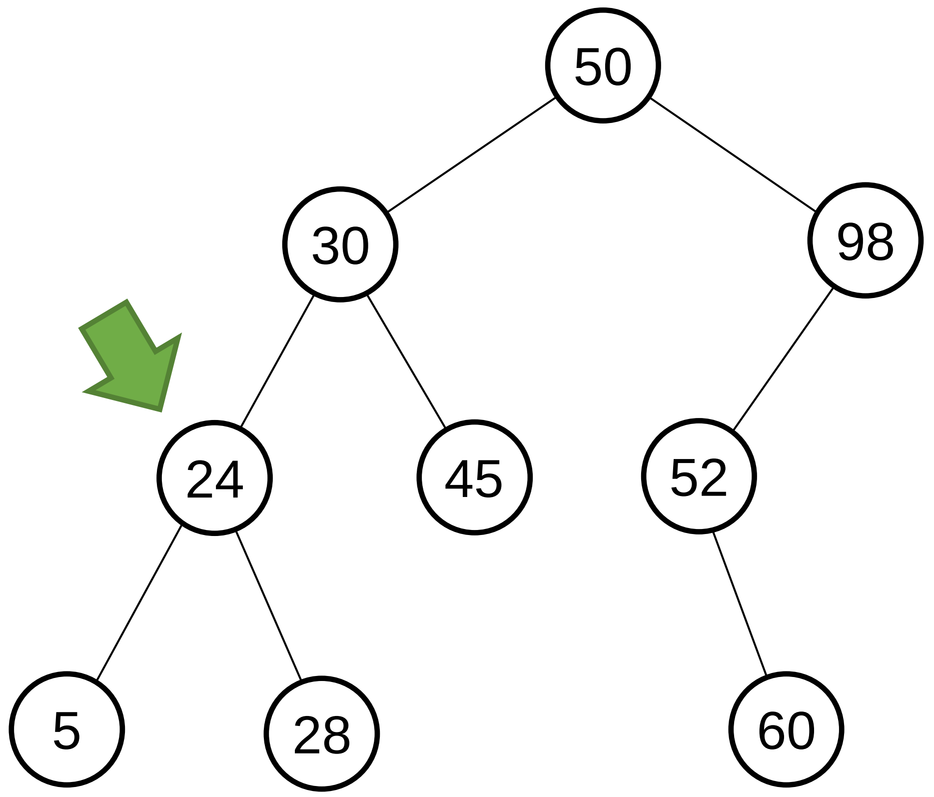


240

Binary Search Tree

ex) find 28,

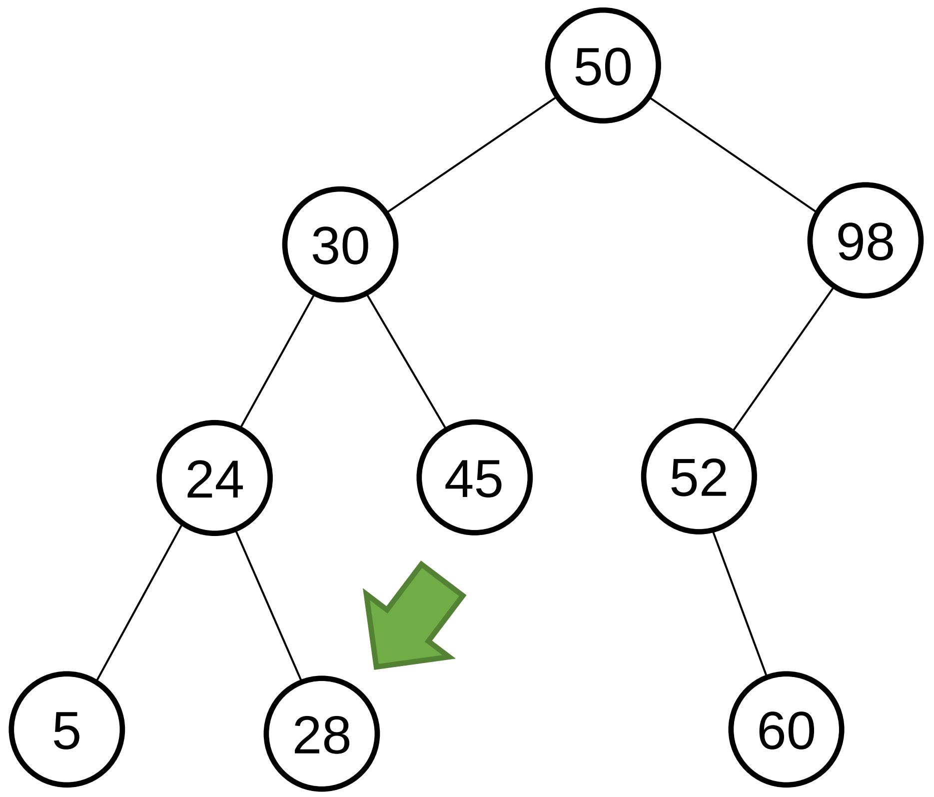
28 > 24, go right



Binary Search Tree

ex) find 28,

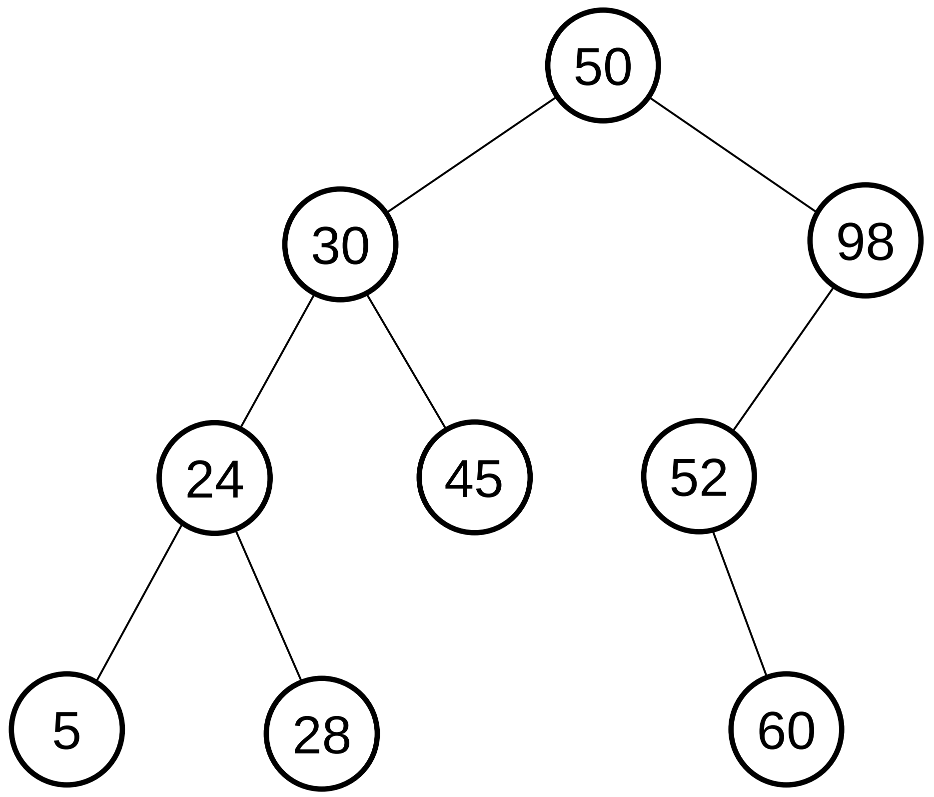
28 found!



Binary Search Tree

* 이진 검색 트리 만들기

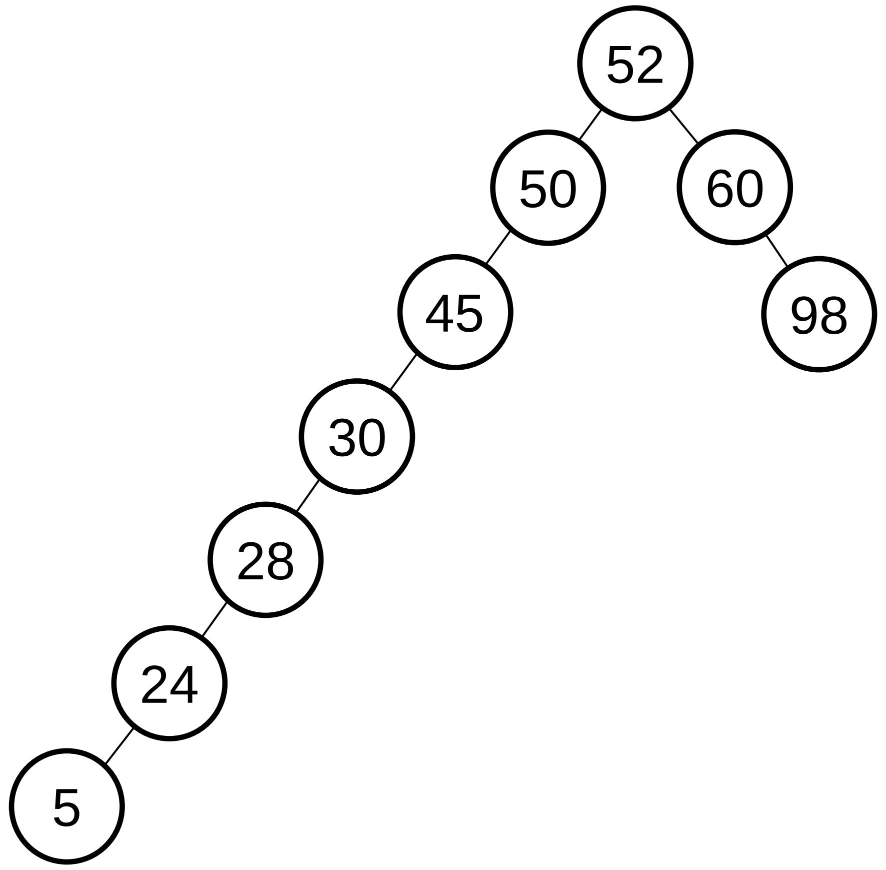
- 탐색 과정과 비슷하게
  넣고자 하는 원소가 작으면 왼쪽 서브트리에,
  크면 오른쪽 서브트리에 삽입



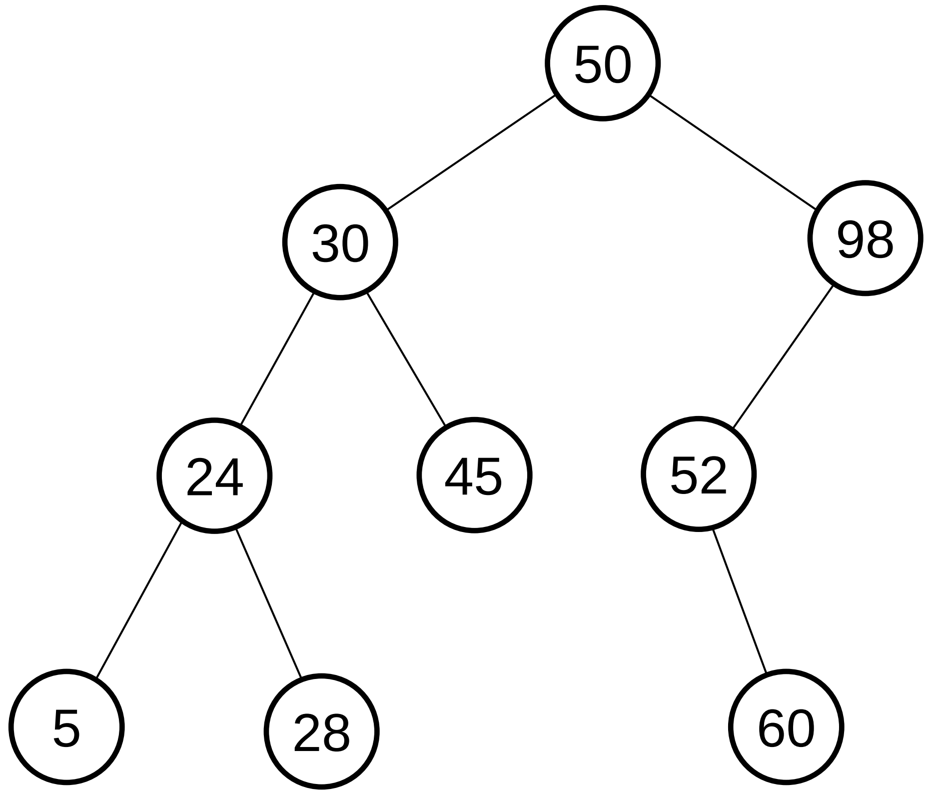
Binary Search Tree

* 이진 검색 트리 만들기

- 넣는 순서에 따라 생김새가 달라질 수 있습니다.



[98, 60, 52, 50, 45, 30, 28, 24, 5]



[50, 30, 24, 5, 98, 28, 45, 52, 60]

Binary Search Tree – Balanced BST

* 시간복잡도

- 넣는 순서에 따라 높이가 달라지고, 높이에 따라 탐색 시간이 달라진다.
  최악 $O(N)$, 평균 $O(logN)$이 걸린다.

* 균형잡힌 이진검색트리(Balanced bbst)

- 입력 순서에 따라 상관없이 트리의 높이를 최대한 낮게 만들어줄 필요가 있다.
- 일부 bst는 자가 균형을 맞추는 방식으로 이를 해결한다.(ex: Red-Black Tree, AVL Tree, splay tree, etc)

Binary Search Tree – using STL

* `std::set, std::map`

- Red-Black Tree를 기반으로 만들어진 대표적인 탐색형 자료구조
- 탐색과 삽입, 삭제에 평균 $O(logN)$의 시간복잡잡도가 들지만 상대적으로 느린 vusb
- 문자열 key의 map, unordered_map 성능비교

Appendix – Additional Topics

* Diameter of tree

트리에서의 지름은 **가중치 있는 트리에서 가장 먼 두 점 사이의 거리**를 의미합니다.

두 점은 리프 노드에서 나오며, 임의의 점에서 dfs를 수행하여 해당 점으로부터

가장 먼 점을 구하고(u), u로부터의 dfs를 통해 가장 먼 점(v)을 구하면 (u, v)상의

경로의 길이가 트리의 지름을 의미합니다.

자세한 증명은 여기에서 보실 수 있습니다.

트리 상에서의 반지름, 이심률 등도 정의될 수 있습니다.

Appendix – Additional Topics

* Dynamic programming on trees

트리의 재귀적인 구조로 인해 전체 트리와 서브트리의 관계에 동적계획법이 적용될 수도 있습니다. 전체 트리에 대한 최적화 문제를 서브트리(부분문제)들의 결과를 이용하는 방식으로 문제를 해결합니다.

여기에서 문제들을 보실 수 있습니다.

Appendix – Additional Topics

* Lowest Common Ancestor (LCA)

루트가 있는 트리에서의 최소 공통 조상은 임의의 두 정점 간의 공통 조상들 중 가장 낮은 노드를 의미합니다. 일반적으로 LCA는 $O(logN)$에 구할 수 있으며 LCA를 구하는 알고리즘을 통해 두 정점 사이의 거리, 경로 상의 무언가(최대, 최소, etc)등을 구할 수 있습니다.

이전 강의자료는 여기에서 확인하실 수 있으며, 관련 문제는 여기에서 보실 수 있습니다.

240

Appendix – Problems

필수문제

14244	트리 만들기
15681	트리와 쿼리
15942	Thinking Heap
19598	최소 회의실 개수
5639	이진 검색 트리
1351	무한 수열

연습문제

11725	트리의 부모 찾기	1655	가운데를 말해요
1991	트리 순회	14425	문자열 집합
2644	촌수계산	20292	컨설팅
11279	최대 힙	21939	문제 추천 시스템 Version 1
7662	이중 우선순위 큐		